SAMS

22309

The Tool Kit Series VIC 20™ Edition

Ted Buchholz and Dave Dusthimer

THE TOOL KIT SERIES
VIC 20 EDITION

Dave Dusthimer is an avid home computer user and a self-taught programmer. As the father of three, Dave found himself teaching his children, as well as several children in the neighborhood, how to use and enjoy computers. A member of IEEE, Dave has edited and developed approximately 200 technical books over the past five years.

Ted Buchholz, a graduate of the University of South Carolina, is an editor for a publishing company in the Washington D.C. area. He enjoys creative computer play with his children and their friends.

The Tool Kit Series

VIC 20 Edition

by
Ted Buchholz and Dave Dusthimer

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1984

International Standard Book Number: 0-672-22309-0
Library of Congress Catalog Card Number: 83-051670

Edited by: *Patricia Perry*
Illustrated by: *Jill E. Martin and David Cripe*

Printed in the United States of America.

Preface

Thousands of home computers are collecting dust, because their owners don't have the tools to make them work properly. All home appliances require tools to keep them humming along and your VIC 20 computer is no exception. In this book you will find all the tools you need to make your VIC perform the way it should.

This book grew out of a need that we noted after buying our VIC 20. Most manuals and books approach BASIC programming by teaching statements, commands, and program data structures. This approach can be successful, but the average person is not motivated to learn program construction this way. After spending many frustrating evenings with Users' Manuals and BASIC programming books, we still could not write the simple programs we had in mind when we bought the computers. But finally we discovered a way of learning to program that really works. By looking at programs in terms of their working parts—their subroutines—we were able to understand how to write simple programs and soon were designing our own games and quizzes. With the help of this book you can too.

The "tools" in this Tool Kit book are brief 5- to 15-line subroutines. Color, sound and music, graphics, animation, and computational subroutines are the "tools," which, when combined, form the basis for a variety of educational programs and computer games. Each line of each subroutine is carefully described and explained. We will tell you what each subroutine will do, how it can be changed, what the variables control, and hints for further experimentation. This approach makes programming easier, less frustrating, and a lot more fun. The modular form of programming is a sound method of structured programming that works well for the average microcomputer owner.

This book will also help parents teach their children to program. Both of us have small children and the techniques used in this book have helped our children become computer literate. Building programs out of subroutines will give you, the user, quicker and more satisfying results than traditional approaches. Since the subroutines are mostly educational and recreational they will appeal to both parents and children.

In the TOOL KIT books we share a valuable approach to building simple programs with the beginning computer user. Now, let's get to it and have some fun.

Ted Buchholz
and Dave Dusthimer

Note

Throughout this book, reversed letters (e.g., RETURN) designate keystrokes. That is, when this designation is used, it means the reversed letters are a single keystroke on the keyboard (or keys that are pressed simultaneously to produce a desired result) instead of being input as individual characters. This same designation is also used for individual letters which are pressed to initiate a desired reaction in the program.

Dedication

Because of you, I am inspired.
Thank you, Julia, for your love and support.
T.B.

To my parents.
Thank you for your loving and careful direction through the years.
D.D.

Contents

The VIC 20 and How It Works 1

Computers are the result of genius-level engineering. Fortunately for us "normal" people, you need not be a genius to use the VIC 20. Since the VIC can't think like humans, you must learn to think as it does. It isn't hard once you understand how the VIC works.

WHAT MAKES THE VIC 20 WORK?

At the core of every computer is its CPU, or Central Processing Unit, which controls, interprets, and executes the computer functions. The VIC's CPU is a 6502 Microprocessor. This microprocessor "chip" is very tiny, but stores a great deal of information and is very powerful. Computers can only receive information through their CPU in a binary system. That is, computers only know two things—whether data is "on" or "off," "0" or "1," and "yes" or "no." Each of the digits readable by the computer (0 and 1) is known as a *bit*. The VIC 20 is an 8-bit machine, partially because each of its characters is composed of 8 bits.

Look at the illustration of the bit pattern for the "A" character shown in Table 1-1.

Table 1-1. Bit Pattern for the Letter 'A'

BITS	IMAGE	Binary code or BIT PATTERN
1	**	00011000
2	* *	00100100
3	* *	01000010
4	******	01111110
5	* *	01000010
6	* *	01000010
7	* *	01000010
8		00000000

We usually consider bits in groups put together to form a character or group known as a *byte*. Knowing the number of bytes available is another way of knowing how much memory or space is available for use.

How Much Memory Does the VIC Have?

Well, there are two types of memory in the VIC 20—Read-only memory (ROM) and Random-access memory (RAM). The ROM is pro-

grammed into the microprocessor hardware by the manufacturer. The CPU controls computer operations by addressing programs in ROM (Read-only memory). The ROM is permanent in the computer. The RAM (Random-access memory) is temporary storage memory that serves as space for the user's programs. The unexpanded VIC (the standard VIC without additional memory) has 3.58 kilobytes or memory storage available. When you turn the VIC on, it proudly displays:

```
* * * * CBM BASIC V2 * * * *
3583 BYTES FREE
```

There is a way to find out how much memory is left after you start programming, and we discuss it in the next chapter. You can expand the VIC's RAM up to 32,000 individual memory locations (32,000 bytes or 32K kilobytes). The longer your programs, the more memory or RAM you need. Unless you save the data in RAM on tape or diskette, it disappears when the computer is turned off.

THE BINARY SYSTEM

If the computer can only understand "0" and "1" in the binary system, how can you communicate with it? Fortunately, there is an operating system that interprets this binary machine code into something more easily understood by us humans. The operating system for the VIC 20 is a BASIC Interpreter, which allows us to communicate using BASIC as a programming language. We input data principally by using the keyboard, but we can also use joysticks and similar devices. Data is input using BASIC so that it can be understood by the machine.

How Do You Write Programs for the Computer in BASIC?

Programs are written with commands and statements, using a structure which is governed by rules not so different from grammar and syntax rules in English. We're lucky, because BASIC is easier to learn than foreign languages. We introduce you to the major commands and statements in the next chapter.

WHAT ACCESSORIES ARE NEEDED?

You can do a lot with just the VIC 20 and a screen (television or monitor). When you write programs of some length, you won't want to

re-enter the program every time you use it. You can use the Commodore cassette unit to save programs and files on tape, or you can purchase the Commodore Disk Drive to save programs and files on floppy disks. The disk drive turns your VIC into a much more powerful machine because the drive can manage a lot of information without tapping the precious amount of RAM in the VIC 20. Additional RAM is available in memory expansion cartridges to build the RAM up to a possible 32K. There is also a Super Expander cartridge which adds 3K of RAM and some advanced BASIC commands to provide enhanced color, sound, and graphics capabilities.

You can print your programs or results using a printer to produce this "hard copy." There are many other accessories available at computer stores and through mail order houses to tailor the VIC 20 to your personal needs.

THE VIC 20 KEYBOARD

The best way to learn how to use the VIC is to familiarize yourself with the chief input device, the keyboard shown in Fig. 1-1. From the

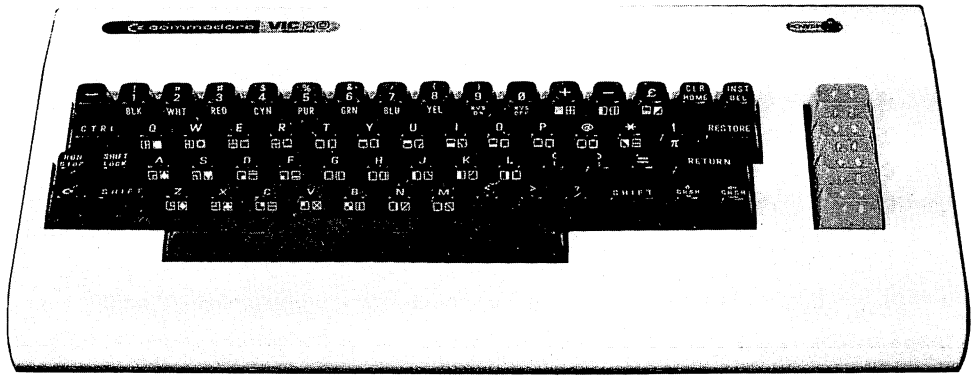


Fig. 1-1. The VIC 20 Keyboard.

guide which came with the computer, you learned how to input characters and the other functioning keys. Let's review this briefly so that you can refer to this section as a reference for the rest of the book. You cannot hurt the computer or destroy the ROM by playing around on the keyboard.

Using the VIC Graphics Characters

All the letter keys and some of the others have graphics characters on the front. In order to type the left character, press the key and the C= key at the same time. For the right character use the SHIFT key. (See Fig. 1-2.)

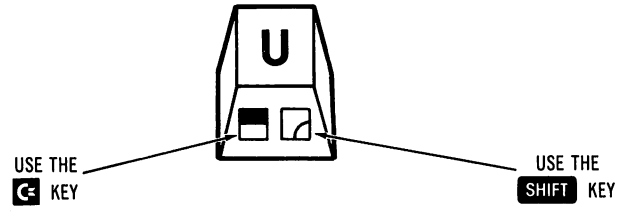


Fig. 1-2. Using graphics characters.

Using the Vic Color Keys and Reverse

Use the CTRL key to change the color of the cursor and what you are inputting. The control key is also used to activate the RVS/ON and RVS/OFF keys. Using the reverse mode prints the character in the screen color and the background in the cursor color. Using reverse to change the colors makes for colorful play. Use the space bar for solid color lines while the reverse mode is on.

EDITING ON THE VIC

The VIC is much easier to use than a regular typewriter because of the features built into the keyboard. The cursor keys are used to edit any line or character on the screen, listed with or without a line number.

PRESS CRSR ↓	–the cursor moves down
PRESS SHIFT or C= + CRSR ↓	–the cursor moves up
PRESS CRSR →	–the cursor moves to the right
PRESS SHIFT or C= + CRSR →	–the cursor moves to the left

All of these cursor movements repeat when the keys are pressed down continuously. After you edit with the cursor, press RETURN to make the change.

The easiest way to edit is with the INST/DEL key. Pressing this key (without the shift) simply deletes the character to the left of the cursor.

Sometimes we think we use this "delete" key more than any other! Pressing this key with the shift key allows you to insert additional characters or spaces.

The CLR/HOME key is very useful for editing. When you want to erase everything on the screen, hold SHIFT (or C=) and this CLR/HOME key. That will rid the screen of all characters and place the cursor in the top left hand corner of the screen (home). If you simply want the cursor "home," use CLR/HOME unshifted.

When you need to stop a program, press the RUN/STOP key. The computer will then say something like:

BREAK IN LINE 190

telling you where the break in the program occurred when RUN/STOP was pressed. (Pressing SHIFT and RUN/STOP is another way to express a load command for a cassette player.) When you press RUN/STOP and RESTORE at the same time, the computer resets to its original colors and the word "READY" appears. Any programs you entered are still in memory, even though the computer looks like you had just turned it on.

You erase all programs in memory and start fresh when you type in "NEW" and press RETURN. This doesn't clear the screen, but it does forget everything.

USING THE VIC DIRECT MODE

The VIC is much more than a calculator, but you can use it just like one in order to solve any problems you wish. You don't need to number the lines like in a program. Try it.

```
PRINT 172 * 4/5          -and press RETURN
PRINT 60 * (3-1)         -and press RETURN
```

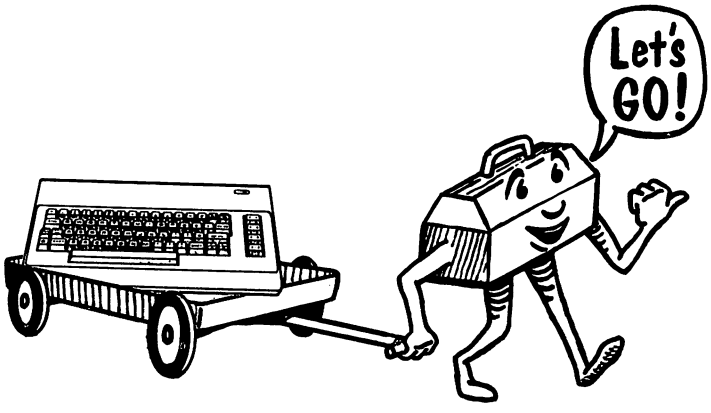
You can even get a short program (without line numbers) to run in the middle of a regular program that has line numbers. As long as you don't exceed 88 characters with the program (4 screen lines—22 characters in a line), then the short program will be executed as soon as you press RETURN. This is helpful when you want to try out a short color or sound routine without disturbing the memory of the program you are working on. Example.

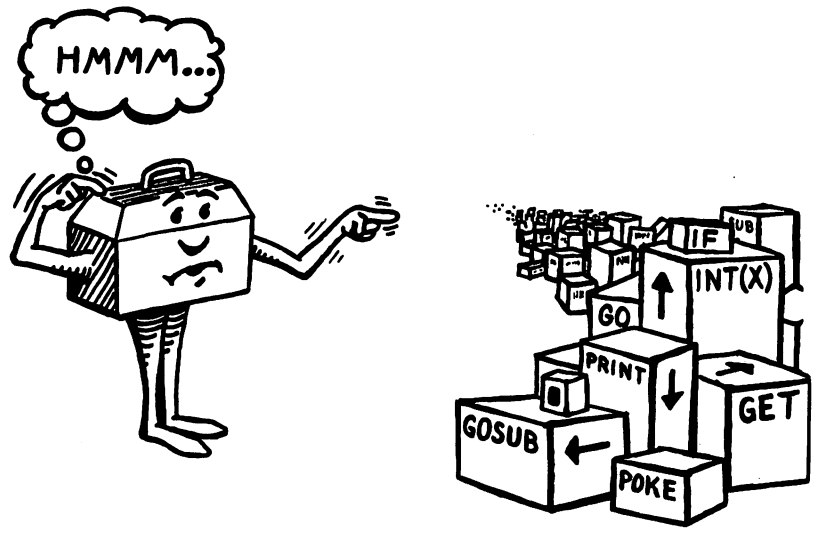
```
10 PRINT "LOUISE AND
```

```
    CHRIS"
POKE 36879,124          -NOTE: There is no line number.
20 GOTO 10
```

When you type in the POKE command, the screen and border turns to yellow and purple. When you key in the second line in the program—GOTO 10—and then type RUN, the computer prints lots of LOUISE AND CHRIS lines.

We hope knowing something about how the VIC works as well as using the keyboard is good preparation for our next chapter, *The Building Blocks of VIC BASIC.*





The Building Blocks of VIC BASIC 2

Once you know how the VIC 20 works and how to speak its language, learning programming is easy. In Chapter One we explained what the VIC can do, and now we are going to teach you to speak VIC BASIC. Remember that the VIC is a "dumb" machine. It can't do anything unless you, the user, command it to perform. Before you get carried away with feelings of great power, you must also remember that the computer can't speak English, so you must give your commands in a language that the computer understands.

VIC BASIC

Your VIC speaks BASIC. Although you can equip your computer to speak other languages, this book deals only with standard VIC BASIC.

Your machine has a very limited vocabulary. The VIC only understands about 70 commands, statements, and functions. In the next few pages we will explain each command and statement and give you an idea of how it works. You will refer to this part of the book often, so you may want to put some notes in the margin. Let's learn how to talk to the computer.

COMMANDS AND STATEMENTS

These are fairly standard BASIC commands and statements. They can be used with most computers that understand BASIC. For convenience, we have alphabetized this listing.

CLR—This erases any variables stored in memory and sets them to zero, but does not erase the program itself.

CLOSE—This statement simply closes a file that you opened earlier with an OPEN statement. You can close specific files by using a file number following the word CLOSE.

CMD—Used to display output, which usually appears on the screen, to a file or a device such as a printer. This device has to be OPENed and CLOSEd just like a file.

CONT—Continue. If you stop a program with a STOP or END statement or the stop key, use CONT to restart the program at the point where you stopped it.

DATA—This command allows you to store information in either string or number form. You must tell the computer to read the DATA with a READ command. Data can be stored in a program by simply typing **DATA, INFO,INFO,INFO**, and so on.

DEF FN—This statement allows you the opportunity to *define* your own functions for the needs of a specific program. You must enclose the parameters in parentheses right after the DEF EN statement.

DIM—DIM lets you set the maximum size of a given array.

END—You guessed it. This statement stops or ends your program. You will most often use this statement to divide subroutines at the end of a program.

FOR TO STEP—Loops are easily created with this statement. The NEXT statement must always be used with FOR TO STEP or you will receive an error message. FOR and TO are used to state a variable.
Example:

```
5 FOR A = 1 TO 5
```

STEP can be used if you want the computer to "step" or cycle in increments greater than one. If you want the computer to move in steps of one, then you can leave the step portion of the command out. By using **STEP** and a negative number you can create a negative step. You must include the NEXT statement so that the computer will go back and act on the next variable in the FOR TO STEP command. For example:

```
60 FOR Y = 200 TO 128 STEP -1
70 POKE 36876,Y
80 NEXT Y
```

GET—Similar to the INPUT statement because data is taken from the keyboard. GET takes the key pressed as the value of the variable following the GET statement. We'll see this used later in the book.

GOSUB—This is one of a couple of branching statements. You must use a RETURN statement to tell the computer to return to the main body of the program. The RETURN statement will

send the computer to the line right after the GOSUB statement. Example:

```
 40 GOSUB 300
300 POKE 36879,124
310 FOR T =!TO 1000: NEXT
320 POKE 36879,11
330 RETURN
```

GOTO—This is another branching statement. Following the GOTO statement and a space, you must input the line number that you want the computer to read next. Once you use a GOTO statement, a continuous loop is made. To break the loop another GOTO statement can be used to send the computer to another line in the program.

IF THEN—This is another very useful statement. The condition described after IF must be true in order for the VIC to perform the statement following THEN. For example, let's consider this program line

```
30 IF X > 50, THEN 200
```

If X is greater than 50, then the VIC will GOTO program line 200. If X is less than or equal to 50, then the next line of the program is executed.

INPUT—This statement halts the program and allows you to input a string or character using the keyboard. This statement may be followed by a prompt in quotation marks. Following the last quotation mark you must type in a semicolon and the name of the variable. Example:

```
100 INPUT "PICK ANY NUMBER"; X
300 INPUT X
```

LET—LET allows you to define variables in a program. You don't have to use the LET statement with the VIC. It interprets **LET A = 1** and **A = 1** the same way.

LIST—This command tells the computer to list the program statements in order. It can only be used in the immediate mode; it cannot be used in a program. You may also list specific lines of the program by entering the line number after the command. When you type a minus sign in front of the number,

the computer will list all lines up to and including that number. Let's look at how the LIST function works.

LIST	-This lists the entire program.
LIST 400	-Only line 400 will appear.
LIST –400	-All lines up to and including 400 will be listed.
LIST 400–	-All lines including and after 400 will be listed.
LIST 200-400	-Lines 200 through 400 will be listed.

LOAD—This command tells the VIC to load a stored program from tape or disk. You can use the program name (in quotes) or a string variable to identify the program you want to load.

NEW—This command clears any program in memory. It acts just like an eraser. Be sure to save and store any programs you want to keep before entering NEW.

NEXT—This statement is used with a FOR TO statement. The NEXT statement tells the computer to evaluate the next value in the FOR TO statement. As long as the value does not exceed the limits of the FOR TO statement, the computer will act on the value.

The NEXT statement controls the loop. If the values are within limits, the loop will continue. Once the values exceed the limits set in the FOR TO statement, the NEXT statement will tell the computer to leave the loop and continue with the next line of the program.

ON GOSUB—This is still another branching statement. If a certain condition exists, the computer will go to the subroutine listed after GOSUB.

```
 5 INPUT X
10 ON X GOSUB 35,100,200
```

When value X is input, the computer will GOSUB lines 35, 100, and 200. You must use a RETURN statement just as you do with GOSUB.

ON GOTO—This statement works just like the ON GOSUB statement, except that you don't have to use RETURN.

OPEN—OPEN tells a BASIC program to use files stored on memory storage devices. OPEN provides a necessary link between the storage device and a file number in the program.

POKE—This command allows you to input a value into the VIC's memory. POKEing is most often used for color, sound, and graphics. Example:

```
10 POKE 36878,15
```
-POKEs a volume (loudness) variable into memory location 36878.

```
20 POKE 36874,200
```
-POKEs a sound variable into one of the VIC's voices, memory location 36874.

PRINT—When you want to display information on the screen the PRINT statement will do it. Words to be printed must appear in quotation marks. When words, numbers, or strings with the variable's name follow the PRINT statement, you don't need to use quotation marks. Example:

```
10 PRINT "HALLOWEEN"
20 PRINT X$
```

READ—This statement tells the computer to read information from DATA statements in order from left to right. You must use READ and DATA statements together.

REM—Remark. The computer does nothing with this statement. It is a notekeeping device for you. REM statements are given line numbers and identify parts of the program for user reference. For example:

```
400 REM *** SCORING SUBROUTINE
```

RESTORE—This statement tells the computer to go back to the beginning of a DATA statement and read the files again.

RETURN—This statement tells the computer to go back and read the line immediately following a GOSUB command. Each subroutine must end with a RETURN statement.

RUN—This is the computer's ignition key. When you command the computer to RUN, it will begin at the first line of the program and carry out each program line in sequence, unless you designate a line number for the program to start. For example:

RUN 190

SAVE—The SAVE command allows you to take a program from the computer's memory and store it on either a cassette tape or a floppy disk. If you are using a cassette recorder for memory storage, simply enter **SAVE** and you will get a series of prompts which will lead you through the saving process. You can put a program name in quotes, or use a string variable name to make location and retrieval easier. You SAVE by simply using:

SAVE "ARCADE SMASH" or

SAVE X$

STOP—This statement stops the program and tells you in what line number the break has been made. Use CONT to restart the program.

VERIFY—This checks the program you have just SAVEd against the one in the computer's memory.

FUNCTIONS

ABS(X)—This function tells the computer to give you the absolute value of an expression. An expression is sometimes referred to as the argument. The **ABS** command will always give you a positive number, even if the argument is negative. This command can be used to find out the absolute value of any complicated series of mathematical computations. For example, input:

ABS(47)	-The absolute value is 47, but we know that. Now try:
ABS 14*(21*6.1) –20	-Now do you see why this command is useful?

ASC(X$)—Each character on the VIC has an ASCII code number. The ASC function will give you the code number for a string variable or a group of numbers that appear in parentheses. Here's how it works.

```
 5 A$ = "D"
10 PRINT ASC(A$)
```

The number 68 will appear on the screen which is the ASCII code for the letter D.

ATN(X)—This function is mathematical and we won't use it in this book. It gives you the arctangent of any number that appears in parentheses after the ATN command.

**CHR$(X)**—This function is the exact opposite of the ASC function. The CHR$ function will change an ASCII code number into the appropriate character. For example, enter:

```
 5 A$ = CHR$(68)
10 PRINT A$
```

This will print the letter D.

COS(X)—This function will compute the cosine of any number that you put into parentheses following the function. For example enter:

```
COS(10)
```

EXP(X)—This returns the exponential function or the opposite of the LOG function. This function raises the number 2.718281828 to the power that you input in parentheses.

```
PRINT EXP(10)
```

This will raise 2.718281828 to the tenth power.

FRE(X)—Tells you the number of unused (free) bytes available in memory.

INT(X)—This function is used when you want a whole number as an output rather than a fraction or decimal. The INT function returns the largest number possible that does not exceed the number in parentheses (ARGUMENT). For negative numbers, the next integer value will be returned. Try this:

```
 5 A = INT(99.887)
10 PRINT A
```

The VIC will print 99. Try the same program only let A = INT(−99.887).

LEFT$(X$,X)—This function will give you the leftmost X characters in X$.

LEN(X$)—This function will return the number of characters, including spaces in a string. Try this

```
 5 A$ = "THE VIC IS A NEAT MACHINE"
20 PRINT LEN(A$)
```

The VIC will print 25. Note that all the spaces count as characters.

LOG (X)—This function will give you the natural logarithms of a number. To figure the log of a number simply input:

PRINT LOG (any number)

MID$ ($,S,X)—This function will give you a part of a string. You can control the part of the string returned.

```
 5 A$ = "MISSISSIPPI"
10 PRINT MID$ (A$,8,4)
```

The 8 tells the VIC to count 8 places. The 4 tells the VIC to print 4 places from the beginning point.

PEEK(X)—You can PEEK into a memory location in order to learn its contents.

POS(X)—This function will compare two strings and tell you at what point a letter in one string begins occurring in another.

**RIGHT$ (X$,X)**—This function will give you the rightmost X characters in X$.

RND(X)—The random function tells the VIC to generate a number between 0 and 1.

SGN(X)—This function will return the algebraic sign of the argument. This function will tell you if the number is positive, negative, or zero. This function is useful in programs that require some complicated mathematics.

SIN(X)—The SIN represents the trigonometric sine function of X.

SPC(X)—Used only with PRINT, it moves forward by X spaces of output.

SQR(X)—This handy function will give you the square root of

any number you choose. Use the following program to extract the square root:

PRINT SQR (any number)

STR$ (X)—STR$ is the opposite of the VAL function. STR$ changes a specified number (the argument) into a string.

TAB (X)—This function works just like the TAB key on a typewriter. With the PRINT statement, the TAB function tells the computer to begin printing a given number of places from the left of the page. It looks like this:

PRINT TAB(20) "YOUR NAME"

The argument can be any number from 1 to 255.

TAN(X)—The TAN function will give you the tangent of the argument.

VAL(X$)—This function will give you the numeric value for a string. It ignores letters in a string and assigns numeric values of number characters only.

INPUTTING PROCEDURES AND HINTS

If you have read your User's Manual, you already know that entering data is just like typing except that it is easier to make corrections on the VIC 20. (You should read your User's Manual if you haven't already.) Here are some simple inputting rules and suggestions.

1. Always number your program lines. When you number your own program lines, we suggest numbering in increments of 10. This gives you space to alter or change a program.
2. Get in the habit of using REM or remark statements as notes to yourself.
3. Be aware of punctuation marks. If you fail to use them properly, the programs won't run.
4. Use the LIST command. As your programs get longer you will need to know how to list certain segments of the program. An example of how to list portions of a program appears earlier in this chapter under the LIST command description.

VARIABLES

A variable is basically something that we assign a value to. We assign values to variables in most of the programs in this book, and you need to understand the concept.

If you say that X = 1, you are defining a variable, X. If we put X = 1 in a program, the computer will interpret X as 1 every time it reads X. Variables are used to perform mathematical operations. Our X = 1 is a numeric variable. We could use this variable in the following way

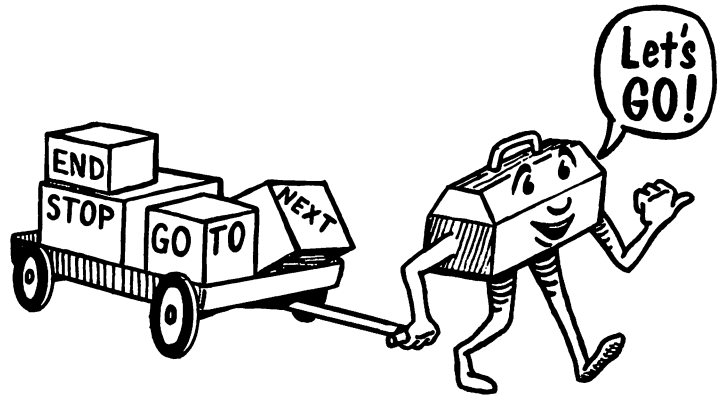
```
10 LET X = 1
20 LET Y = 10 * X
30 PRINT X
40 PRINT Y
```

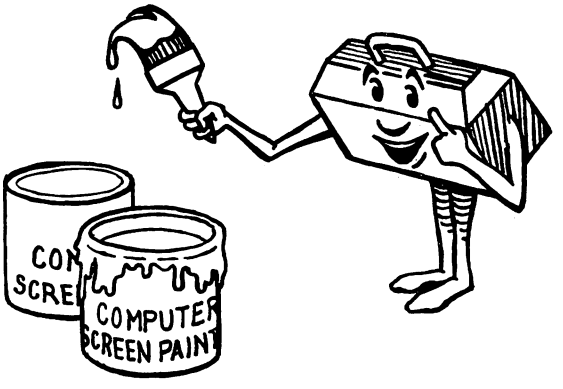
We have stated two variables, X and Y, in terms of each other. We can also state a variable in terms of a string. This type of variable is called, oddly enough, a string variable. If we want to represent a word with the letter X we tell the computer to expect a string by adding a $. Let's say we are designing a "States and Capitals" game, and we want a message to indicate an incorrect answer. We can say

```
10 LET X$ = "WRONG"
20 PRINT X$
```

We are telling the computer that X$ means the same thing as the letters WRONG.

Now that we understand how the computer works and we have an introduction to BASIC language commands, statements, and functions, let's begin doing some real programming.





Color Subroutines: The Color Tool

3

How lucky you are to use the colorful VIC 20. With color alone, you can dazzle yourself and friends using the "show" programs you will learn in this chapter.

KEYBOARD COLORING

Using color on the VIC is easy. If your TV set or color monitor is adjusted properly, the flashing cursor that appears on the screen is blue. Anything that you type will be blue. Try it.

There are eight colors you can choose from the keyboard. Let's change the cursor color from blue to red. Key in CTRL and RED at the same time. The cursor and anything you type will be red. Now change the cursor to green.

At the top of the VIC keyboard are eight color keys, as seen in Fig. 3-1. Try them all out to see how they look.

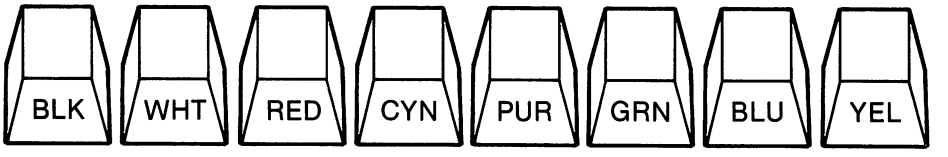


Fig. 3-1. The VIC color keys.

(NOTE: Of course WHT on a white background looks invisible. Hiding a character by changing its color to the same color as the background can be useful to us when we develop programs later in the book.)

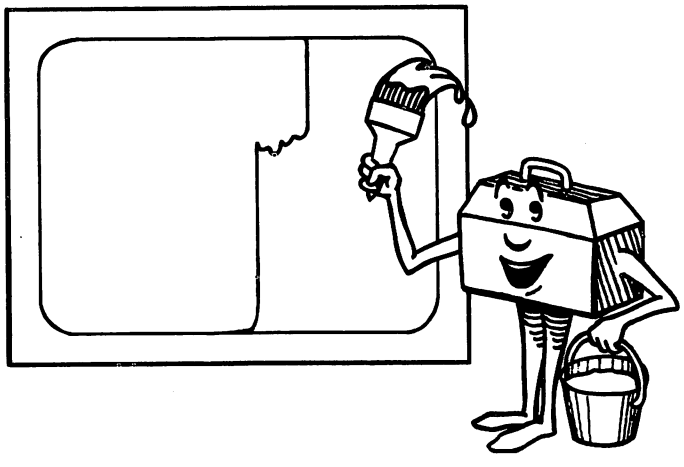
Let's turn the cursor into a paint brush and have some fun. The easiest way to make a brush is to use the Reverse On key. Press CTRL and RVS/ON simultaneously. This makes any key "reversed"—that is, the letter or character will be the same color as the background, while the character background is the same color as the cursor. For example:

Normal mode	K	Blue letter
Reverse mode on	K	Blue background

When you press the space bar with reverse on, a solid block appears, and you can make a solid line with the space bar blocks. Try this:

Key CTRL and RVS/ON.
Key CTRL and RED.

Now when you press the space bar, a solid red line appears. Make lines using all eight colors. Can you make a design on the screen using different colors with the blocks and lines? To turn the reverse mode off, key in CTRL and RVS/OFF.



SCREENS AND BORDERS

When you turn the VIC on, you see a cyan border and white screen. This is a pleasing combination, but you can easily change the border and screen to any one of 255 different combinations.

How do you do this? By addressing or POKEing a specific memory location in the computer. It's easy to do. Type in:

```
POKE 36879,24
```

Now you have a black border and white screen. 36879 is the location in memory where screen/border combinations are stored. 24 is the black and white color combination.

Try this simple program to see the range of combinations the VIC can make.

```
10 FOR C = 1 TO 255
```
-Sets the C variable (color combination) from values 1 to 255.

```
20 POKE 36879,C
```
-We poke C into memory location 36879.

```
30 FOR T = 1 TO 100: NEXT T     -The VIC timer counts from 1 to
                                 100 to delay the changes.
40 NEXT C                       -Tells the computer to display the
                                 next POKEd value.
```

For reference, Table 3-1 shows you the values you can POKE into 36879 for the following screen and border combinations.

Table 3-1. Screen and Border Combinations

Screen	Border							
	BLK	**WHT**	**RED**	**CYAN**	**PUR**	**GRN**	**BLU**	**YEL**
BLACK	8	9	10	11	12	13	14	15
WHITE	24	25	26	27	28	29	30	31
RED	40	41	42	43	44	45	46	47
CYAN	56	57	58	59	60	61	62	63
PURPLE	72	73	74	75	76	77	78	79
GREEN	88	89	90	91	92	93	94	95
BLUE	104	105	106	107	108	109	110	111
YELLOW	120	121	122	123	124	125	126	127
ORANGE	136	137	138	139	140	141	142	143
LT. ORANGE	152	153	154	155	156	157	158	159
PINK	168	169	170	171	172	173	174	175
LT. CYAN	184	185	186	187	188	189	190	191
LT. PURPLE	200	201	202	203	204	205	206	207
LT. GREEN	216	217	218	219	220	221	222	223
LT. BLUE	232	233	234	235	236	237	238	239
LT. YELLOW	248	249	250	251	252	253	254	255

Let's write a colorful "reward" program. We use a program like this later in the book to reward you for being successful in a game program.

```
5 PRINT "■"                     -Clears the screen. Inside the
                                 quotes you key in SHIFT and
                                 CLR/HOME.
10 FOR A = 24 TO 31             -Sets the values of A from 24 to
                                 31.
```

```
20 POKE 36879,A
```
-POKEs A values into the screen and border memory location. If you look at the A values, they correspond to different border combinations for a white screen.

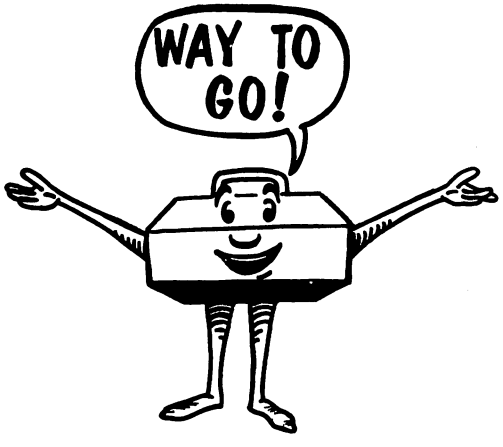
```
30 FOR T = 1 TO 200: NEXT
   T
```
-Tells the VIC timer to count from 1 to 200 before changing to the next A value.

```
40 NEXT A
```
-Changes the A value.

```
50 FOR B = 120 TO 127
```
-Lines 50–80 are similar to lines 10–40 except they define POKE values for different border colors with a yellow screen.

```
60 POKE 36879,B
70 FOR T = TO 200: NEXT T
80 NEXT B
90 GO TO 5
```
-Tells the computer to return to line 5 and thus begin the program again.

(NOTE: In order to stop this program, you can push the stop key since the program has no end or stop within itself.)



COLOR IN PRINT STATEMENTS

When we use the PRINT statement, the directions and messages are included in between the quotation marks following PRINT. Remember

that when you type: PRINT "GOOD MORNING", the computer responds with—GOOD MORNING (the message inside the quotation marks).

When we give directions for color or other functions in PRINT statements, special symbols appear inside the quotation marks. Try this out:

(simultaneously)

PRINT "CTRL BLK VIC 20"

This actually shows on the computer as:

PRINT " ■ VIC 20"

and prints VIC 20 in black letters once the RETURN key is pressed.

The colors appear as such inside PRINT statements:

CTRL BLK = ■	CTRL PUR = ◨
CTRL WHT = E	CTRL GRN = ↑
CTRL RED = £	CTRL BLU = ←
CTRL CYN = ◥	CTRL YEL = π

Note also that inside the PRINT statement, reverse on and off produces special characters. Let's use our knowledge of color to print a somewhat completely colorful statement.

```
10 PRINT "♥■R-22 spaces-■"
20 PRINT "£R-22 spaces-■"
30 PRINT "◥R-22 spaces-■"
40 PRINT "◨R-22 spaces-■"
50 PRINT "↑R-22 spaces-■"
60 PRINT "£R-8 spaces-HELLO-9 spaces-■"
70 PRINT "←R-22 spaces-■"
80 PRINT "πR-22 spaces-■"
```

Try to figure out what colors we have used. There is a character key in the appendices which may help you. As you use color more frequently in PRINT statements, these symbols will become much more familiar to you. For now, make up your own special messages.

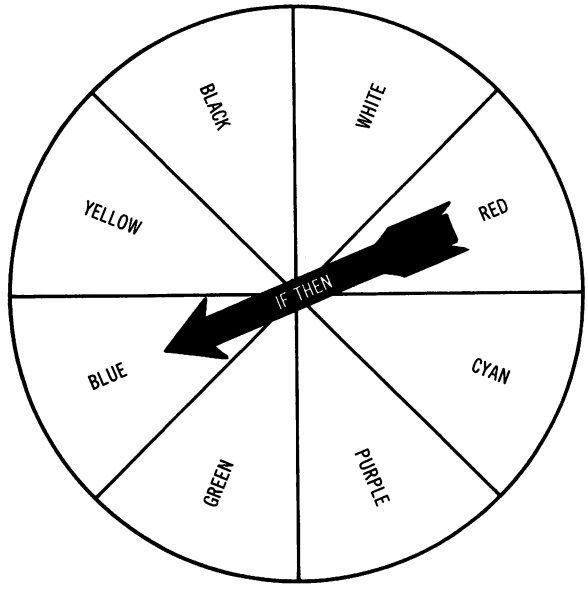


Fig. 3-2. Color choice.

COLOR SUBROUTINES

Now that we have shown how to color the screen and border, as well as how to use color in programs, let's look at some interesting programs and routines which we can use alone or as parts of other programs.

Color Choice

```
10 PRINT"■"
20 PRINT "DO YOU LIKE BL
   ACK, WHITE, RED, CYA
   N, PURPLE, GREEN, OR
   YELLOW THE BEST?"
50 A1$ = "BLACK"

60 A2$ = "WHITE"
70 A3$ = "RED"
```

–Lines 50–120 define as string variables the eight major VIC screen color possibilities.

```
 80 A4$ = "CYAN"
 90 A5$ = "PURPLE"
100 A6$ = "GREEN"
110 A7$ = "BLUE"
120 A8$ = "YELLOW"
130 PRINT "WHAT IS
    YOUR FAVORITE
    COLOR?"
140 INPUT B$
```

–Asks you your favorite color.

–Inputs your favorite color (and is automatically defined as one of the string variables).

```
141 PRINT "♥"
150 IF B$ = A1$ THEN
    GOTO 400
```

–Lines 150–220 set the condition that if your input (B$) matches one of the eight string variables, then the computer "goes to" another line in the program.

```
160 IF B$ = A2$ THEN
    GOTO 500
170 IF B$ = A3$ THEN
    GOTO 600
180 IF B$ = A4$ THEN
    GOTO 700
190 IF B$ = A5$ THEN
    GOTO 800
200 IF B$ = A6$ THEN
    GOTO 900
210 IF B$ = A7$ THEN
    GOTO 1000
220 IF B$ = A8$ THEN
    GOTO 1100
400 FOR A = 8 TO 15
```

–Lines 400–420 and 500–520, etc., define values A through H as combinations to be POKEd into the VIC's memory location. At the end of each of these routines, the program tells the computer to go to line 130 to ask the question again.

```
405 POKE 36879,A
```

```
410 FOR T = 1 TO 100:
    NEXT T
415 NEXT A
420 GOTO 130
500 FOR B = 24 TO 31
505 POKE 36879,B
510 FOR T = 1 TO 100:
    NEXT T
515 NEXT B
520 GOTO 130
600 FOR C = 40 TO 47
605 POKE 36879,C
610 FOR T = 1 TO 100:
    NEXT T
615 NEXT C
620 GOTO 130
700 FOR D = 56 TO 63
705 POKE 36879,D
710 FOR T = 1 TO 100:
    NEXT T
715 NEXT D
720 GOTO 130
800 FOR E = 72 TO 79
805 POKE 36879,E
810 FOR T = 1 TO 100:
    NEXT T
815 NEXT E
820 GOTO 130
900 FOR F = 88 to 95
905 POKE 36879,F
910 FOR T = 1 TO 100:
    NEXT T
915 NEXT F
920 GOTO 130
1000 FOR G = 104 TO 111
1005 POKE 36879,G
1010 FOR T = 1 TO 100:
    NEXT T
1015 NEXT G
1020 GOTO 130
```

Color Choice is a program that uses simple IF...THEN logic. Now let's try another colorful program.

Show of Color

```
10 PRINT "■" SELECT A
   COLOR SHOW IN ONE
   OF THE FOLLOWING
   COLORS:"
20 PRINT
30 PRINT "P . . . PURPLE"
40 PRINT "C . . . CYAN"
50 PRINT "B . . . BLACK"
60 PRINT "G . . . GREEN"
70 GET C$: IF C$ = "" "
   THEN 70

80 IF C$ = "P" THEN
   GOTO 200

90 IF C$ = "C" THEN
   GOTO 300
100 IF C$ = "B" THEN
   GOTO 400
110 IF C$ = "G" THEN
   GOTO 500
120 IF C$<>"P" AND C$<
   >"C"AND C$<>"B"
   AND C$<>"G" THEN
   600
200 FOR A = 72 TO 79

210 POKE 36879,A
220 FOR T = 1 TO 200:NEXT
   T
```

-Lines 10–60 give the program directions.

-Tells the VIC to get a value for the string variable C$. If no value is keyed (" "), then the VIC keeps waiting.

-Lines 80–110 tell the VIC which values to pay attention to and where to go if they are pressed.

-If a key other than P, C, B, or G is pressed, the program branches to Line 600 where instructions are given.

-Lines 200–240, 300–340, 400–440, and 500–540 give us the color shows for each of the four possible selections. These routines are nearly identical to the "Color Choice" program we have just finished.

```
230 NEXT A
240 GOTO 10
300 FOR B = 56 TO 63
310 POKE 36879,B
320 FOR T = 1 TO 200:NEXT
    T
330 NEXT B
340 GOTO 10
400 FOR C = 8 TO 15
410 POKE 36879,C
420 FOR T = 1 TO 200:NEXT
    T
430 NEXT C
440 GOTO 10
500 FOR D = 88 TO 95
510 POKE 36879,D
520 FOR T = 1 TO 200:NEXT
    T
530 NEXT D
540 GOTO 10
600 PRINT "YOU DIDN'T HIT
    ONE OF THE COLOR
    KEYS!!!"
```

-Tells you that you hit another key besides P, C. B, or G. Note the use of not equal to (<>)and the logical operator, AND.

```
605 FOR T = 1 TO 700:NEXT
    T
610 GOTO 10
```

Random Color

The last two programs allowed you to choose colors by means of INPUT, IF...THEN, and GET statements. Now let's make a program that produces screen and border combinations at random.

```
 5 PRINT "□"
10 C = INT (RND(0)*255)
```

-Clears the screen.

-Sets the color value of the variables using the INT (integer) and RND (random number generator) functions.

A) RND(0) produces random numbers between 0 and 1, such as .012 and .469.

B) The *255 multiplies the random number by 255 (the maximum value for color combinations).

C) The INT function rounds off the random number times 255 to the nearest integer (or whole number)—giving the computer a value that can be POKEd into the VIC memory location for screen and border color combinations.

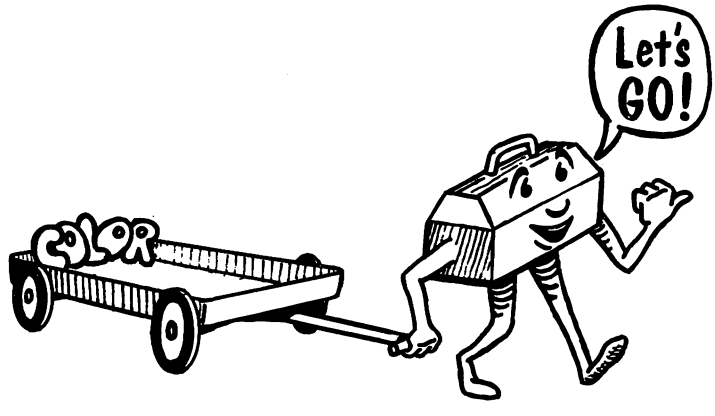
```
20 POKE 36879,C
```
-Pokes the value into memory.

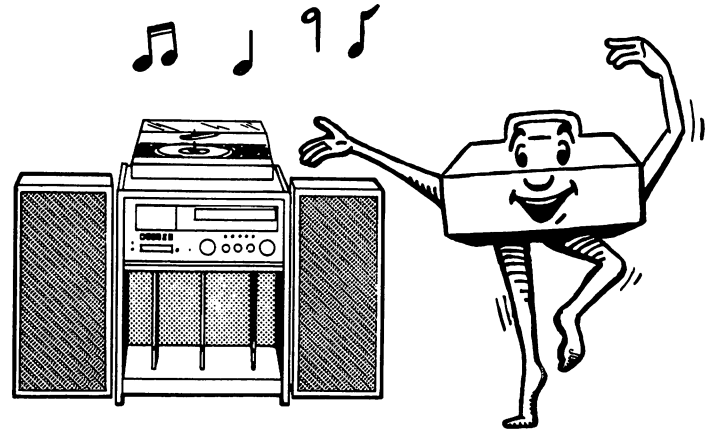
```
30 FOR T = 1 TO 100: NEXT
   T
```
-Tells the VIC timer to count from 1 to 100 before the value changes.

```
40 GOTO 5
```

We hope you have enjoyed learning to use color on the VIC. These three programs are designed to be fun as well as to introduce you to some BASIC programming and logic.

In the next few chapters we cover sound, graphics, and animation. We will use color subroutines heavily in these chapters and throughout the rest of the book. Experiment and enjoy!





Sound and Music Subroutines: The Sound Tool

4

Your VIC 20 has a great voice. It has surprising musical capabilities with a range of five octaves in each of four voices or tones. You can learn to "play" the VIC with a minimum of practice. In this chapter we will show you how to make beautiful music with your VIC and how to use sound subroutines to add some pizzazz to your future programs.

POKEING SOUND

You play the four VIC tones by using the POKE statement to access the sound-related memory locations.

First you must set the volume for sound. One memory location sets the volume for all four tones. Its location is 36878, and there are 15 volume settings available (1-15).

Each of the four tones can play up to 128 different ranges or notes. Let's look at their memory locations (shown in Table 4-1.)

Table 4-1. Sound Memory Locations

Location	Range	Description
36874	128-255	Lowest tone - "ALTO"
36875	128-255	Middle tone - "TENOR"
36876	128-255	Highest tone - "SOPRANO"
36877	128-255	Growler or White noise tone
36878	1-15	Volume

To start making music, input these lines:

```
POKE 36878,15
POKE 36874,225
```

You have just played Middle C on the musical scale in the lowest tone and the highest volume. If you want the tone to stop you must:

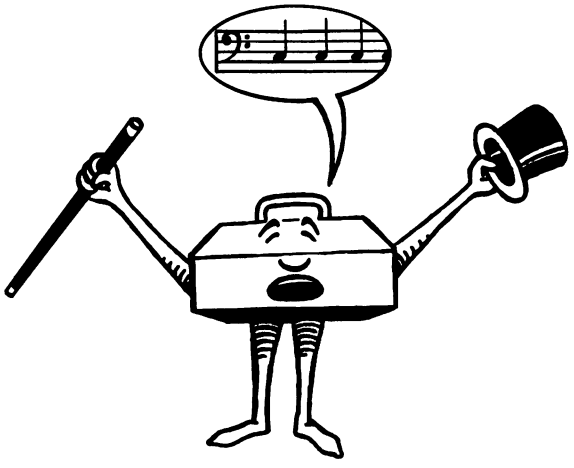
```
POKE 36874,0
```

This sets the value at sound memory location 36874 to 0.

If you want Middle C to play for a specific period of time,you must set your statements in program form. Try this:

```
10 POKE 36878,15
20 POKE 36875,225
30 FOR T = 1 TO 500 : NEXT T
40 POKE 36875,0
```

Middle C is played in the tenor tone for about ½ second. The statement in line 40 sets the value for the tenor tone to 0, ending this sound display.



SOUND VARIABLES

Lets play with this program to learn more about sound.

VOLUME—Change the value in line 10 to anything from 1 to 15.

NOTE—Change the value in line 20 to anything from 128 to 255 (128 is the lowest note, 255 is the highest possible note).

TONE—Change the tone from 36875 to alto (36874) or soprano (36876).

DURATION—Change the range of T in the **FOR. . .NEXT** statement to a larger number for longer duration, smaller for shorter.

THE NOISE MAKER

Memory location 36877 gives us some fun noises which we can use to enhance our programs and games. Input this program:

```
 10 POKE 36878,15
 20 POKE 36877,159
 30 FOR T = 1 TO 1000 : NEXT
 40 POKE 36877,0
 50 POKE 36877,147
 60 FOR T = 1 TO 1000 : NEXT
 70 POKE 36877,0
 80 POKE 36877,135
 90 FOR T = 1 TO 1000 : NEXT
100 POKE 36877,0
```

This sounds like something is about to crash—but doesn't. This program uses a lot of lines for a simple sound effect. Later in this chapter we show how other BASIC statements can be used more efficiently when playing sounds.

SOUND ENCYCLOPEDIA

Let's look at this short program which allows you to select the note, duration, and volume you want to hear.

```
10 PRINT "■"
20 INPUT "NOTE: 128-255"; N
30 INPUT "DURATION"; D
40 INPUT "VOLUME: 1-15"; V
50 POKE 36878,V
60 POKE 36875,N
70 FOR T = 1 TO D: NEXT T
80 POKE 36875,0
90 GOTO 10
```

The INPUT statement in lines 20–40 lets you select your own variables. In order to stop this continuous loop program, you must press RUN/STOP and RESTORE.

We play this program using our tenor (or middle) tone. If you would like to change this variable, add and change these lines:

```
Add:     145 INPUT "TONE: 36874,36875,36876, OR 36877"; Z
Change:  60 POKE Z,N
Change:  80 POKE Z,O
```

Now you control sound more completely! Defining each value as a variable;

N= note
D= duration
V= volume
Z= tone

makes programming and playing with sound much easier.

POKEing More Than One Tone or Note

We can play all three music tones (four if we want to include noise) at the same time by using the POKE statement. We can also play notes within one tone at the same time. This allows us to make musical chords rather than single notes. Try this:

```
10 POKE 36878,5
20 POKE 36875,195
30 POKE 36875,207
40 POKE 36875,209
50 FOR T = 1 TO 500 : NEXT
60 POKE 36875,0
```

You'll hear three notes in the tenor range, played at the same time (actually C, E, and F notes). Now try this:

```
10 POKE 36878,5
20 POKE 36874,195
30 POKE 36875,195
40 POKE 36876,195
50 FOR T = 1 TO 500 : NEXT
60 POKE 36878,0
```

Three tones for one note! This is almost harmonic. Note that we shut the sound off in line 60 POKEing the Volume value to 0.

Music Lesson Subroutines

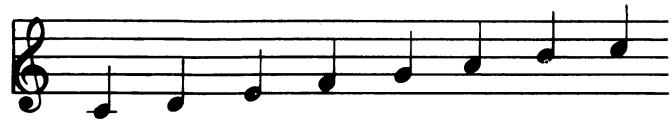


Fig. 4-1. The C scale.

Let's begin our lesson with a scale. This scale is not exactly musically correct, but it is an easy way to approximate a scale for a game or quiz.

```
10 POKE 36878,10
20 FOR X = 128 TO 255 STEP 10
30 POKE 36876,X
40 FOR D = 1 TO 150 : NEXT D
50 NEXT X
60 POKE 36876,0
```

In this program, the VIC plays notes from 128 to 255 in increments (or steps) of 10. Each note is played for a count of 150. Let's play a real scale now. In order to do so, we can help you with Table 4-2, a chart of musical notes which makes the notes and corresponding numbers easy to identify.

C Scale

```
 20  T = 36874
 30  POKE 36878,10
 40  POKE T,135          -C Note
 50  GOSUB 200
 60  POKE T,147          -D Note
 70  GOSUB 200
 80  POKE T,159          -E Note
 90  GOSUB 200
100  POKE, T,163         -F Note
110  GOSUB 200
120  POKE T,175          -G Note
130  GOSUB 200
140  POKE T,183          -A Note
150  GOSUB 200
```

```
160 POKE T,191          -B Note
170 GOSUB 200
180 END
200 FOR D = 1 TO 500 :
    NEXT
210 POKE T,O
220 RETURN
```

Table 4-2. Musical Notes

Note	Low Octave	Middle Octave	High Octave	Top Octave
C	135	195	225	240
C#	143	199	227	241
D	147	201	228	
D#	151	203	229	
E	159	207	231	
F	163	209	232	
F#	167	212	233	
G	175	215	235	
G#	179	217	236	
A	183	219	237	
A#	187	221	238	
B	191	223	239	

This little program will play the C Scale one note at a time. After all seven notes have been played, the program ends. How could you get the scale to play again and again? Change 180 to:

```
180 GOTO 20,
```

and the scale will play continuously.

You can instruct the computer to play any musical scale using this program as an example. Just change the notes in lines 40-160.

The values for G Major Scale are:

G = 175
A = 183
B = 191
C = 195
D = 201

E = 207
F# = 212

Scales can be combined if you like. Simply POKE the scales on the same line with a colon between the statements. Line 40 would be:

```
40 POKE T,135 : POKE T,175
```

PLAYING CHORDS ON THE VIC

As we mentioned earlier in this chapter, you can generate more than one tone at the same time. Here is how the VIC can sound like an organ.

The VIC Plays Chords!

```
 10 PRINT "□"
 20 S = 36875 : V = 36878
400 POKE V,10 : POKE S,225 : POKE S,159 : POKE S,175 :
    GOSUB 2000
500 POKE V,10 : POKE S,201 : POKE S,212 : POKE S,219 :
    GOSUB 2000
600 POKE V,10 : POKE S,207 : POKE S,217 : POKE S,223 :
    GOSUB 2000
700 POKE V,10 : POKE S,209 : POKE S,221 : POKE S,228 :
    GOSUB 2000
800 POKE V,10 : POKE S,215 : POKE S,223 : POKE S,228 :
    GOSUB 2000
900 POKE V,10 : POKE S,219 : POKE S,227 : POKE S,231 :
    GOSUB 2000
1000 POKE V,10 : POKE S,223 : POKE S,229 : POKE S,233 :
    GOSUB 2000
```

Lines 400-1000 represent the natural chords for C, D, E, F, G, A, and B.

```
2000  FOR T = 1 TO 1000 : NEXT
```

Subroutine 2000 makes the computer count before moving to the next chord.

```
2010 POKE S,0
2020 GOTO 20
2030 RETURN
```

How can we add to this program to make it even more interesting? Instead of letting the computer have the fun of playing the chords, let's play this VIC organ ourself.

THE VIC ORGAN

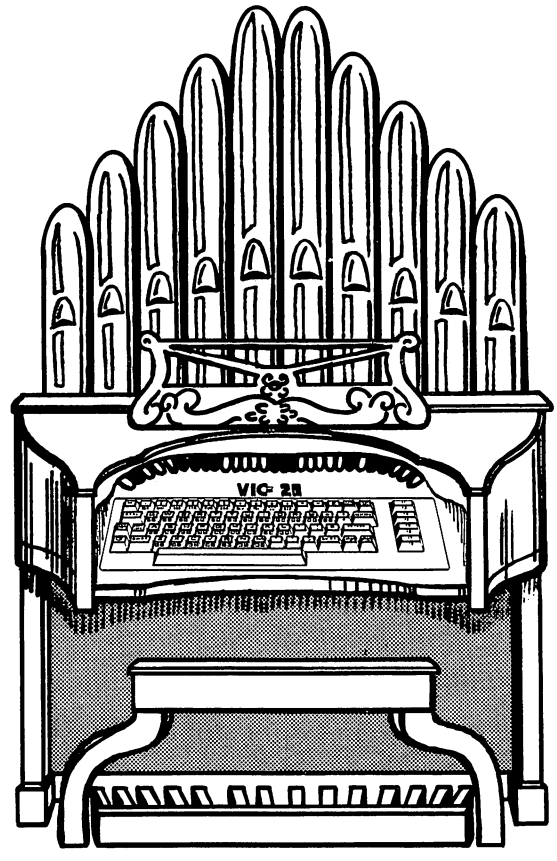


Fig. 4-2. The VIC organ.

Using our chord program let's make some changes so we can input the chord we want to play.

Add these lines:

```
30 INPUT A$
```

```
 40 IF A$ = "C" THEN GOTO 400
 50 IF A$ = "D" THEN GOTO 500
 60 IF A$ = "E" THEN GOTO 600
 70 IF A$ = "F" THEN GOTO 700
 80 IF A$ = "G" THEN GOTO 800
 90 IF A$ = "A" THEN GOTO 900
100 IF A$ = "B" THEN GOTO 1000
```

It is simple to make the VIC "interactive." Lines 40-100 ask if a key is entered in response to the input and make the computer go to the appropriate chord routine.

Hints About Playing and Adapting the VIC Organ

1. You can make the organ play without interruption by inputting your next chord choice as soon as you hear the first. The VIC will accept the input as soon as the preceding chord starts to play. Of course you must press RETURN after each selection.
2. The organ can become a piano by deleting the note values you don't need. Only one note will be played at a time. For example, change line 400:

   ```
   400 POKE V,10 : POKE S,225
   ```

   We prefer the simplicity and clarity of piano music over that of a chord organ. Have some fun with this adaptation.
3. Would you like to end the program with a key? Easy. Simply add these lines

   ```
    110 IF A$ = "Z" THEN GOTO 1500
   1500 END
   ```

   Now press "Z" and the program stops cold. This saves you from using RUN/STOP and RESTORE.

VIC Song Book

You can play any song you want by POKEing the sounds in the proper order. We can see how entering even short songs would become long and dull, but VIC BASIC provides an easier way to do it.

Sound Subroutines Using Data Files

The DATA and READ statements can save a lot of inputting time. Values can be assigned to variables by telling the computer to READ

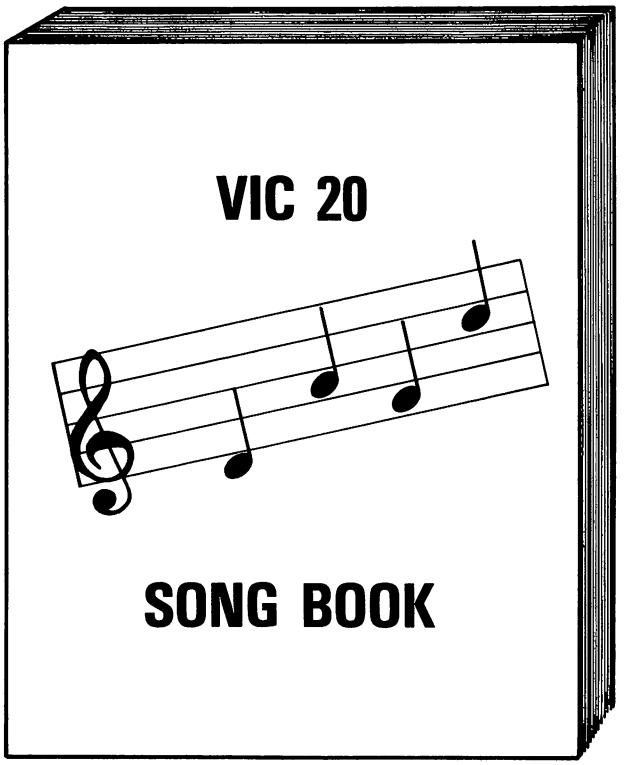


Fig. 4-3. The VIC songbook.

them from a DATA list. The DATA statement does not tell the computer to do anything; it simply provides the values in the list.

When we use the READ command and a Data File, we must keep some things in mind. The computer begins with the first value in a data list and continues until there is no more data or until the computer finds a value that tells it to stop. So be sure you input the correct data and be sure that the notes are in the right order, because if you don't, the VIC will play sour notes.

Let's use the READ Command and DATA FILES to make some beautiful music with the VIC.

Input this program:

```
10 PRINT "□"
20 POKE 36878,15
```

```
 30 S = 36876
 40 READ F
 50 IF F = 999 THEN POKE S,0 : END
 60 POKE S,F
 70 FOR T = 1 TO 500 : NEXT
 80 GOTO 30
 90 DATA 195,195,215,215,219,219,215,209,209,207,207,201,
    201,195,215,215,209,209,207,207
100 DATA 201,215,215,209,209,207,207,201,195,195,215,215,
    219,219,215,209,209,207,207
110 DATA 201,201,195
120 DATA 999
```

When you run this beauty you will hear an electronic rendition of "Twinkle, Twinkle Little Star." Save this program because we show you some graphic subroutines to use with this song later in Chapter 6.

Here are note sequences of some old favorites that you can try out.

MARY HAD A LITTLE LAMB
EDCDEEE DDD EGG EDCDEEEDD EDC

ARE YOU SLEEPING
GABGGABG BCD BCD DEDCB GDEDCBGGDG GDG

You can add many songs to your own library, anything from nursery songs to Carole King. We will use the DATA and READ statements more in later chapters. For now, remember that these are great time savers.

ARCADE SOUNDS

Earlier in this chapter we mentioned the noisemaker or growler for the VIC. All of the programs in this section will be particularly useful for game, arcade game, and educational programs. You can use all four voices of the VIC for these sound effects.

Type this program in:

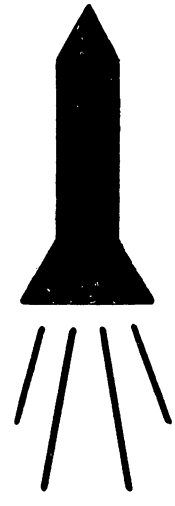
```
10 POKE 36878,15
20 S = 36875
30 POKE S,221
40 FOR T = 1 TO 500 : NEXT
50 POKE S,241
60 FOR T = 1 TO 500 : NEXT
70 GOTO 10
```

This sounds like a European police siren, doesn't it? You can use this in a chase or time-sensitive game. One simply cannot listen to this siren for long without panicking.

We'll use this next routine later to make the sound of a crash.

```
10 POKE 36877,200
20 FOR V = 15 TO 0 STEP -1
30 POKE 36878,V
40 FOR T = 1 TO 50 : NEXT T
50 NEXT V
60 POKE 36877,0
```

You should change the duration in line 40 to experiment. This routine uses the volume as the changing variable while the white noise actually stays constant.



If you develop any type of space game, a rocket blast-off noise can be essential. Try this:

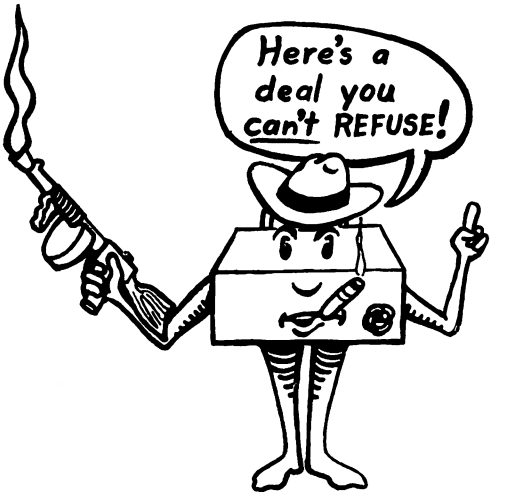
```
10 POKE 36878,10
20 FOR S = 128 TO 255 STEP 5
```

```
30 POKE 36877,S
40 FOR T = 1 TO 100 : NEXT T
50 NEXT S
60 POKE 36878,0
```

Machine Gun

Now try typing this:

```
10 POKE 36878,10
20 POKE 36877,220
30 FOR T = 1 TO 25 : NEXT
40 POKE 36877,0
50 GOTO 10
```



How about some musical reinforcement for a correct answer in a quiz.

```
10 POKE 36878,5
20 S = 36875
30 POKE S,195
40 GOSUB 200
50 POKE S,201
60 GOSUB 200
70 POKE S,207
80 GOSUB 200
90 POKE S,201
```

```
100 GOSUB 200
110 POKE S,207
120 GOSUB 200
130 POKE S,195
140 GOSUB 200
150 END
200 FOR T = 1 TO 300 : NEXT
210 POKE S,0
220 RETURN
```

If you enter an answer incorrectly your VIC might give you this message

```
 10 POKE 36878,10
 20 S = 36874 : N = 36877
 30 FOR D = 200 TO 128 STEP -10
 40 POKE S,D
 50 FOR T = 1 TO 300 : NEXT
 60 NEXT D
 70 POKE S,O
 80 POKE N,128
 90 FOR T = 1 TO 300 : NEXT
100 POKE N,O
```

The more you work with VIC sound and music, the more interesting touches you can add to your programs. Keep experimenting. Try to use DATA and READ statements for these last two programs.

PUTTING COLOR AND SOUND TOGETHER

We discussed color in Chapter 3 and now we have introduced you to sound routines for the VIC. Let's combine what we have learned in one program. Let's change the VIC Organ into the VIC *Color* Organ.

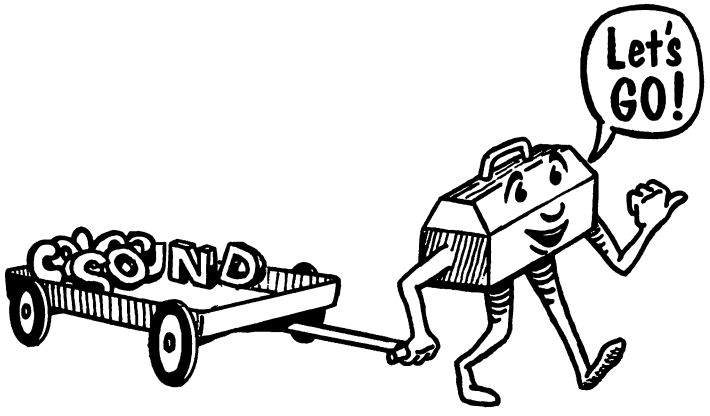
VIC Color Organ

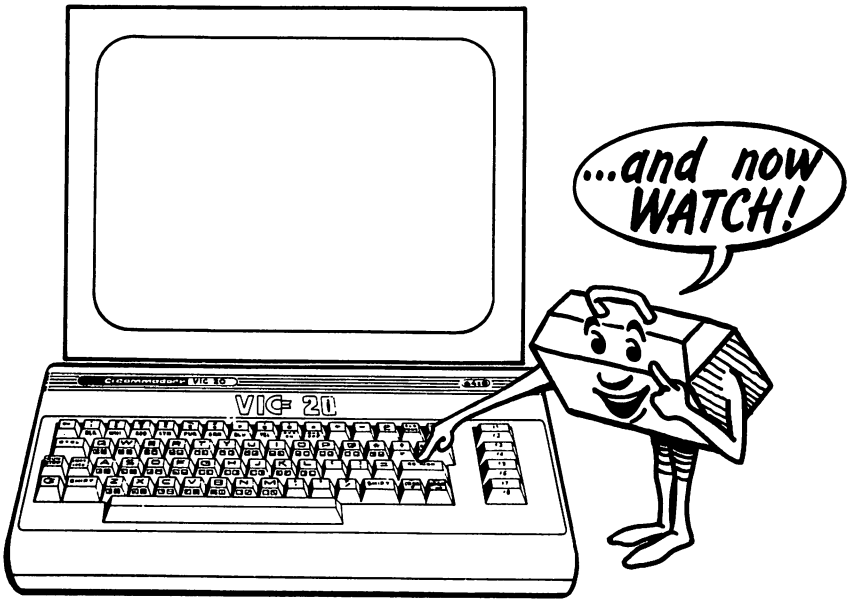
```
 10 PRINT "■"
 20 S = 36875 : V = 36878 : K =36879
 30 INPUT A$
 40 IF A$ = "C" THEN GOTO 400
 50 IF A$ = "D" THEN GOTO 500
```

```
 60 IF A$ = "E" THEN GOTO 600
 70 IF A$ = "F" THEN GOTO 700
 80 IF A$ = "G" THEN GOTO 800
 90 IF A$ = "A" THEN GOTO 900
100 IF A$ = "B" THEN GOTO 1000
400 POKE V,10 : POKE S,225 : POKE S,159 : POKE S,175 :
    POKE K,15 : GOSUB 2000
500 POKE V,10 : POKE S,201 : POKE S,212 : POKE S,219 :
    POKE K,30 : GOSUB 2000
600 POKE V,10 : POKE S,207 : POKE S,217 : POKE S,223 :
    POKE K,45 : GOSUB 2000
700 POKE V,10 : POKE S,209 : POKE S,221 : POKE S,228 :
    POKE K,60 : GOSUB 2000
800 POKE V,10 : POKE S,215 : POKE S,223 : POKE S,228 :
    POKE K,75 : GOSUB 2000
900 POKE V,10 : POKE S,219 : POKE S,227 : POKE S,231 :
    POKE K,90 : GOSUB 2000
1000 POKE V,10 : POKE S,223 : POKE S,229 : POKE S,233 :
    POKE K,124 : GOSUB 2000
2000 FOR T = 1 TO 1000 : NEXT
2010 POKE S,0
2020 GOTO 20
2030 RETURN
```

With the additional POKE statements in lines 400-1000, we have added a splash of color to our VIC Organ. Each time a different chord is played, the screen and border flash with different colors.

In this chapter we have covered a lot of ways to use sound and music on the VIC. We encourage you to experiment and invent your own subroutines to use.


Let's
GO!
SOUND



Graphic Subroutines: The Graphic Tool

5

Probably one of the main reasons you bought your VIC 20 was to create computer graphics, right? Well, in this chapter we show you how to create mazes, borders, graphs, playing boards for games, and a variety of fun and useful characters.

CREATING GRAPHICS WITH THE PRINT COMMAND

Graphics can be created using the PRINT command. Simply put the graphics characters inside quotation marks following PRINT. Try this program:

```
10 PRINT "■■■■"          (Press C and I four times)
20 PRINT "■  ■"          (Press C and J, SPACE, SPACE,
                          and Press C and N)
30 PRINT"■●●■"           ( C -J, SHIFT -Q, SHIFT -Q,
                          and C -N)
40 PRINT"■  ■"           (Same as 20)
50 PRINT"\‿/"            (SHIFT -M, SHIFT -J, SHIFT
                          -K, SHIFT -N
60 PRINT"/  \"           (SHIFT -N, SPACE, SPACE,
                          SHIFT -M)
```

This program will display a friendly face when you type RUN and press RETURN. The completed face is shown in Fig. 5-1.

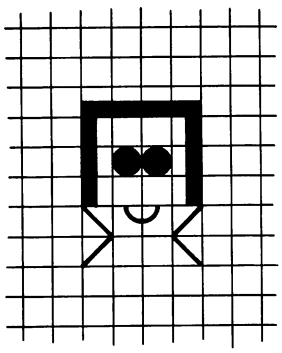


Fig. 5-1. Smiling face.

Anyone you know? For a face you will never forget add this line:

```
70 GOTO 10
```

and the face is printed over and over.

You can place the face anywhere on the screen using the PRINT command. Try this:

```
5 PRINT "♥QQQQ"
```

This clears the screen and starts printing 4 lines down the screen. The Q is the unshifted cursor key CRSR ↓ inside quotation marks. To move the face more toward the screen's center use the TAB function. Add TAB(10) to each of lines 10 to 60. Line 10 would look like this:

```
10 PRINT TAB(10)"■■■■"
```

Now let's use PRINT to make our own space shuttle.

```
10 PRINT "♥QQQQQQQQQQ
20 PRINT "     /\     "
30 PRINT "    |  |    "
40 PRINT "    |  |    "
50 PRINT "   /    \   "
60 PRINT "/________\"
70 PRINT "|        |"
```

Improve on the space shuttle if you want. The graphics on the keyboard offer a lot of flexibility, especially when you learn to use the reverse of the graphics. As your graphic design becomes more complex, you may find graph paper helpful. Using the PRINT command, try to make this new spacecraft.

This is not easy. When you add color to the graphic, you realize how tough graphics like this can become when used inside the PRINT quotation marks.

CREATING GRAPHICS USING POKE

We POKEd sound and color, and we POKE graphics as well. Just as with sound, music, and color, we POKE values into various screen memory locations to create graphics.

SCREEN MAPS

Your VIC has 506 screen memory locations (22 across times 23 down). Count the rows and columns for yourself. Each of these locations has an address. Whenever you want to POKE a character into a location,

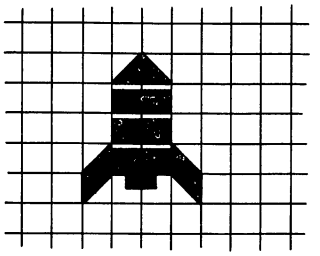


Fig. 5-2. The space shuttle.

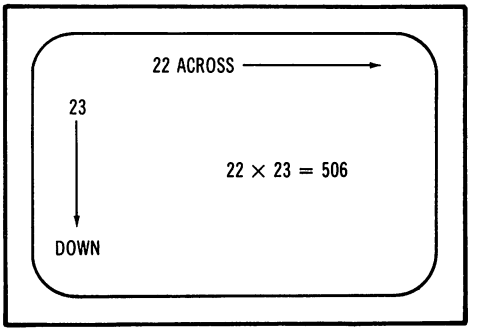


Fig. 5-3. Screen memory locations.

you POKE the character into its "memory address." For characters the addresses range from 7680 to 8185 (from the top lefthand corner to the lower righthand corner).

Each keyboard character has a POKE value. See the appendix in the back of the book for a listing of screen codes for each character. The POKE value for a solid ball is 81. If you want to POKE the solid ball ● into the upper lefthand corner ("home" position), then:

```
POKE 7680, 81
```

No line numbers are needed at this point.
Did you POKE it in? And nothing happened? Well, you must also POKE a color code in memory for the ball to appear.

There are 506 memory locations for color also. The locations start at 38400 and end at 38905. You must POKE in the color with the character for it to appear. The upper lefthand corner of the color code memory map is position 38400. The colors available are coded as such:

BLACK	-0	PURPLE	-4
WHITE	-1	GREEN	-5

RED	-2	BLUE	-6
CYAN	-3	YELLOW	-7

Now try this:

```
POKE 7680,81
POKE 38400,5
```

A solid green ball appears in the upper lefthand position.

The following maps show you all the locations by row and column for character and color memory.

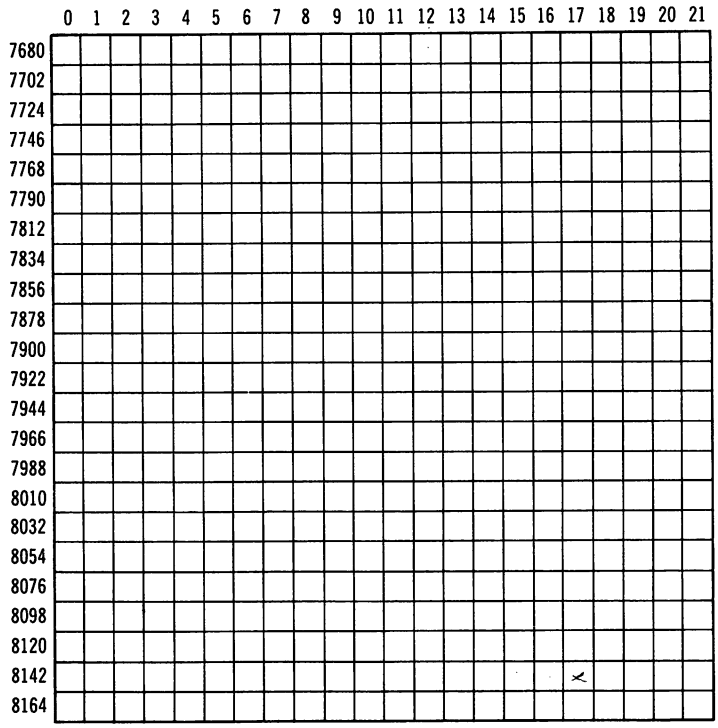


Fig. 5-4. Screen character memory map.

These maps are identical to those in the user's guide which came with the VIC 20.

Looking at the two memory maps, let's devise a program for creat-

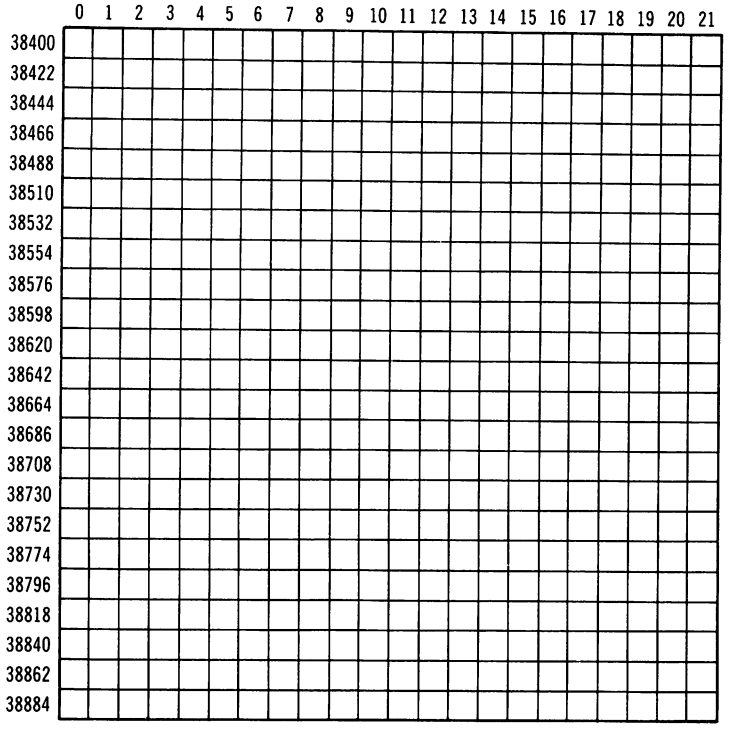


Fig. 5-5. Color code memory map.

ing an additional border around the screen. (We changed screens and borders using color in Chapter 3.) We want a red checkerboard border around our screen. How do we do this?

Screen Border

In order to get the top border filled we have to POKE from 7680 to 7701. The BASIC language offers us a convenient shortcut using FOR . . . NEXT loops. 127 is the character code for ▚ . Type in:

```
10 FOR Z=7680 TO 7701: POKE Z,127: POKE Z + 30720,2:
   NEXT
```

30720 is the difference between 38400 and 7680.
Now let's fill the border on the left side.

```
20 FOR Z=7680 TO 8164 STEP 22: POKE Z,127: POKE Z +
   30720,2: NEXT
```

We use increments of 22 to tell the computer where to POKE the border. Now let's fill the border on the right side:

```
30 FOR Z=7701 TO 8185 STEP 22: POKE Z,127: POKE Z +
   30720,2: NEXT
```

Now the bottom border:

```
40 FOR Z=8164 TO 8185: POKE Z,127: POKE Z + 30720, 2:
   NEXT
```

We could make our program a little shorter by dropping line 30 and changing line 20 to become:

```
20 FOR Z=7680 TO 8164 STEP 22: POKE Z,127: POKE Z +
   30720,2: POKE Z + 21,127: POKE Z + 30741, 2: NEXT
```

See how helpful arithmetic can be? Your border should look like the one in Fig. 5-6.

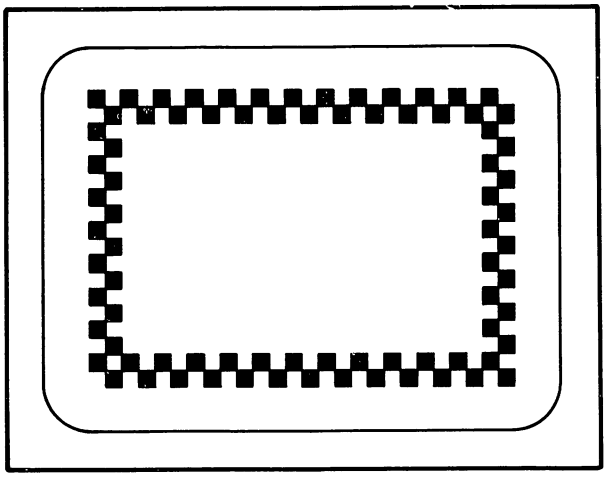


Fig. 5-6. Screen border program.

Change the border color or the character. How about a solid black border—POKE Z,160 and POKE Z + 30720, 0?

Score Box

Using what we've learned about the border, let's make a simple "score box" to be used for games or to enhance other programs. (See Fig. 5-7.)

```
10 FOR Z=7734 TO 7738: POKE Z, 160: POKE Z + 30720,5:
   NEXT
20 FOR Z=7734 TO 7778 STEP 22: POKE Z,160: POKE Z +
   30720,5: 25 FOR 2=7734 TO 7778 STEP 22: POKE Z +
   2,160: POKE Z + 30722,5
30 POKE Z + 4,160: POKE Z + 30724,5: NEXT
40 FOR Z=7778 TO 7782: POKE Z,160: POKE Z + 30720, 5
```

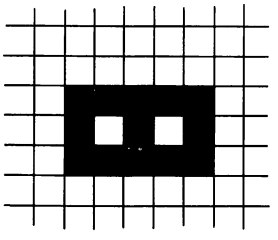


Fig. 5-7. Score box.

Lines 20-30 contain the directions for drawing the three vertical lines; Z, Z+2, and Z+4. Move the score box to another location on the screen. Use the other graphics characters to make a better score box.

TIC TAC TOE BOARD

Let's make a Tic Tac Toe board. Give this a try.

```
100 PRINT "□"
110 FOR Z=7834 TO 7855: POKES Z,160: POKE Z + 30720,0:
    NEXT
120 FOR Z=8010 TO 8031: POKE Z,160: POKE Z + 30720,0:
    NEXT
130 FOR Z=7687 TO 8171 STEP 22: POKE Z,160: POKE Z +
    30720,0: NEXT
140 FOR Z=7694 TO 8178 STEP 22: POKE Z,160: POKE Z +
    30720,0: NEXT
```

This program displays nine boxes of roughly the same size suitable for use in games like Secret Square or Tic Tac Toe.

Let's fill the screen with graphics. In order to fill the screen with diamonds let's input this program.

```
100 PRINT "□"
```

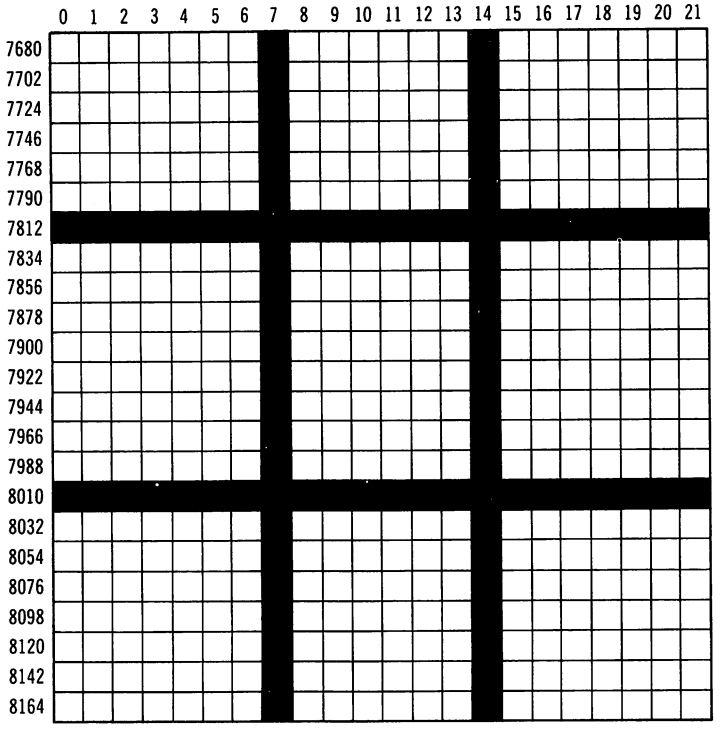


Fig. 5-8. Tic tac toe board.

```
110 FOR Z=7680 TO 8185: POKE Z,90: POKE Z + 30720,4:
    NEXT
```

We like "reversed" diamonds. In order to reverse the character, you add 128 to the POKE value. Therefore, for a diamond which has a POKE value of 90: the reverse value would be 218.

Character Code?

We encourage you to try all the graphics characters on the keyboard. Here is a little program that makes it easy to experiment. Look in the Appendix for the screen codes for each character in order to use this program.

```
10 PRINT "■"
20 INPUT "WHAT CHARACTER CODE";X
30 FOR Z=7680 TO 8185: POKE Z,X: POKE Z + 30720,4: NEXT
```

```
40 FOR T=1 TO 3000: NEXT
50 GOTO 10
```

Use this program to get an idea of what a full screen of keyboard characters looks like. (See Fig. 5-9.) You may want to use this kind of display in games—as we do later in the book. We chose purple for this program—you are welcome to use any color you like.

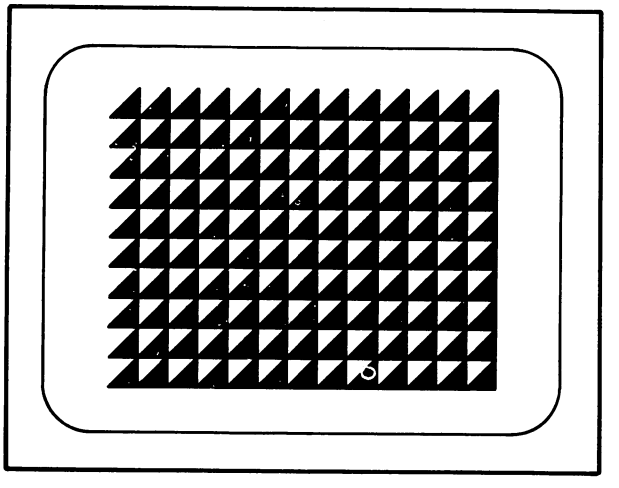


Fig. 5-9. A screen full of characters.

BAR CHARTS OR GRAPHS

Bar graphs are useful because they allow you to compare information graphically. In this program, we show you how you can use bar graphs. Let's pretend you want to compare different cars' acceleration. How long does it take for a car to reach 55 miles per hour? See the results in Fig. 5-10.

```
 10 PRINT "■"
100 PRINT "HOW MANY SECONDS TO REACH 55 MPH?"
110 INPUT "CORVETTE";A
120 INPUT "TOYOTA";B
130 INPUT "DODGE";C
```

```
140 INPUT "VOLKS";D
150 INPUT "MUSTANG";E
160 INPUT "JEEP";F
200 PRINT "■"
210 PRINT "COR";
220 X=A: GOSUB 500
230 PRINT "TOY";
240 X=B: GOSUB 500
250 PRINT "DOD";
260 X=C: GOSUB 500
270 PRINT "VW";
280 X=D: GOSUB 500
290 PRINT "MUS";
300 X=E: GOSUB 500
310 PRINT "JEE";
320 X=F: GOSUB 500
340 END
500 FOR Y=1 TO X
510 PRINT CHR$ (162);
520 NEXT Y
530 PRINT
590 RETURN
```

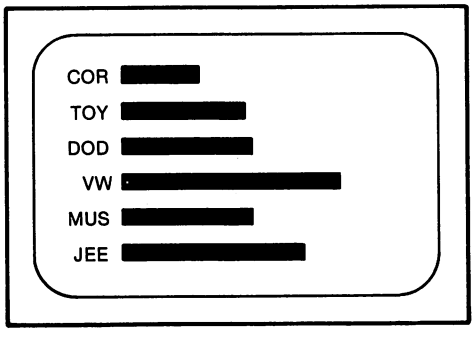
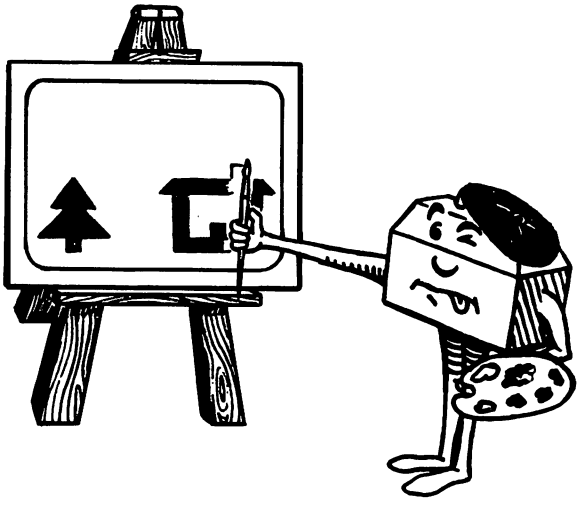


Fig. 5-10. A bar graph comparing acceleration rates.

You can graph a lot of information this way. You can try it out using different graphic characters. For example, we actually like the way the solid ball CHR$(113) looks in a bar chart.

PICTURES WITH THE VIC

You can create pictures of almost any type with the VIC 20. In this next section we show you how to put several graphics together to form a scene. We will also add music and a sneak preview of animation. (We cover animation in the next chapter in detail.)



Pine Tree

Let's draw a pine tree. Using graph paper, we have sketched a tree like the one in Fig. 5-11.

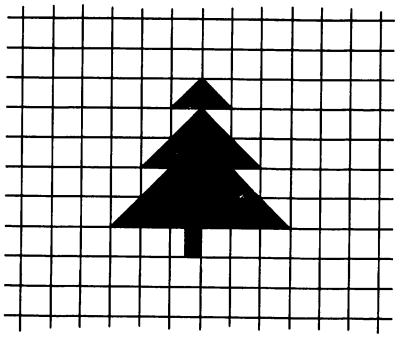


Fig. 5-11. Pine tree.

The program for POKEing the pine tree is:

```
 5 PRINT "■"
20 Z=7680: C=38400
25 REM**PINE TREE
30 POKE Z + 356,233: POKE C + 356,5: POKE Z + 357,223:
   POKE C + 357,5: POKE Z + 378,233: POKE C + 378,5
35 POKE Z + 379,223: POKE C + 379,5
40 POKE Z + 399,233: POKE C + 399,5: POKE Z + 400,160:
   POKE C + 400,5: POKE Z + 401,160: POKE C + 401,5
50 POKE Z + 402,223: POKE C + 402,5: POKE Z + 421,233:
   POKE C + 421,5: POKE Z + 422,160: POKE C + 422,5
60 POKE Z +423,160: POKE C + 423,5: POKE Z + 424,223:
   POKE C + 424,5: POKE Z + 442,233: POKE C + 442,5
70 POKE Z + 443,160: POKE C + 443,5: POKE Z + 444,160:
   POKE C + 444,5: POKE Z + 445,160: POKE C + 445,5
80 POKE Z + 446,160: POKE C + 446,5: POKE Z + 447,223:
   POKE C + 447,5: POKE Z + 466,118: POKE Z + 466,0
```

You should have a green tree with a black trunk near the lower lefthand corner of the screen. Save this program because we will add to it.

Person

Add these lines to the program:

```
85 REM**PERSON
90 POKE Z + 406,81: POKE C + 406,7: POKE Z + 427,78:
   POKE C + 427,7: POKE Z + 428,160: POKE C + 428,7
100 POKE Z + 429,77: POKE C + 429,7: POKE Z + 450,117:
    POKE C + 450,7: POKE Z + 451,117: POKE C + 451,7
```

Your screen should look like Fig. 5-12.

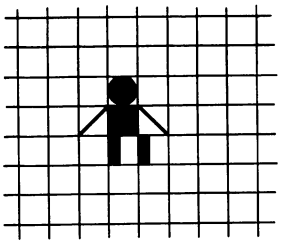


Fig. 5-12. Person.

Now you have a person near the bottom center of the screen. Since we are going to add a house to this "scene," let's show you how we planned

the whole thing. As you can see in Fig. 5-13, we devised special graph paper, showing screen and color code memory locations as variables. This is a handy tool and you're welcome to copy it from the book for your own designing. There is a blank example of this graph paper in the Appendix.

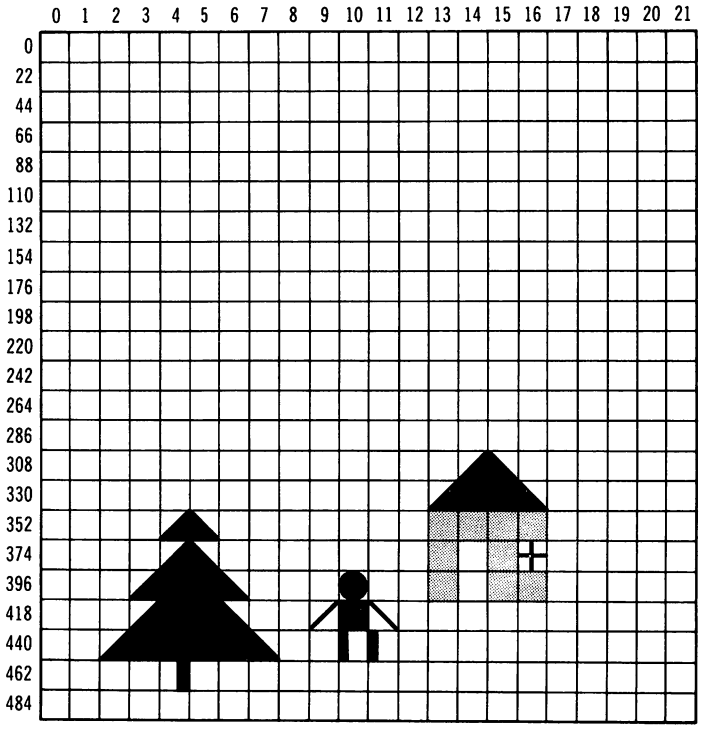


Fig. 5-13. Graphing the "scene."

House

Add these lines to the program to draw a little red house:

```
105 REM**HOUSE
110 POKE Z + 322,233: POKE C + 322,0: POKE Z + 323,223:
    POKE C + 323,0: POKE Z + 343,233: POKE C + 343,0
120 POKE Z + 344,160: POKE C + 344,0: POKE Z + 345,160:
    POKE C + 345,0: POKE Z + 346,223: POKE C + 346,0
```

```
130 POKE Z + 365,160: POKE C + 365,2: POKE Z + 366,160:
    POKE C + 366,2: POKE Z + 367,160: POKE C + 367,2
140 POKE Z + 368,160: POKE C + 368,2: POKE Z + 387,160:
    POKE C + 387,2: POKE Z + 388,160: POKE C + 388,1
150 POKE Z + 389,160: POKE C + 389,2: POKE Z + 390,219:
    POKE C + 390,1: POKE Z + 409,160: POKE C + 409,2
160 POKE Z + 410,160: POKE C + 410,1: POKE Z + 411,160:
    POKE C + 411,2: POKE Z + 412,160: POKE C + 412,2
```

We have POKEd a tree, person and house on the screen. Let's make the scene even more interesting. Add these lines:

```
190 POKE 36878,15                  -This sets the volume for sound.
200 POKE 36879,105                 -This makes the screen blue with
                                    a white border.
210 FOR L=1 TO 100                 -Counts for 100 (100 stars will
                                    appear on the screen).
211 S=36876
212 READ F
214 IF F=999 THEN POKE
    S,0: END
220 POKE 7680 + INT                -POKEs a white star randomly on
    (RND(1)*306),42                 the screen.
225 POKE S,F                       -POKEs the music routine.
230 FOR T=1 TO 20: POKE            -Counts from 1 to 20 and then
    7680 + INT                      POKEs a space where the star
    (RND(1)*306),32                 was. This line also gives the note
                                    duration for the music.
250 NEXT: NEXT
260 GOTO 211
570 DATA
    195,195,215,215,219,219,
    215,209,209,207,207,201,
    201,195,215,215,209,209,
    207,207
580 DATA
    201,215,215,209,209,207,
    207,201,195,195,215,215,
    219,219,215,209,209,207,
    207
```

```
590 DATA 201,201,195
599 DATA 999                    -Include data for "Twinkle,
                                 Twinkle Little Star."
```

Let's start a graphics subroutines library. You probably have some of your own to include.

Spacecraft

Remember the spacecraft we asked you to create using the PRINT command earlier in this chapter? Let's create it with POKE.

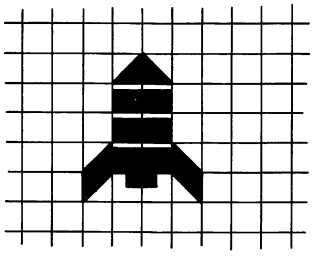


Fig. 5-14. Creating the Space Shuttle with POKE.

```
110 PRINT "■"
120 S=7680: C=38400
130 POKE S + 271,233: POKE C + 271,6: POKE S + 272,223:
    POKE C + 272,6: POKE S + 293,247: POKE C + 293,6
140 POKE S + 294,247: POKE C + 294,6: POKE S + 315,247:
    POKE C + 315,6: POKE S + 316,247: POKE C + 316,6
150 POKE S + 336,233: POKE C + 336,6: POKE S + 337,247:
    POKE C + 337,6: POKE S + 338,247: POKE C + 338,6
160 POKE S + 339,223: POKE C + 339,6: POKE S + 358,105:
    POKE C + 358,6: POKE S + 359,124: POKE C + 359,6
170 POKE S + 360,126: POKE C + 360,6: POKE S + 361,95:
    POKE C + 361,6
```

Dice

Now let's draw some dice that we can use in our "Traditional Games" chapter.

```
100 PRINT "■"
105 S=7680: C=38400
110 POKE S + 360,85: POKE C + 360,0: POKE S + 361,64:
    POKE C + 361,0: POKE S + 362,64: POKE C + 362,0
```

```
115 POKE S + 363,64: POKE C + 363,0: POKE S + 364,73:
    POKE C + 364,0: POKE S + 382,66: POKE C + 382,0
120 POKE S + 386,66: POKE C + 386,0: POKE S + 404,66:
    POKE C + 404,0: POKE S + 408,66: POKE C + 408,0
125 POKE S + 426,66: POKE C + 426,0: POKE S + 430,66:
    POKE C + 430,0: POKE S + 448,74: POKE C + 448,0
130 POKE S + 449,64: POKE C + 449,0: POKE S + 450,64:
    POKE C + 450,0: POKE S + 451,64: POKE C + 451,0
135 POKE S + 452,75: POKE C + 452,0: POKE S + 322,85:
    POKE C + 322,0: POKE S + 323,64: POKE C + 323,0
140 POKE S + 324,64: POKE C + 324,0: POKE S + 325,64:
    POKE C + 325,0: POKE S + 326,73: POKE C + 326,0
145 POKE S + 344,66: POKE C + 344,0: POKE S + 348,66:
    POKE C + 348,0: POKE S + 366,66: POKE C + 366,0
150 POKE S + 370,66: POKE C + 370,0: POKE S + 388,66:
    POKE C + 388,0: POKE S + 392,66: POKE C + 392,0
155 POKE S + 410,74: POKE C + 410,0: POKE S + 411,64:
    POKE C + 411,0: POKE S + 412,64: POKE C + 412,0
160 POKE S + 413,64: POKE C + 413,0: POKE S + 414,75:
    POKE C + 414,0
```

These lines POKE the outsides of the dice. Now let's put faces on each die.

```
500 POKE S + 383,81: POKE C + 383,0: POKE S + 429,81:
    POKE C + 429,0
800 POKE S + 345,81: POKE C + 345,0: POKE S + 391,81:
    POKE C + 391,0
```

The dice look like the ones in Fig. 5-15. Show the dice with two sixes, or all the other possible values.

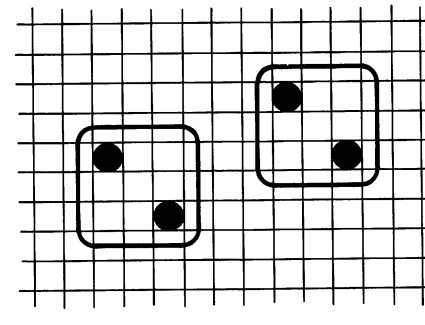


Fig. 5-15. Creating dice.

Three Dimensions

You can POKE in more than two dimensions on the VIC. There is a routine that POKEs a nice staircase for you. We planned it as shown in Fig. 5-16.

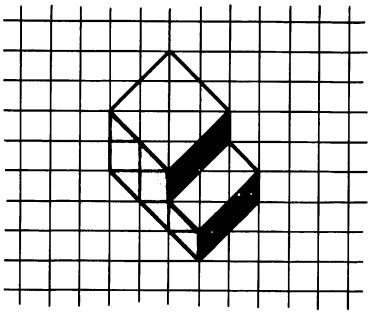


Fig. 5-16. POKEing in three dimensions.

```
10 PRINT "■"
20 S= 7680: C=38400
30 POKE S + 314,78: POKE C + 314,0: POKE S + 315,77:
   POKE C + 315,0: POKE S + 335,78: POKE C + 335,0
40 POKE S + 338,77: POKE C + 338,0: POKE S + 357,223:
   POKE C + 357,7: POKE S + 360,233: POKE C + 360,3
50 POKE S + 379,160: POKE C + 379,7: POKE S + 380,223:
   POKE C + 380,7: POKE S + 381,233: POKE C + 381,3
60 POKE S + 382,105: POKE C + 382,3: POKE S + 383,77:
   POKE C + 383,0: POKE S + 401,95: POKE C + 401,7
70 POKE S + 402,160: POKE C + 402,7: POKE S + 403,105:
   POKE C + 403,3: POKE S + 405,95: POKE C + 405,3
80 POKE S + 424,95: POKE C + 424,7: POKE S + 425,223:
   POKE C + 425,7: POKE S + 426,233: POKE C + 426,3
90 POKE S + 427,105: POKE C + 427,3: POKE S + 447,95:
   POKE C + 447,7: POKE S + 448,105: POKE C + 448, 3
```

Maze Maker

Learn a little more about the graphics characters by playing with this program.

```
10 PRINT "■"
20 PRINT CHR$(109.5 + RND(1));
30 GOTO 20
```

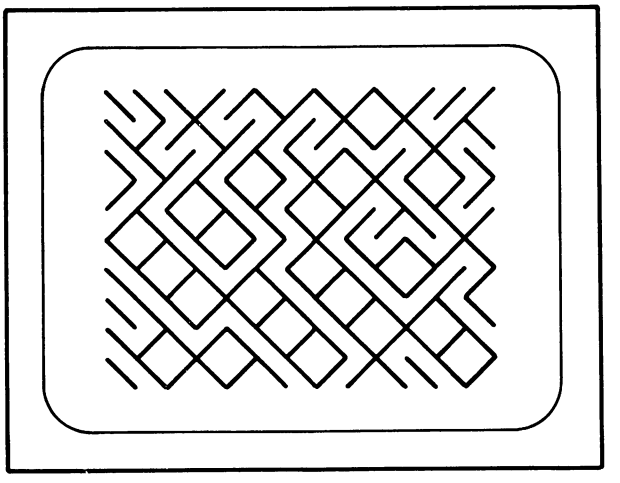
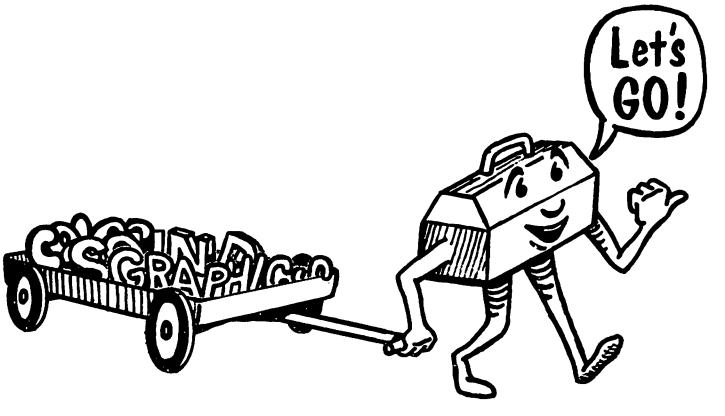
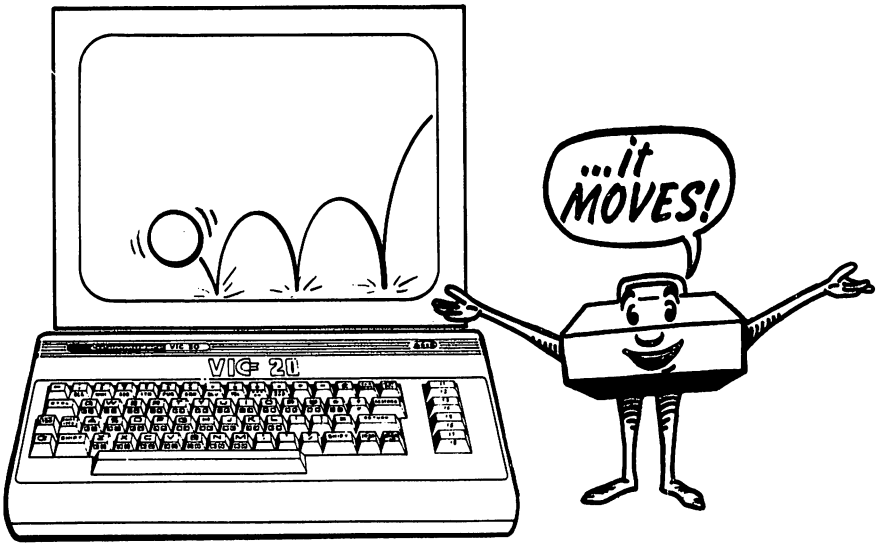


Fig. 5-17. Maze maker.

This displays a maze from characters 109 and 110 (\ and /) picked randomly by line 20 in the program. (See Fig. 5-17.)

The more you experiment with the graphics, the better you get! We use some of the subroutines developed in this chapter in later ones. The twinkling stars on the picture scene gave us a primer for animation. Let's animate our graphics in the next chapter.





Animation Subroutines: The Animation Tool

6

Animation, the movement of figures on the screen, is a lot of fun. Although it may seem difficult, it really is not. Having mastered the graphics routines in the last chapter, you are ready to animate.

HOW DOES THE COMPUTER ANIMATE?

Just like Walt Disney Studios does. Animation is performed by plotting a character in one position, erasing it, and then plotting the character in another position. The computer completes this operation so quickly and efficiently that all we see is the character moving along. Because of the speed, we hardly notice the erasing process. As in graphics, we can animate by using the PRINT command or by POKEing screen locations.

ANIMATION USING THE PRINT COMMAND

To illustrate this process, let's make a blinking ball.

```
10 PRINT "■"
20 PRINT "●"
30 FOR T=1 TO 200: NEXT
40 PRINT "■"
50 FOR T=1 TO 200: NEXT
60 GOTO 20
```

Actually, we have made this ball blink on and off much like the computer blinks the "cursor" for us. If this excites you, try this heart flutter program.

```
10 PRINT "■"
20 PRINT "[S]♥"
30 FOR T=1 TO 300: NEXT
40 PRINT "[S]♥"
50 FOR T=1 TO 300: NEXT
60 GOTO 20
```

The [S] in lines 20 and 40 set the cursor at the home position. Inside quotation marks (for the PRINT command) you press CLR/HOME *without* SHIFT. Since only one character can occupy a place at the same time, each new character "erases" the old one.

Jumping Jack

Let's use the person we made in the graphics chapter for animation. Type this into the VIC.

```
10 PRINT "■"
20 PRINT "●"
30 PRINT "╭ R ▭ ▬ ╮"
40 PRINT "▌ ▐"
50 FOR T=1 TO 500: NEXT
60 PRINT "■"
70 PRINT "●"
80 PRINT "╰ R ▭    "
90 PRINT "╱  ╲"
100 FOR T=1 TO 500: NEXT
110 GOTO 10
```

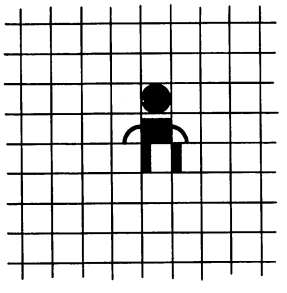


Fig. 6-1. Standing Jack.

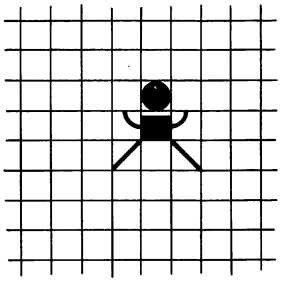


Fig. 6-2. Jumping Jack.

Jack exercises on the screen for us. Lines 30 and 80 use Reverse On and Reverse Off to make the abdomen for Jack. If you are having trouble keying graphics inside quotation marks, look at how line 30 was made:

```
30 PRINT "SHIFT + U,
   CTRL + RVS/ON, C=
   + T, CTRL + RVS/OFF,
   SHIFT + I"
10 PRINT "■"
20 PRINT "●□□□";
```

-Now let's move our "Jumping Jack" figure across the screen using the PRINT command and the cursor controls.

-Prints the head and then moves the cursor left 3 spaces. □ is SHIFT and CRSR inside the quotation marks.

```
25 PRINT "[Q][/][■][\][□][□][□]";
```

–Moves the cursor down 1, prints the arms and body, and then moves the cursor left 3 spaces.

```
30 PRINT "[Q][▌][▌][□][□][□]";
```

–Moves the cursor down 1, prints the 2 legs, and then moves the cursor left 3 spaces.

```
40 FOR T=1 TO 100; NEXT
45 PRINT "[□][□][□]";
```

–45-60 Erase the figure. The last character inside the quotation marks in line 60 ([1]) moves the cursor 1 space to the right, moving the whole process across (horizontally) the screen. It is created with the unshifted CRSR key. [C] is SHIFT and CRSR.

```
50 PRINT "[C]  [□][□][□]";
60 PRINT "[C]  [□][□][□][1]";
70 FOR T=1 TO 100: NEXT
180 GOTO 20
```

Now let's make our figure do exercises while moving across the screen. Add these lines to your program.

```
80 PRINT "[●][□][□][□]";
90 PRINT "[Q][\][■][/][□][□][□]";
100 PRINT "[Q][/][\][□][□][□]";
120 FOR T=1 TO 100: NEXT
130 PRINT "[□][□][□]";
140 PRINT "[C][□][□][□]";
150 PRINT "[C][□][□][□][1]";
160 FOR T=1 TO 100: NEXT
```

As you enter these lines, you should realize that there is repetition. Let's make our program shorter and more efficient by using the GOSUB.

```
10 PRINT "[C][Q][Q]"
20 PRINT "[●][□][□][□]";
25 PRINT "[Q][\][■][/][□][□][□]";
30 PRINT "[Q][▌][▌][□][□][□]";
40 GOSUB 200
80 PRINT "[●][□][□][□]";
90 PRINT "[Q][\][■][/][□][□][□]";
```

```
100 PRINT "Q/\□□□";
120 GOSUB 200
180 GOTO 20
200 FOR T=1 TO 200: NEXT
210 PRINT "□□□";
220 PRINT "□□□□";
230 PRINT "□□□□□]";
240 FOR T=1 TO 50: NEXT
250 RETURN
```

We changed the timing a little in lines 200 and 240—and the animation looks a little more realistic. How can we move the figure vertically (down the screen)? Change line 230 to:

```
230 PRINT "□□□□□Q";
```

How about diagonally across the screen? Change line 230 to:

```
230 PRINT "□□□□□Q]";
```

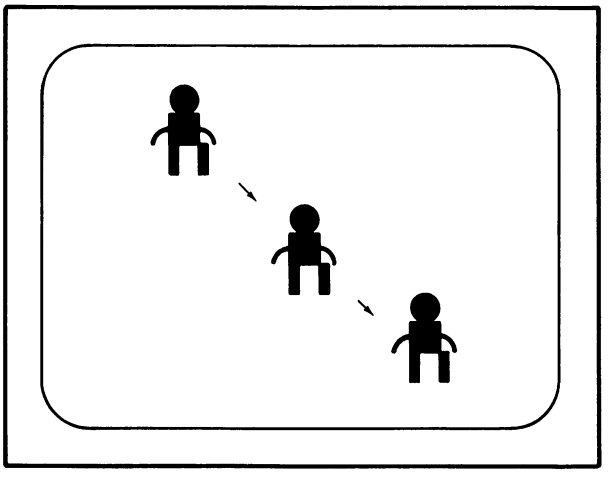


Fig. 6-3. Moving Jack diagonally across the screen.

USING STRINGS IN ANIMATION

We can save some effort by defining the graphics that we want to animate as string variables. Look at this snake example.

```
 10 PRINT "■"
 20 A$= "  ■■■●□□□□□□□"
 30 B$= "     □□□□□□□"
 50 PRINT A$;
 80 GOSUB 100
 90 GOTO 20
100 FOR T=1 to 100: NEXT
110 PRINT B$;
140 FOR T=1 TO 50: NEXT
150 RETURN
```

See how easy string variables are to use? Let's make the snake's tail wiggle when it moves. Add these lines.

```
40 C$= "◤■■■●□□□□□□□"
60 GOSUB 100
70 PRINT C$;
```

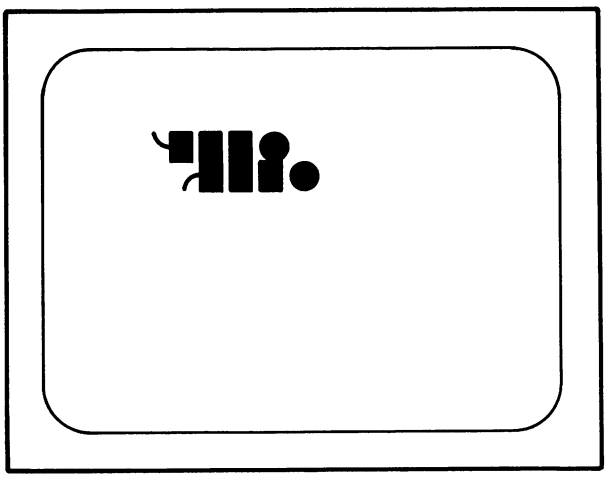


Fig. 6-4. The snake.

Try to animate your own graphics. How about a flying saucer or a car? Make a cruise missile or a cruise ship move across the screen. We have discussed animation using the PRINT command and now will move to animation using POKE.

ANIMATION USING POKE

Let's move a ball around the screen using the POKE statement. Type this in.

```
 10 PRINT "■"
 80 SP=7683                    -80 & 90 Set SP and SC as the start-
                                ing place for the ball.
 90 SC=SP + 30720
110 FOR X=1 TO 500 STEP        -Moves the ball 22 spaces (down
    22                          a column) for a count of 500.
120 POKE SP + X,81             -120 & 130 POKE a black ball.
130 POKE SC + X, 0
150 FOR T=1 TO 100: NEXT
170 POKE SP + X,32             -POKEs a space where the ball
180 FOR T=1 TO 100: NEXT        was, therefore "erasing" it.
190 NEXT X
200 GOTO 110
```

You can change the program to move the ball wherever you want by changing either the starting positions or by changing line 110. Try this: Change line 110 to be:

```
110 FOR X=1 TO 500 STEP 2
```

The ball moves across the screen in a horizontal fashion. To add some randomness to the ball's position enter this:

```
110 X=INT(RND(1)*500) + 1
```

Moving Jack

We can move a graphic figure like our "Jack" using this method. Try this program:

```
 10 PRINT "■"
 80 SP=7683
 90 SC=SP + 30720
110 FOR X=1 TO 500 STEP        -Moves Jack down the screen.
    22
120 POKE SP + X, 81            -120–125 POKE the "Jack" char-
                                acter.
```

```
121 POKE SP + 22 + X,102
122 POKE SP + 21 + X,85
123 POKE SP + 23 + X,73
124 POKE SP + 44 + X,116
125 POKE SP + 45 + X,116
130 POKE SC + X, 0
131 POKE SC + 22 + X,0
132 POKE SC + 21 + X,0
133 POKE SC + 23 + X,0
134 POKE SC + 44 + X,0
135 POKE SC + 45 + X,0
150 FOR T=1 TO 100: NEXT
170 POKE SP + X, 32
171 POKE SP + 22 + X,32
172 POKE SP + 21 + X,32
173 POKE SP + 23 + X,32
174 POKE SP + 44 + X,32
175 POKE SP + 45 + X,32
180 FOR T=1 TO 50: NEXT
190 NEXT X
200 GOTO 110
```

–130–135 POKE Jack black in color.

–170–175 Erase Jack by POKEing a blank space.

You could easily change the direction by changing starting positions and directions in lines 80, 90, and 110.

If we want to control the ball or Jack by testing its location, we can use the following formula to do this:

L=7680 + X + 22*Y where:
L=location
X=row (from 0 to 21)
Y=column (from 0 to 22)

Bouncing Ball

Let's use the formula in this program. See the results in Fig. 6-5.

```
 10 PRINT "■"
 20 X=1: Y=1: ZX=1: ZY=1
120 POKE 7680 + X + 22*Y,81
130 POKE 38400 + X + 22*Y,0
```

```
150 FOR T=1 TO 50: NEXT
170 POKE 7680 + X + 22*Y,32
190 X=X + ZX
200 IF X=0 OR X=21 THEN ZX=-ZX
220 Y=Y + ZY
230 IF Y=0 OR Y=22 THEN ZY=-ZY
250 GOTO 120
```

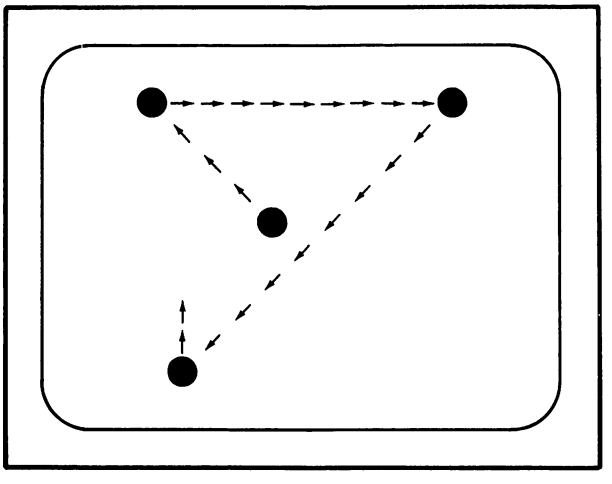


Fig. 6-5. The bouncing ball.

So you see how the X and Y variables move the ball about the screen? Lines 200 and 230 tell the ball to reverse its direction if it touches any of the screen borders.

If you want to, you can make our Jumping Jack bounce around the screen. Pretty strange, but fun to watch.

Speeding Up Jack

We can control and change Jack's speed by "PEEKing" to see if a key has been pressed.

```
10 PRINT "♥QQ"
20 PRINT "●□□□";
25 PRINT "Q◜▩◝□□□";
30 PRINT "Q▌▌□□□";
```

```
 32 F1=39: X=64: F3=47
 33 Z=PEEK(197)
 34 IF Z=X THEN S=150
 35 IF Z=F1 THEN S=25
 36 IF Z=F3 THEN GOTO 10
 40 GOSUB 200
 50 GOSUB 210
 80 PRINT "●□□□";
 90 PRINT "Q◥▩◢□□□";
100 PRINT "Q/ \□□□";
120 GOSUB 200
130 GOSUB 210
180 GOTO 20
200 FOR T=1 TO S: NEXT
205 RETURN
210 PRINT "□□□";
220 PRINT "□□□□";
230 PRINT "□□□□";
250 RETURN
```

This is our original "Jumping Jack" program with lines 32–36 added as well as line 200 broken out as its own subroutine. If function key (F1) is pressed, then Jack's speed is increased. If no key is pressed, Jack moves at his "normal" speed (T=1 TO 150). If function key (F3) is pressed, then the program is reset with Jack in the original position.

Using these keys allows us control over a program while it is running. This proves very useful in game designs.

Use the PEEK for the bouncing ball program or use it for POKEd Jack. Try this logic to control direction as well as speed. Use your imagination and have fun.

Space Commander

You are in control of this spacecraft (shown in Fig. 6-6) because you can: 1) increase the speed by pressing function key 1 (F1); 2) move the craft to the left by pressing function key 3 (F3); and 3) move the craft to the right by pressing function key 5 (F5).

Key in this program:

```
10 S=7680: C=38400
15 POKE 36879, 11
```

–Sets the screen black. If you

make the screen any color other than white, you may POKE characters without POKEing color codes. If you do not POKE color codes (as in this program), the color of the characters is set as white.

```
20 PRINT "■"
50 FOR X=0 TO -506 STEP
   -22                          -Sets the movement of the space-
                                craft up the screen.
52 F1=39: F3=47: F5=55:
   Q=64
53 Z=PEEK(197)
54 IF Z=F1 THEN W= 10
55 IF Z=F5 THEN X=X + 1
56 IF Z=F3 THEN X=X-1
57 IF Z=Q THEN W=100            -52-57 Define the function keys'
                                effect on speed and side move-
                                ment.
60 POKE S + 443 + X,78:         -60 & 70 POKE the spacecraft at
   POKE S + 444 + X,77:         the screen bottom.
   POKE S + 421 + X,106:
   POKE S + 422 + X,116
70 POKE S + 399 + X,233:
   POKE S + 400 + X,~~233~~ 223
90 GOSUB 300
100 POKE S + 443 + X,32:        -100 & 110 Erase the spacecraft.
   POKE S + 444 + X,32:
   POKE S + 421 + X,32:
   POKE S + 422 + X,32
110 POKE S + 399 + X,32:
   POKE S + 400 + X,32
135 NEXT X
140 GOTO 50
300 FOR K=1 TO W: NEXT          -The timing subroutine.
310 RETURN
```

You could add a number of asteroids or enemy space stations. Add these lines to the program:

```
30 FOR D=1 TO 15
32 POKE 7680 + INT(RND(1)*206), 102
34 NEXT
```

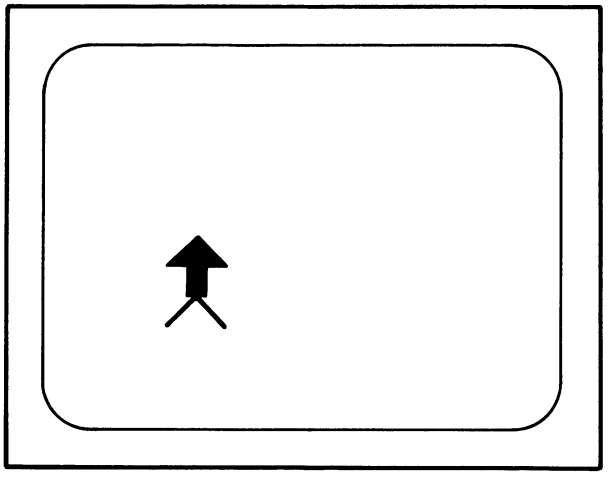


Fig. 6-6. The Space Commander spacecraft.

These lines add 15 asteroids which the spacecraft can eliminate by touching them. You control the craft's movement. Can you clear the sky of the pesky asteroids? This animation routine shows you how such games are conceived. In order to turn this program into a full-fledged game, we would want to add special effects in sound and color, and a scoring condition.

DUCK HUNTING WITH BOW AND ARROW

This program will show you how to combine movement with a shooting effect. The hunter is at the bottom of the screen in a boat (the ♣ graphic character). You can move the hunter to the left and right by pressing function keys 1 and 2.

There are 20 ducks in the sky. When you want the hunter to shoot his arrow at the ducks, press the "F" key. An arrow is released and if the duck is hit, it is erased by the arrow. (See Fig. 6-7.)

```
10 SX=7680
```

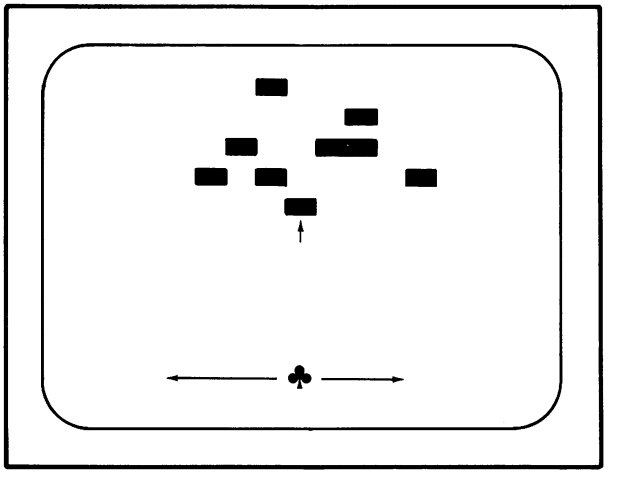


Fig. 6-7. Duck hunting with bow and arrow.

```
15 FL=39: F3=47: MO=64:
   FØ=42                          -Sets the values for the keys to be
                                   pressed.
20 PRINT "■"
30 POKE 36879,11                  -Turns the screen black.
35 FOR V= 1 TO 20                 -35-37 put the 20 ducks randomly
                                   in the sky in the first 306 screen
                                   locations.
36 POKE 7680 + INT(RND(1)
    *306),104
37 NEXT V
40 Y=SX +450                      -Sets 4 as the starting location for
                                   the boat and hunter.
45 S=0: K=Y                       -Sets S and a variable we use in
                                   shooting as 0.
50 X=PEEK(197)                    -Asks if a key is being pressed.
60 IF X=MO THEN D=0               -If no key is pressed, then D (the
                                   direction is the same - 0).
70 IF X=FL THEN D=1               -If function key 1 is pressed, the
                                   hunter is moved to the right.
80 IF X=F3 THEN D=-1              -If function key 3 is pressed, the
                                   hunter is moved to the left.
90 IF X=F0 AND S=0 THEN           -If the FIRE key is pressed, "F",
   S=S + 1                         then S is set as S + 1.
```

```
100 IF S<>0 THEN GOSUB 400      -If S is not equal to 0, then the arrow shooting subroutine begins.
120 POKE Y,32                   -POKEs a space for Y screen location.
130 Y=Y + D                     -Adds the value of D to Y.
160 POKE Y,88                   -POKEs the hunter on the screen.
190 GOTO 50                     -Returns to see if any keys are being pressed.
400 IF S=1 THEN K=Y-22:         -If S equals 1, then screen location K is reduced by 22. The
    POKE K, 30: S=2:            arrow is POKEd, S is increased in
    RETURN                      value and the routine is returned.
410 POKE K,32                   -The arrow is erased.
415 IF S=23 THEN S=0:           -If S=23 (the arrow has left the
    RETURN                      screen) the value of S is set at 0
                                and we are returned to the main
                                program.
420 K=K-22: S=S+1               -For every movement of the arrow, K is decreased by 22 and
                                S is increased by 1.
425 POKE K,30                   -The arrow is POKEd on the screen.
490 RETURN
```

The use of the S variable allows the program to alternate movement from the hunter to the arrow. Since no two characters can move at the same time on the VIC 20, we alternate their movement with the S variable in order to make animation more realistic. We could make our graphics more elaborate and colorful and add sound to develop this into a challenging game of marksmanship.

MARS LANDING

Let's practice landing a flying saucer on the surface of Mars. You control the saucer using function keys 1, 3, and 5. (See Fig. 6-8.)

```
10 PRINT "□"
15 POKE 36879,11                -Turns the screen black.
```

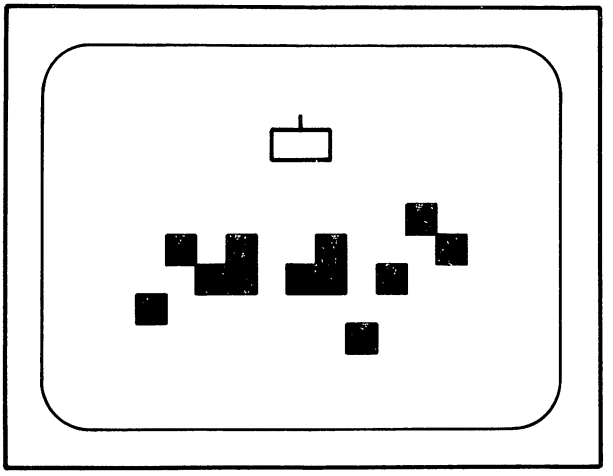


Fig. 6-8. Mars landing.

```
16 POKE 36878,15          -Sets the volume for sound.
20 GOSUB 400
30 X=0                    -Sets X at 0. X is the column
                           variable.
35 FOR Y=0 TO 23 STEP 1   -Sets Y from 0 to 23; the number
                           of rows in the screen.
40 F1=39: F3=47: Z0=64:   -Sets values for the function keys
   F5=55                   and if no key is pressed (Z0)
45 K=PEEK(197)            -Asks if a key is being pressed.
50 IF K=F1 THEN X=X + 1   -If function key 1 is pressed then
                           the saucer is moved one space
                           to the right.
55 IF K=Z0 THEN Q=40      -If no key is pressed, the saucer's
                           speed is set at the normal rate
                           (T=1 TO 40).
60 IF K=F3 THEN X=X-1     -If function key 3 is pressed, then
                           the saucer is moved one space
                           to the left.
65 IF K=F5 THEN Q=220     -If function key 5 is pressed the
                           saucer is slowed down.
90 IF PEEK (7712 + X +    -If the lower part of the saucer
   22*Y)=102 OR PEEK(7713  touches the Mars "surface," the
```

```
    X + 22*Y)=102 OR PEEK
    (7714 + X + 22*Y)=102
    THEN GOSUB 800
120 POKE 7690 + X + 22*Y,
    112: POKE 7691 + X +
    22*Y,113:POKE 7692 +
    X + 22*Y,110
130 POKE 7712 + X + 22*Y,
    109: POKE 7713 + X +
    22*Y,67: POKE 7714 +
    X + 22*Y,125
150 GOSUB 300
160 POKE 7690 + X + 22*Y,
    32: POKE 7691 + X + 22
    *Y,32: POKE 7692 + X +
    22*Y
170 POKE 7712 + X + 22*Y
    ,32: POKE 7713 + X + 22
    *Y,32: POKE 7714 + X +
    22*Y,32
175 NEXT Y
200 GOTO 10
300 FOR T=1 TO Q: NEXT
310 RETURN
400 FOR K=1 to 100
410 M=7680 + INT(RND(1)*
    506)+400
420 POKE M,102
430 NEXT K
450 RETURN

800 POKE 36879,40
805 POKE 36876,200
810 FOR T=1 TO 100: NEXT
815 POKE 36876,0
820 RETURN
```

program moves to Subroutine 800.

–120–130 POKE the saucer graphic.

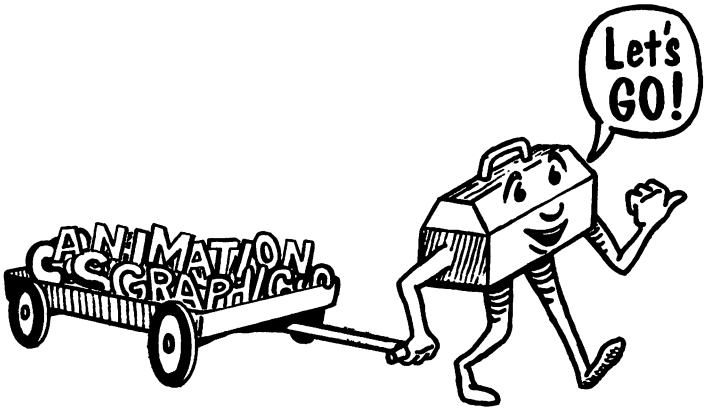
–160–170 Erase the saucer by POKEing spaces.

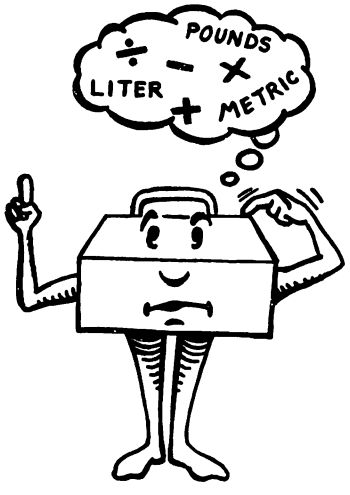
–The timing subroutine.

–POKEs the Mars surface obstacles in the lower part of the screen.

When the saucer hits the Mars surface, the screen turns red and beeps for every obstacle hit.

Use your own imagination to develop animation routines for the VIC 20. You can animate almost any graphics character, using color and sound to make the sight more exciting. PEEKing at where one character is in relation to another is the basis of arcade game development.





Calculating Subroutines for the VIC 20: Facts, Figures and Conversions

7

Your VIC 20 can easily become a helpful calculator and problem solver. We think it is smart to use your computer in this way, and you'll be learning how to develop subroutines for fun math games and quizzes in the next chapter.

Don't be anxious about math, because our gentle approach shows you how easy arithmetic and algebra are on the VIC. Let's start by reviewing simple arithmetic operations.

SIMPLE CALCULATIONS ON THE VIC

All you have to do to perform simple calculations is use the PRINT statement. Try it.

```
PRINT 2 + 2                  -(and press RETURN)
PRINT 5 - 2                  -(and press RETURN)
PRINT 5 * 2                  -(The multiplication sign is the
                               asterisk *.)
PRINT 6 / 2                  -(The slash / is the division sign.)
PRINT 4 ↑ 2                  -(The ↑ is used for exponents. 4
                               ↑ 2 is the same as 4^2.)
PRINT 10 * (2 / 5)           -(The parentheses tell the com-
                               puter which operation to per-
                               form first.)
```

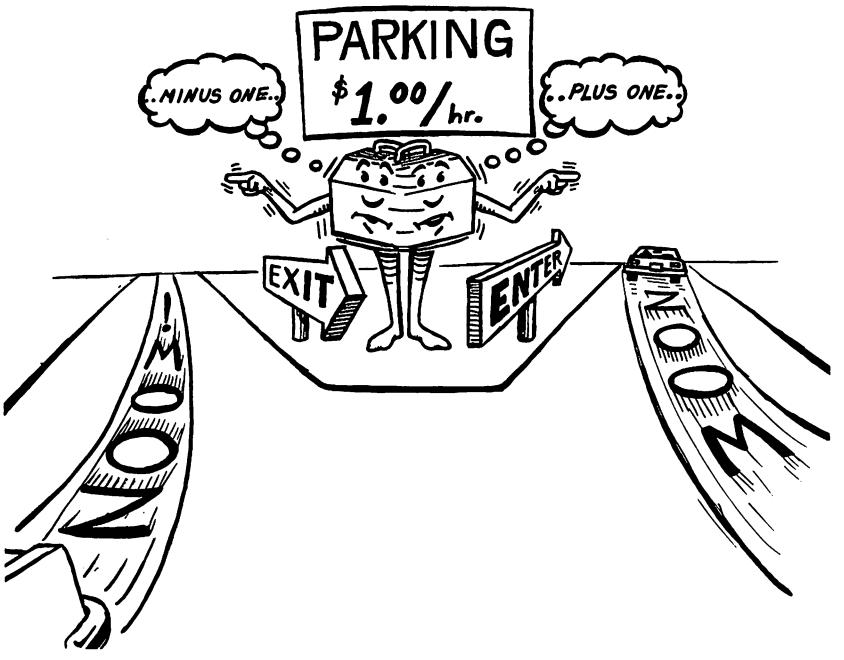
(NOTE: Without the parentheses, the computer follows rules of priority. These priorities are shown in Table 7-1.)

Table 7-1. Priorities in Mathematical Operations

Operation	Symbol	Priority
exponent	↑	highest priority
multiplication	*	⋮
division	/	⋮
addition	+	⋮
subtraction	-	lowest priority

Let's look at some examples to see how you and the VIC can handle them.

Parking Lot



At your favorite parking lot there are 20 rows of cars with 15 cars in each row. How many cars are there?

```
PRINT 20 * 15
300
```

There are 300 cars.

In the same full parking lot, four cars leave every hour. Two new cars arrive at the same rate, every hour. After three hours, how many cars are in the lot?

Step by step, we can work out the problem:

300 cars in the full lot	300
3 hours times 4 cars leaving	− (3 * 4)
3 hours times 2 cars arriving	+ (3 * 2)

Input this line:

```
PRINT 300 − (3 * 4) + (3 * 2)
294
```

There are 294 cars in the lot after three hours.

* *Problem 1*

What if 10 cars leave every hour, and 6 arrive at the same rate? How many would there be in 4 hours?

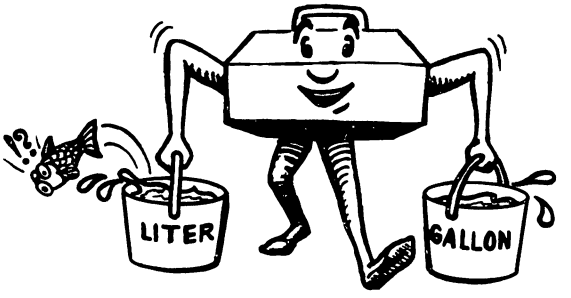
There are other numeric functions your VIC can perform. We encourage you to learn about square roots, logarithms, cosines, and the like in the manual that came with the VIC.

(*NOTE: The answers to all problems are at the end of each chapter.)

CALCULATING SUBROUTINES

We're going to show you how to solve problems by using short programs, or subroutines, on the VIC. First, let's look at this problem.

Gallons and Liters



If one liter is the same as .2642 gallons, how many gallons are in 60 liters.

We can solve this problem by multiplying 60 liters times .2642 to get the correct number of liters. Using the PRINT statement, this would be:

```
PRINT 60 * .2642
```

If we want to convert lots of different amounts of liters into gallons, we can write the program.

L = liters
G = gallons

```
10 INPUT L                  -Asks you to input a value for L.
20 LET G = .2642 * L        -Defines G as a function of L or in
                             relation to L.
30 PRINT G                  -Tells the computer to print Gs
                             value.
```

Type this program in and see what happens. The ? (question mark) prompts you to key in an amount of liters and then prints the amount of gallons.

Let's make the program more "friendly." Change line 10 to read:

```
10 INPUT "# of LITERS" ; L
```

and change line 30 to read:

```
30 PRINT L "LITERS IS THE SAME AS" G "GALLONS"
```

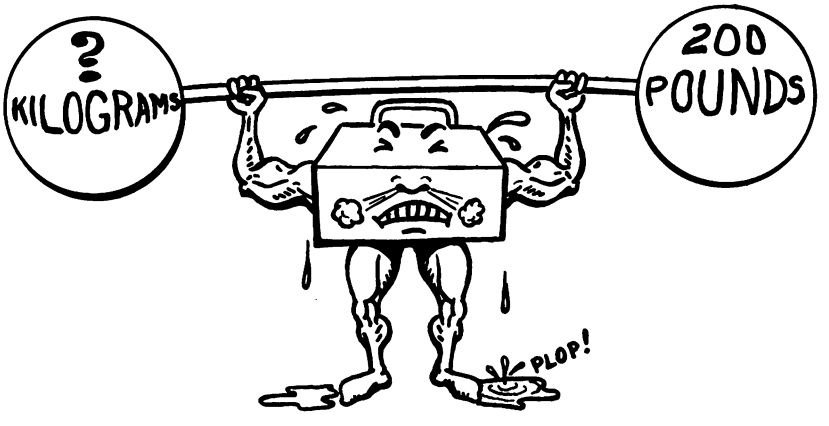
Now the program proudly asks you to input the # of liters and displays for you in plain English liters to gallons conversion.

Kilometers and Miles

Problem 2

Write a subroutine for converting kilometers from miles. For every kilometer, there is 0.621 mile.

Pounds, Ounces and Kilograms



Let's do a slightly more complex subroutine by adding another variable to the input.

1 pound = 2.205 kilograms

We want the user to input the number of pounds and ounces in order to convert to kilograms. Try this program out:

```
5 PRINT "♥"
```
-Starts the program on a clear screen. In order to get the "heart" inside the quotes, you key the SHIFT and CLR/HOME at the same time. Try the program with and without line 5.

```
10 PRINT "FIRST POUNDS,
   THEN OUNCES"
```
-Instructs you to first input pounds, then ounces.

```
20 INPUT P, OZ
```
-Asks you to give values to P and OZ.

```
30 LET PZ = P + OZ / 16
```
-Defines PZ in relation to pounds and ounces (P and OZ).

```
40 KG = PZ / 2.205
```
-Defines KG in relation to pounds.

```
50 PRINT PZ "POUNDS ARE"
   KG "KILOGRAMS"
```
-Tells the computer to print pounds and kilograms for the values you have assigned.

Inches and Centimeters

One inch equals 2.54 centimeters. Let's write the simple subroutine for solving centimeters:

```
10 INPUT "HOW MANY INCHES" ; I
20 LET CM = 2.54 * I
30 PRINT I "INCHES IS APPROXIMATELY" CM "CENTI
METERS"
```

Problem 3

Using what you learned in the pounds/kilograms subroutine, can you change this into a "feet, inches, and centimeters" conversion subroutine?

Temperature

Converting from Fahrenheit to Celsius is not easy, but this subroutine should help:

```
10 INPUT "DEGREES
   FAHRENHEIT" ; F
20 LET C = .55556 * (F–32)
30 PRINT F "DEGREES
   FAHRENHEIT IS" C
   "DEGREES CELSIUS"
40 GOTO 10
```

–Lines 10 to 30 are similar to other subroutines developed.

–Tells the computer to go back to line 10 and prompt you for another Fahrenheit value. This line turns the program into a continuous loop. To stop it, key in RUN/STOP and RESTORE at the same time.

Dollars and Deutschmarks

Have you ever looked at the foreign rates of exchange in the financial section of the newspaper? This subroutine will show you dollar amounts and the equivalent in Deutschmarks. In 1983, one dollar was worth approximately 2.55 marks in West Germany.

```
10 PRINT "■"
20 PRINTS "DOLLARS",
   "MARKS"
30 D = O
40 PRINT D, 2.55 * D
50 D = D + 5
60 IF D < 30 THEN 40
70 STOP
```

–Starts the program on a clear screen.

–Tells the computer to divide the screen in half (because of the comma) and put dollars in one side and marks in the other.

–Assigns O to D.

–Tells the computer to divide the screen in half (because of the comma) and puts D's value on the left and 2.55 * D on the right.

–Tells the computer to add 5 to its value.

–Says that as long as D is less than 30, then to print D and 2.55 times D.

–Tells the program to stop.

Type the program into the computer and RUN it. Do you like the table format?

What happens when you change line 60 to

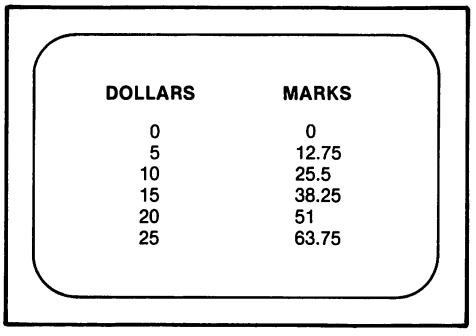


Fig. 7-1. Dollars and Deutschmarks.

IF D < 50 THEN 40?

How about changing the increment in line 50 from D + 5 to D + 2? Experiment with this table format.

Problem 4

You realize that exchange rates change almost daily. Change the Dollars and Deutschmarks subroutine so that you can input the most current exchange rate.

Problem 5

Wouldn't it be interesting to display Fahrenheit and Celsius conversions in table format? Go back to the Temperature subroutine and prepare a new program.

Investing

How much money can you make by investing? This program shows you how easy it is to make money when you have some. The standard formula for return is:

$$R = M * (1 + I/100)^{Y} \text{ where}$$

R= the return
M= money invested
I= interest rate per period
Y= the number of periods

```
10 PRINT "■"
20 PRINT "AMOUNT OF MONEY TO INVEST": INPUT M
```

```
30 PRINT "WHAT IS THE INTEREST RATE PER YEAR": INPUT I
40 PRINT "HOW MANY YEARS": INPUT Y
50 LET R = M * (1 + I/100) ↑ Y
60 PRINT "YOUR RETURN ON" M "INVESTED IS" R
```

None of the lines used in this subroutine are new except for lines 20 and 30, which combine the PRINT and INPUT statements using the colon.

This is a fun subroutine to play with. We suggest you add another line so you don't have to type in NEW after each run.

```
70 GOTO 20
```

These continuous loops are easy to make. What happens if you had typed in

```
70 GOTO 10?
```

Profit

Most people are in business to make money, and making money means profits. How do we figure profit?

Profit = Total Revenues − Total Costs

Now let's do a "profitable subroutine."

```
10 PRINT "■"
20 PRINT "WHAT WERE YOUR TOTAL REVENUES": INPUT R
30 PRINT "WHAT WERE YOUR TOTAL COSTS": INPUT C
40 LET P = R - C
50 PRINT P "IS YOUR TOTAL PROFIT"
```

Easy enough. Now we should figure at what "rate" we are profitable. When comparing businesses, we look at the profit as a percentage of revenue. By using this ratio, businesses' performances can be compared regardless of their size. We have to add the following lines to get this ratio:

```
60 LET PP = P/R * 100
```

-Defines PP (Profit as a % of Revenue) as Profit divided by Revenues times 100.

```
70 PRINT PP "IS YOUR
   PROFIT AS A PERCEN
   TAGE OF REVENUES"
```

-Simply prints the value of PP with a message.

Now RUN the whole program.

Current Ratio

Banks and financial analysts often look at your current ratio as a measure of your "financial health." It boils down to how much you have and what you owe. The simple formula is:

Current Ratio = Current Assets/Current Liabilities

Problem 6

Write a subroutine for figuring current ratio.

Loan Analysis

Using what we know about the Current Ratio in the last subroutine, let's figure a way to gauge a person's (or business') creditworthiness in a loan situation.

Some banks won't loan you money if your liabilities exceed your assets (or your current ratio is less than 1). Let's assume that banks will only grant loans if the current ratio is greater than 1.

```
10 PRINT "■"
20 PRINT "WHAT ARE
   YOUR CURRENT
   ASSETS": INPUT A
```

```
30 PRINT "WHAT ARE
   YOUR CURRENT LIABILI
   TIES": INPUT L
40 LET R = A/L
50 IF R > 1 THEN 100
60 IF R < = 1 THEN 200

100 PRINT "YOUR CURRENT
    RATIO IS" R, "YOUR
    LOAN IS ACCEPTED."
105 STOP
200 PRINT "YOUR CURRENT
    RATIO IS" R, "YOUR
    LOAN IS DENIED"
205 STOP
```

-Tells the computer that if the current ratio (R) is greater than 1 to go to line 100.

-Tells the computer that if R is less than or equal to 1 to go to line 200.

-Prints the current ratio and tells you that the loan is accepted.

-Prints the current ratio and tells you that the loan is denied.

Problem 7

Using what you have learned in this Loan Analysis case, go back to the Profit subroutine and add lines to the program so that:

-if the percentage of profit is more than 20%, "the company is in good health."
-if the percentage of profit is less than 20%, "the company is performing poorly."

ANSWERS TO CHAPTER 7 PROBLEMS

Problem 1

Parking Lot

```
PRINT 300 - (4 * 10) + (4 * 6)
284
```

Problem 2

Kilometers and Miles

```
10 INPUT "MILES"; M
20 LET KM = 0.621 * M
30 PRINT M "MILES EQUAL" KM "KILOMETERS"
```

Problem 3

Feet, Inches and Centimeters

```
10 PRINT"FIRST FEET, THEN INCHES"
20 INPUT F, I
30 LET FI = I + (F * 12)
40 LET CM = 2.54 * FI
50 PRINT FI "INCHES IS APPROXIMATELY" CM
   CENTIMETERS"
```

Problem 4

Exchange Rate

```
10 INPUT "EXCHANGE RATE"; EX
20 PRINT "DOLLARS", "MARKS"
30 D = O
40 PRINT D, EX * D
50 D = D + 5
60 IF D < 30 THEN 40
70 STOP
```

Problem 5

Temperature Table

```
10 PRINT "■"
20 PRINT "FAHRENHEIT", "CELSIUS"
30 F = 0
40 PRINT F, .55556 * (F–32)
50 F = F + 20
60 IF F < = 220 THEN 40
70 STOP
```

Problem 6

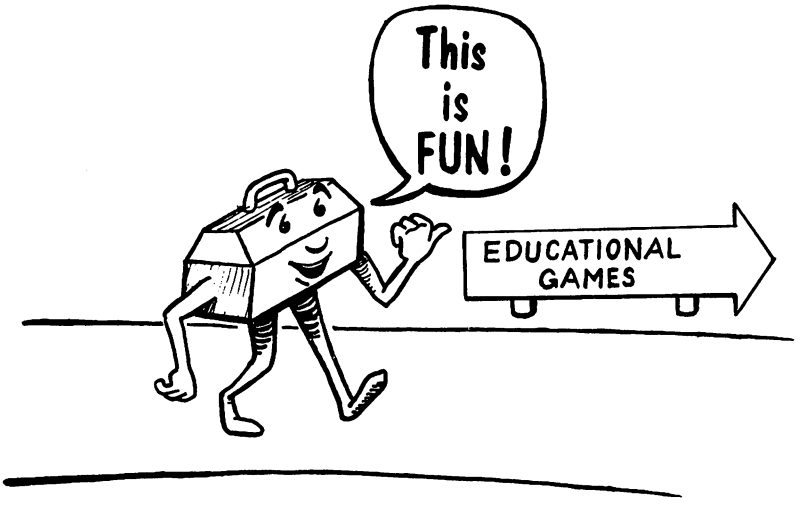
Current Ratio

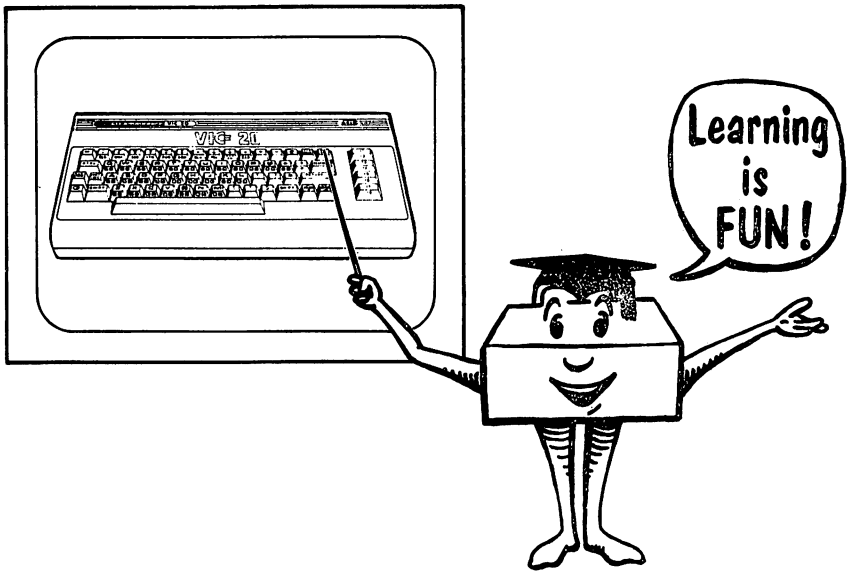
```
10 PRINT "■"
20 PRINT "WHAT ARE YOUR CURRENT ASSETS": INPUT A
30 PRINT "WHAT ARE YOUR CURRENT LIABILITIES": INPUT L
40 LET R = A/L
50 PRINT R "IS YOUR CURRENT RATIO"
```

Problem 7

Profit Analysis

```
 70 IF PP > = 20 THEN 100
 80 IF PP < 20 THEN 200
100 PRINT "YOUR PERCENTAGE PROFIT IS" PP, "AND YOUR
    COMPANY IS IN GOOD HEALTH"
105 STOP
200 PRINT "YOUR PERCENTAGE PROFIT IS" PP, "AND THE
    COMPANY IS PERFORMING POORLY"
205 STOP
```





Educational Games

8

The educational games in this chapter are designed so that you can have fun while learning.

We explain the structure of each program and encourage you to improve and enhance it with graphics, color, and sound wherever you like. The logic for these games can be used to create many more, and we will give you lots of ideas for those, too.

One of the main reasons we use a computer and write our own programs is to teach our children math and general educational concepts. Our kids enjoy quizzing each other and their parents about all kinds of things, from states and capitals to riddles. We hope you have as good a time with these games as we do.

ARITHMETIC TESTER

This program, (shown in Fig. 8-1), asks you to add numbers. It stops the game after 10 questions, gives you the score, and ranks you on three levels of competence. Type it in and try it out.

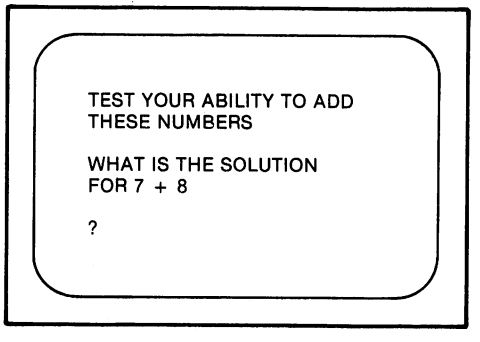


Fig. 8-1. Arithmetic tester.

```
10 PRINT "■"
20 PRINT "TEST YOUR ABIL-
   ITY TO ADD THESE
   NUMBERS"
30 Q = 0          -Starts the number of questions
                   asked at 0.
40 N = 0          -Starts the number of correct
                   answers at 0.
```

```
50 A = INT(RND(1) *
   10) + 1
```
-Lines 50 and 60 state A and B are integers (whole numbers) randomly selected from 1 to 10.

```
60 B = INT(RND(1) * 10) + 1
```

```
70 IF Q = 10 THEN GOTO
   240
```
-If the number of questions asked is 10, then the scoring subroutines are called.

```
80 Q = Q + 1
```
-Q is defined as 0 in line 30, and 1 will be added each time a question is asked.

```
90 PRINT
100 PRINT "WHAT IS THE
    SOLUTION FOR" A
    "PLUS" B
110 PRINT
120 INPUT X
```
-The computer asks for the answer to A plus B.

-The computer asks you for the answer.

```
130 PRINT
140 IF X = A + B THEN 190
```
-If your answer is right, then the program moves to lines 190-230.

```
150 PRINT "WRONG"
```
-In lines 150 and 160 if your answer is not right, the computer prints "WRONG" and the correct answer.

```
160 PRINT A + B "IS THE
    CORRECT ANSWER"
170 FOR T = 1 TO 1000 :
    NEXT T
180 GOTO 220
```
-Tells the computer to skip 190-210 and returns the program to the beginning because of the directions in 230.

```
190 PRINT "CORRECT"
200 FOR T = 1 TO 1000 :
    NEXT T
210 N = N + 1
```
-Counts the number correct for all the questions asked.

```
220 PRINT
230 GOTO 50
240 GOTO 250
```

```
250 PRINT N "QUESTIONS
    RIGHT OUT OF" Q
260 J = N/Q

270 IF J = 1 THEN GOSUB
    3000


280 IF J < 1 AND J > =.7
    THEN GOSUB 4000
290 IF J < .7 THEN GOSUB
    5000
300 END
3000 PRINT "VERY GOOD
    WORK"
```

-This is your score.

-This line turns your score into a fraction-decimal.

-Lines 270 to 290 tell the computer which subroutine to go to, depending upon the percentage answered correctly.

(Add your own sound or color routines here for program dressing.)

```
3400 FOR T = 1 TO 1500 :
    NEXT T
3500 RETURN
4000 PRINT "GOOD WORK,
    BUT MORE PRACTICE
    WILL HELP"
```

(More room for program dressing.)

```
4400 FOR T = 1 TO 1500 :
    NEXT T
4500 RETURN
5000 PRINT "YOU NEED TO
    PRACTICE A LOT"
```

(More room for program dressing.)

```
5400 FOR T = 1 TO 1500 :
    NEXT T
5500 RETURN
```

Suggested Program Changes and "Dressing"

You can enhance or dress up the program by adding sound, color, and graphics. Look at your "tools" chapters for ideas. In this program, there are several natural places for program enhancement: 1) when a

correct answer is entered; 2) when an incorrect answer is entered; or 3) when the score for the round is displayed using subroutines 3000–5000.

The enhancement can be as simple as using a different screen and border combination, or as complex as creating a smiling face with changing background colors and music for a job well done.

The program can be easily altered to suit your needs in many ways.

1. **LEVEL.** Change the level of difficulty by changing lines 50 and 60. To make the quiz harder, change the statement to read

   INT(RND(1) * 100) + 1

   for both A and B. Now you will have addition problems for numbers 1 to 100.
2. **LENGTH.** You can make the quiz longer by changing the number of questions asked before the program ends. Adapt line 70 to

   IF Q = 20 THEN GOTO 240

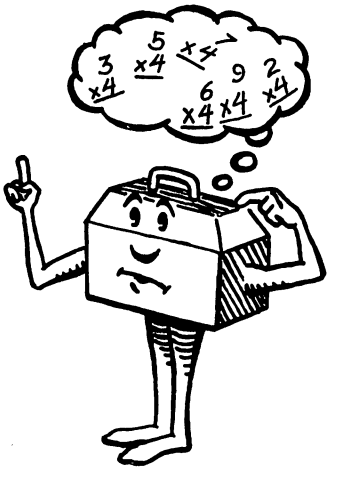
   Now you have 20 questions. You could change it to anything—even 1000 or more.
3. **GRADING.** You can set your own grades or competency levels by changing lines 270 to 290. If you would rather have "very good work" mean .9 (or 90%) and higher, change the lines to read:

   270 IF J >= .9 THEN GOSUB 3000
   280 IF J < .9 and J >= .7 THEN GOSUB 4000
   290 IF J < .7 THEN GOSUB 5000

4. **OPERATION.** You can change this from addition to any other arithmetic operation you want, such as subtraction, division, and multiplication. How? For subtraction change (A + B) to (A − B) in lines 140 and 160. Change the print statements in lines 20 and 100 to read MINUS, and you're done.

MULTIPLICATION TABLES

Using the same program structure as "Arithmetic Tester" and using READ-DATA statements, this program tests your ability to multiply the numbers provided in DATA statements in the program. The number of questions asked is the same as the amount of data. When your answers are all correct, the computer smiles for you. (See Fig. 8-2.)




```
10 PRINT "■"
20 PRINT "THIS PROGRAM
   WILL HELP YOU LEARN
   YOUR MULTIPLICATION
   TABLES—HAVE FUN"
30 Q = 0
40 N = 0
50 READ A
60 IF A = 99 THEN 240
70 READ B
80 Q = Q + 1
90 PRINT "■"
100 PRINT "WHAT IS THE
   ANSWER FOR" A
   "TIMES" B
110 PRINT
120 INPUT X
130 PRINT
140 IF X = A * B THEN 190
```

-Tells the computer to read Data in A(11,12,15,14,etc., in this case). These are numeric variables.

-Data in line 700 is our way of telling the program to stop rather than go OUT OF DATA.

-Reads DATA in B(3,6,9,4,etc., in this case). These are numeric (not string) variables.

```
 150 PRINT "WRONG"
 160 PRINT A * B "IS THE
     RIGHT ANSWER"
 170 FOR T = 1 TO 1000 :
     NEXT T
 180 GOTO 220
 190 PRINT "CORRECT"
 200 FOR T = 1 TO 1000 :
     NEXT T
 210 N = N + 1
 220 PRINT
 230 GOTO 50
 240 GOTO 250
 250 PRINT N "QUESTIONS
     RIGHT OUT OF" Q
 260 J = N/Q
 270 IF J = 1 THEN GOSUB
     3000
 280 IF J < 1 AND J > = .6
     THEN GOSUB 4000
 290 IF J < .6 THEN GOSUB
     5000
 500 DATA
     11,3,12,6,15,9,14,4,10,
     11,13,12,15,6,11,12,13,
     5,7,14
 700 DATA 99
1000 END
3000 PRINT "EXCELLENT
     WORK"
3010 S = 7680 : C = 38400
3020 POKE S + 404,87:
     POKE S + 406,87:
     POKE S + 427,81:
     POKE S + 448,77:
     POKE S + 449,100:
     POKE S + 45O,78
3030 POKE C + 404,5: POKE
     C + 406,5: POKE
     C + 427,5: POKE
```

```
        C + 448,5: POKE
        C + 449,5: POKE
        C + 450,5
3050 RETURN
4000 PRINT "GOOD WORK.
        PRACTICE MAKES
        PERFECT"
4050 RETURN
5000 PRINT "POOR WORK.
        PRACTICE MORE"
5050 RETURN
```

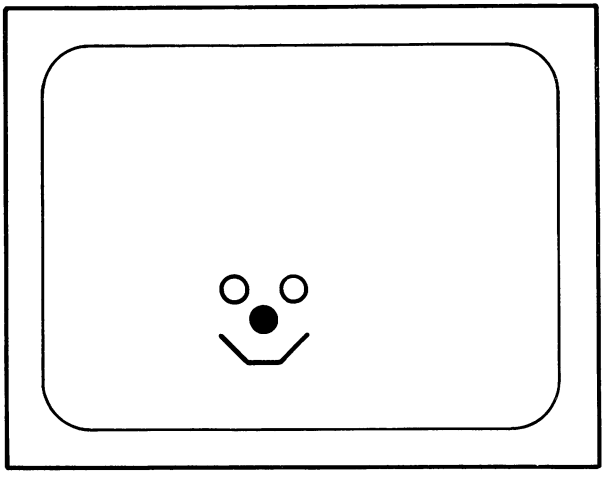


Fig. 8-2. The computer smiles for you.

Suggested Program Changes and "Dressing"

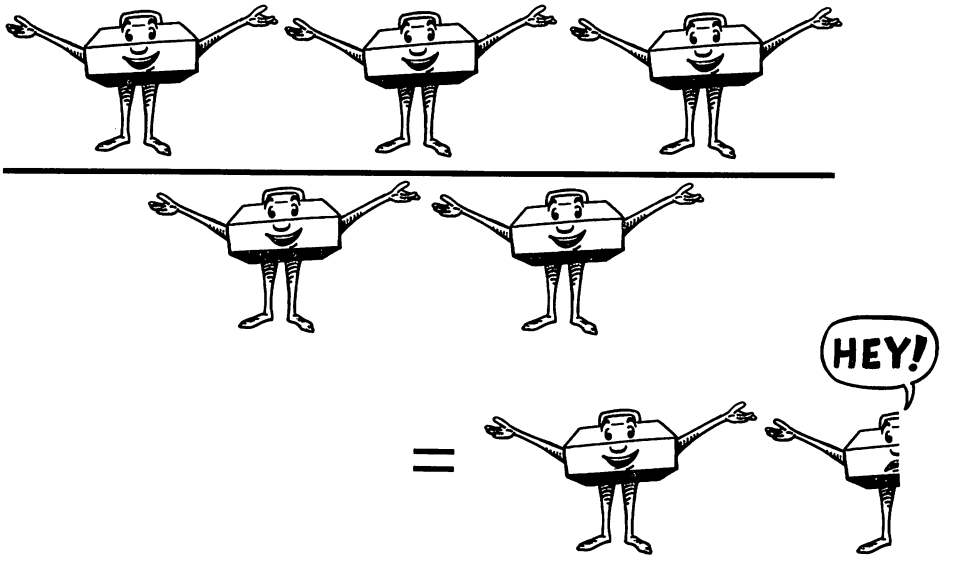
Add sound, color, and more graphics to the subroutines. We have used a smiling face to encourage you to be creative. Use a frowning face for subroutine 5000. Can you make the eye wink? Use the Random Color program in the "VIC Color" chapter for good scores. You want music? Look at what you learned in "VIC Sound and Music" for some fun routines.

Change the program in these ways to suit you:

Increase or decrease the level of difficulty of the problems by changing the data.

Add more data to make the program as long as you wish.
Change the scoring any way you wish. Can you think of a way to use a point system rather than the number correct out of the number asked?
Change the operation to addition, division, or subtraction.

RIDDLE FRACTIONS



This riddle game is fun and tests your ability to understand fractions. The computer asks you to figure out clues to the riddle using your knowledge of fractions. Then you have to use your imagination to answer the riddle! For example:

MIDDLE ½ OF MATE = AT
FIRST ½ OF RUNNER = RUN

If you cannot answer the riddle, key in anything and the program will give you a hint. You must answer the question correctly with the hint. The computer will ask *forever* using the hint until you finally get the riddle—just like your friends do when they tell you a riddle!

```
10 READ A$
```

```
20 IF A$ = "END" THEN
   STOP
30 READ B$: READ C$:
   READ D$: READ E$:
   READ F$
35 READ G$: READ H$:
   READ I$: READ J$: READ
   K$
37 IF J$ = " NO MORE
   CLUES " THEN GOTO
   100
40 READ L$: READ M$:
   READ N$
100 PRINT "♥R↑-22
   spaces-■←"
110 PRINT
120 PRINT "SOLVE THIS RID
   DLE"
130 PRINT
200 PRINT A$: PRINT B$:
   PRINT C$: PRINT D$:
   PRINT E$ PRINT F$:
   PRINT G$:
205 PRINT H$: PRINT I$:
   PRINT J$
210 PRINT
220 PRINT "R↑ -22 spaces
   ■←"
230 PRINT
235 INPUT X$
240 IF X$ = K$ THEN GOTO
   400
245 IF J$ = "NO MORE CLU
   ES" AND X$ = G$ THEN
   GOTO 350
```

-Defines the "stop program" condition.

-Lines 10, 30 and 40 read all string data.

-Provides a green stripe across the top of the screen.

-Prints the riddle "code."

-Provides a green stripe along the bottom of the riddle. The keypresses are CTRL 9, CTRL 6, CTRL 0, and CTRL 7 respectively.

-Asks your input, or answer to the riddle.

-Defines the correct answer condition (your input = riddle answer.)

```
250 PRINT "■HERE IS A
    HINT—TRY AGAIN"
260 PRINT
265 IF J$ = "NO MORE
    CLUES" THEN PRINT H$:
    PRINT I$
270 PRINT L$: PRINT M$:
    PRINT N$
280 INPUT Y$
290 IF Y$ = K$ THEN GOTO
    400
300 GOTO 250

350 PRINT "■YOU ARE
    RIGHT, THE ANSWER
    TO THE RIDDLE WAS "
    G$
375 GOTO 420
400 PRINT "■YOU ARE
    RIGHT, THE ANSWER
    TO THE RIDDLE WAS "
    K$
410 FOR T = 1 TO 2000:
    NEXT T
420 GOTO 10
1000 DATA FIRST ½ OF
    WHATEVER, MIDDLE
    ⅗ OF CHASE, FIRST ½
    OF FIVEFOLD
1020 DATA LAST ½ OF
    BUCKEYES, MIDDLE ⅗
    OF HANDY, LAST ⅝
    OF ARRESTS
1030 DATA LAST ⅓ OF
    BEACON
1040 DATA MIDDLE ⅑ OF
    CRABAPPLE, FIRST ⅝
    OF WATERFALL, LAST
    ⅜ OF GRABBED
```

-Provides a hint when a wrong answer is given.

-Prints the coded hint.

-Asks for your riddle answer.

-Defines the correct answer condition.

-Tells the program to go back to the hint since you missed it again.

-The correct answer is acknowledged.

-Lines 1000 to 1130 are the riddles in the Data statements.

```
1050 DATA THE MISSISSIPPI
     RIVER, MIDDLE 3/5 OF
     OTHER, FIRST 11/13 OF
     MISSISSIPPIAN
1060 DATA LAST 5/6 OF
     DRIVER
1100 DATA FIRST 1/2 OF
     WHATEVER, LAST 1/2
     OF BABYFOOD, MID-
     DLE 1/3 OF UNDONE
1110 DATA FIRST 5/7 OF
     BRAVERY, LAST 3/6 OF
     GENTLEPEOPLE
1120 DATA MIDDLE 1/2 OF
     CREATOR , , , ,
1125 DATA HERO SUB
1130 DATA FIRST 4/7 OF
     HEROINE, LAST 3/5 OF
     GOSUB,
1140 DATA NO MORE
     CLUES, THIS IS ALL,
     END
```

Suggested Program Changes and "Dressing"

Add color, sound, and more elaborate graphics. If you don't like the green riddle box we have provided in the program—change it!

This program is persistent about asking for the answer. It won't stop trying. If you would rather have the riddle answer appear after failing to answer correctly using the hint, here is how to do it:
Change line 300 to:

```
300 PRINT "WRONG AGAIN. THE ANSWER IS" K$
```

Add line 310:

```
310 GOTO 10
```

We recommend that you use your own riddles in the data statements. Turning the riddles into the "fraction code" is half the fun. The other half is trying them out on your friends. These are corny, but maybe they will bring to mind your favorites.

What kind of bridge makes people most anxious?
ANSWER: A suspension bridge.
What has teeth but cannot bite or chew?
ANSWER: A comb.
What does a pet canary say on Halloween?
ANSWER: Twick or tweet.
What do you get when you cross a centipede with a parrot?
ANSWER: A walkie-talkie.

Remember that in Data statements you are allowed only 88 characters (or four lines on the screen) for each statement.

PAINTER'S PALETTE

Welcome to art class at VIC school. We think you will enjoy creating your own pictures and graphic designs in the screen using all eight VIC colors and a brush.

The brush movement is controlled by the keyboard:

A = UP	< = LEFT
Z = DOWN	> = RIGHT

The brush color is changed with the color keys at the top of the keyboard.

If the brush touches the picture frame, it is returned to the screen center in white. Note that turning the brush to white allows you to erase.

```
10 PRINT "■"
20 PRINT "THIS PROGRAM
   TURNS YOUR SCREEN
   INTO A VIC PALETTE"
30 PRINT "YOUR BRUSH IS
   THE BLACK BALL
   WHICH WILL APPEAR
   IN THE CENTER OF THE
   SCREEN"
40 PRINT "YOU CONTROL
   THE BRUSH BY USING
   THESE KEYS:"
50 PRINT " . . . A = UP"
60 PRINT " . . .
   Z = DOWN"
70 PRINT " . . . < = LEFT"
80 PRINT " . . .
   > = RIGHT"
85 FOR T = 1 TO 3000:
   NEXT T
90 PRINT "■ YOU
   CONTROL THE COLOR
   OF THE BRUSH WITH
   THE VIC COLOR KEYS"
95 PRINT "YOU CAN
   CHANGE COLOR AT
   ANY TIME"
100 PRINT "WARNING:"
105 PRINT "IF YOU TRY TO
   PAINT ON THE PICTURE
   FRAME—"
107 PRINT "A WHITE BRUSH
   IS RETURNED TO THE
   CENTER OF THE
   SCREEN"
110 GOSUB 500
120 A = 7955
130 B = 38675
```

–Lines 20 to 100 are the directions for painting.

–Lines 120 and 130 let A and B be defined as screen character and

color code locations (near the center of the screen).

```
140 K = 0
```
-Sets K (the color code for the memory map).

```
150 X = PEEK (197)
```
-Asks a memory location in the computer if a key is being pressed.

```
160 IF X = 17 THEN D = -22
```
-Lines 160 to 220 tell the computer if a key is being pressed, and then assigns values according to D and K (D being distance, K being the color code). We include a table in the appendix of all the codes in memory box 197. Example: 17 = A; 33 = Z; 64 = no key pressed (see line 180); 0 = 1 (or BLK)

```
165 IF X = 33 THEN D = 22
170 IF X = 29 THEN D = -1
175 IF X = 37 THEN D = 1
180 IF X = 64 THEN D = 0
185 IF X = 0 THEN K = 0
190 IF X = 56 THEN K = 1
195 IF X = 1 THEN K = 2
200 IF X = 57 THEN K = 3
205 IF X = 2 THEN K = 4
210 IF X = 58 THEN K = 5
215 IF X = 3 THEN K = 6
220 IF X = 59 THEN K = 7
230 A = A + D
```
-States that A = A (7955 to start with) + D (depending upon what key is pressed).

```
235 B = B + D
```
-As line 230, but for variable B.

```
240 IF PEEK (A) = 160 THEN
    A = 7955: B = 38675:
    K = 1
```
-If the paint brush character location touches a PEEK value of 160 (the picture frame), then the brush is returned to the center of the screen in white (K=1).

```
260 POKE A,81: POKE B,K
```
-The brush is moved according to

```
                                    what keys are pressed (and thus
                                    what distance to move and color
                                    to become).
270 FOR W = 1 TO 50:                -Slows the movement of the
    NEXT                            brush by using this timer.
280 GOTO 150                        -Asks if another key is being
                                    pressed.
500 FOR P = 7680 TO 7701:           -Lines 500 to 540 are from a sub-
    POKEP,160:POKEP + 30720,        routine in the VIC graphics
    0: NEXT                         chapter—which gives us a solid
                                    black picture frame.
510 FOR P = 7680 TO 8164
    STEP 22: POKE P,160:
    POKE P +
    30720,0
515 FOR P = 7680 TO 8164
    STEP 22: POKE P + 21,
    160: POKE P + 30741,0:
    NEXT
520 FOR P = 8164 TO 8185:
    POKE P,160: POKE
    P + 30720,0: NEXT
540 RETURN
```

Suggested Program Changes and "Dressing"

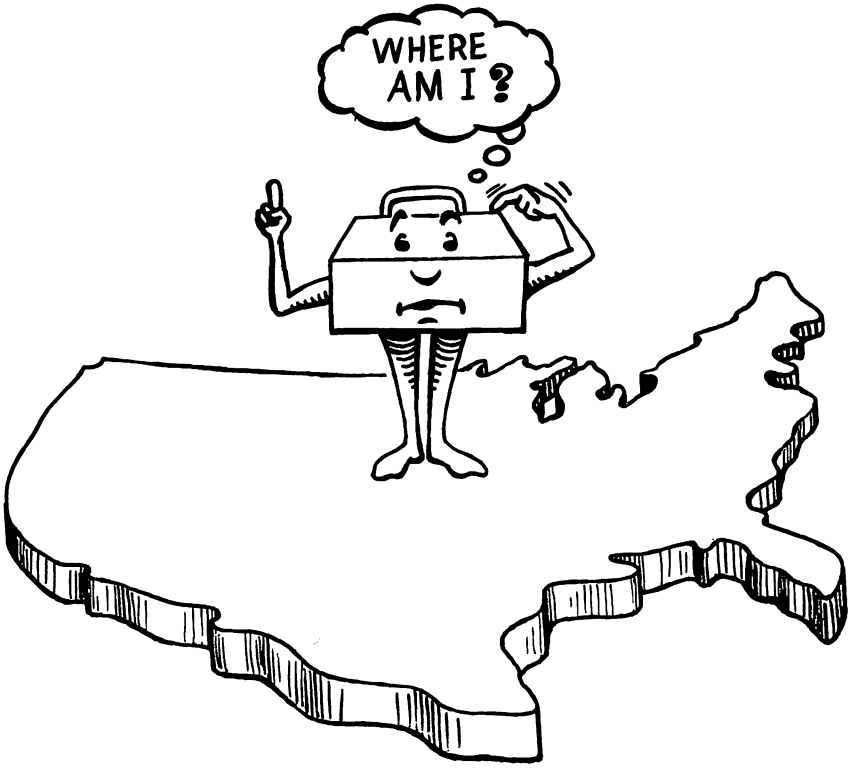
This program obviously uses graphics and color extensively. Some ideas for changes include:

1. Make the picture frame a different color or use a different screen character. See the POKE screen code table in the appendix. Try 127—■—for a checkerboard effect.
2. Make the paint brush different using different characters:

Solid block	-160
Diamond	- 90
Spade	- 65

3. Slow down or speed up the movement of the brush by changing (or deleting) line 270.

STATES AND CAPITALS



This is our centerpiece educational game because it provides such a useful program structure for states and capitals as well as many other educational games.

You are asked capitals of all 50 states and then scored and evaluated at the end of the game. Color and sound both greet you in this game, in the scoring subroutines as well as when you enter an incorrect or correct response.

The structure of the overall program is similar to Arithmetic Tester and Multiplication Tables. It uses a powerful BASIC tool—arrays. With the DIM statement, in this program, you are allowed to dimension the data in two ways, one way for states, one for capitals. Importantly for these educational quizzes, you can have the questions asked randomly from the data.

```
 5 Q = 0: N = 0
10 DIM A$ (50), B$ (50)
15 RESTORE
20 FOR Z = 1 TO 50: READ
   A$ (Z), B$ (Z): NEXT Z
30 Z = INT (RND (1) * 50) + 1
40 IF Q = 20 THEN GOTO
   200
50 Q = Q + 1
60 PRINT "■WHAT IS THE
   CAPITAL OF" A$ (Z)
70 PRINT
90 INPUT X$
100 PRINT
110 IF X$ = B$ (Z) THEN
   GOSUB 1500
120 IF X$ <> B$ (Z) THEN
   GOSUB 2000
200 GOTO 210
210 PRINT N "QUESTIONS
   RIGHT OUT OF A
   TOTAL OF" Q
220 J = N/Q
230 IF J >= .9 THEN
   GOSUB 3000
240 IF J < .9 AND J >= .7
   THEN GOSUB 4000
250 IF J < .7 THEN GOSUB
   5000
500 DATA ALABAMA, MO
   NTGOMERY, NEW YO
   RK, ALBANY, NEVADA,
```

-Sets the number of questions asked (Q) and the number correct (N) at 0.

-Dimensions an array of two string variables, 50 in number or quantity.

-Read the data in the array as expressed in the subscript (Z) from 1 to 50.

-Z is a random integer from 1 to 50.

-If there are 20 questions asked, go to the scoring lines and subroutines.

-If the answer is correct, the program goes to subroutine 1500.

-If the answer is incorrect, the program goes to subroutine 2000.

-Lines 500 to 1400 are the data statements for the array. The format is the same as for READ-

```
      CARSON CITY, OHIO,
      COLUMBUS
600   DATA IOWA, DES
      MOINES, SOUTH CAR
      OLINA, COLUMBIA,
      OREGON, SALEM, TEX
      AS, AUSTIN
700   DATA KENTUCKY, FRA
      NKFORT, NEW JERSEY,
      TRENTON, MASSACH
      USETTS, BOSTON, ARI
      ZONA, PHOENIX
800   DATA TENNESSEE,
      NASHVILLE, WYOMIN
      G, CHEYENNE, MINNE
      SOTA, ST. PAUL, MARY
      LAND, ANNAPOLIS
900   DATA NEW MEXICO,
      SANTA FE, NORTH CA
      ROLINA, RALEIGH, CO
      NNECTICUT, HART
      FORD
950   DATA ILLINOIS, SPRIN
      GFIELD
1000  DATA MAINE, AUGUS
      TA, MICHIGAN, LANSI
      NG, GEORGIA, ATLAN
      TA, ALASKA, JUNEAU,
      MONTANA, HELENA
1100  DATA ARKANSAS, LIT
      TLE ROCK, CALIFORN
      IA, SACRAMENTO, CO
      LORADO, DENVER, DE
      LAWARE, DOVER
1150  DATA FLORIDA, TALLA
      HASSEE, HAWAII, HO
      NOLULU, IDAHO, BOI
      SE, INDIANA, INDIAN
      APOLIS
1200  DATA KANSAS, TOPEK
```

DATA statements.

```
      A, LOUISIANA, BAT
      ON ROUGE, MISSISSIP
      PI, JACKSON
1250  DATA UTAH, SALT LAK
      E CITY, NEBRASKA, LIN
      COLN, NEW HAMPSHI
      RE, CONCORD, NORT
      H DAKOTA, BISMARCK
1300  DATA OKLAHOMA, O
      KLAHOMA CITY, PENN
      SYLVANIA, HARRISBU
      RG, RHODE ISLAND,
      PROVIDENCE
1350  DATA SOUTH
      DAKOTA, PIERRE,
      VERMONT, MONTPE
      LIER, VIRGINIA, RICH
      MOND, WASHING
      TON, OLYMPIA
1400  DATA WEST VIRGINIA,
      CHARLESTON, WIS
      CONSIN, MADISON,
      MISSOURI, JEFFERSON
      CITY
1500  PRINT "CORRECT"
1510  POKE 36878,8
1520  FOR L = 200 TO 225
      STEP 2
1530  POKE 36876,L
1540  FOR M = 1 TO 100
1550  NEXT M: NEXT L
1560  POKE 36878,0: POKE
      36876,0
1570  N = N + 1
1590  GOTO 15
1600  RETURN
2000  PRINT "WRONG. THE
      ANSWER IS" B$(Z)
```

–The 1500 subroutine plays a nice scale for the correct answer.

–The 2000 subroutine plays a low descending scale for a wrong answer.

```
2010 POKE 36878,8
2020 FOR L = 140 TO 128
     STEP -2
2030 POKE 36874,L
2040 FOR M = 1 TO 100
2050 NEXT M: NEXT L
2060 POKE 36878,0: POKE
     36874,0
2090 GOTO 15
2100 RETURN
3000 POKE 36879,28
```

-The 3000 subroutine awards you a purple color border and a "very good" message.

```
3010 PRINT "VERY GOOD,
     YOU REALLY KNOW
     YOUR STATES AND
     CAPITALS"
3020 END
3100 RETURN
4000 POKE 36879,29
```

-The 4000 subroutine prints a green screen border and an OK message appears for this satisfactory level of competence.

```
4010 PRINT "OK-BUT YOU
     COULD USE MORE
     PRACTICE"
4020 END
4100 RETURN
5000 POKE 36879,8
```

-The 5000 subroutine turns the screen all black and then returns to the original colors with a message suggesting more study of states and capitals.

```
5010 FOR T = 1 TO 5000:
     NEXT T
5020 POKE 36879,27
5030 PRINT "YOU NEED TO
     STUDY STATES AND
     CAPITALS"
5090 END
```

```
5100 RETURN
6000 PRINT "■"
6005 FOR P = 7680 TO 8185
     STEP 1: POKE P, 209:
     POKE P + 30720,7:
     NEXT
6010 PRINT " WELCOME
     TO"
6020 PRINT " STATES AND
     CAPITALS"
6030 FOR T = 1 TO 2000 :
     NEXT
6090 RETURN
```

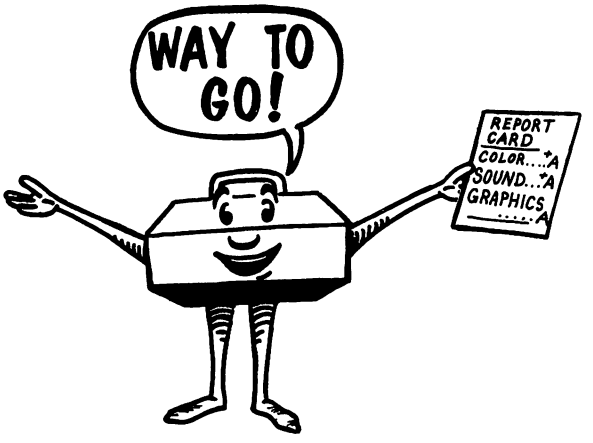
Suggested Program Changes and "Dressing"

We have included color and sound subroutines in this game. You should improve them for yourself. There are a number of ways to adapt this game: 1) change the data for the states and capitals. Use more data or less data; 2) change the scoring levels; 3) add a title opener to the program. A map of the U.S. would be appropriate, but it will take some serious POKEing about.

This is the last educational game we use as an illustration. We hope you have lots of ideas for games and quizzes to use now. Some ideas we would like to suggest include:

Countries and capitals
Continents and countries
Inventors and objects
Historical events and dates
Mythological characters
Biblical names
Diseases and their meaning
Music and musicians
Chemical elements and symbols
Countries and leaders
Words and their opposites
Numbers and their spelling
Roman numerals and numbers
Works of art and artists

In the next chapter on traditional games, we expand our knowledge of BASIC programming while continuing to have fun.

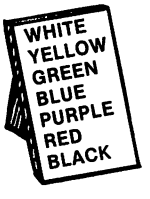

WAY TO GO!
REPORT CARD
COLOR....A
SOUND...A
GRAPHICS.....A

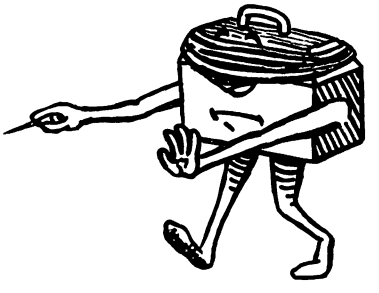
Traditional Games 9

For our purposes, Traditional Games are games that have been played for years using dice, cards, paper and pencil, or gameboards. In this chapter we will look at four such games and change them so that they can be played with your VIC 20. Using these games as models you should be able to create or adapt your own traditional games.

GUESSING GAMES

One of the oldest and simplest computer games is the Guessing Game. In the Guessing Game you can guess colors, sounds, numbers, or just about anything. Regardless of what you guess, the program will be about the same. The computer picks a number, color, or whatever by using the Random Number function. You try to guess the computer's pick using the Input statement. The computer checks to see if you guessed correctly or incorrectly and lets you know the outcome.





```
 10 TRY=0
 20 PRINT "■"
 30 X=INT (RND(1) * 8) + 1
 40 PRINT "PICK A COLOR USING THE VIC COLOR KEYS"
 50 INPUT Y$
 60 TRY=TRY + 1
 70 PRINT "THAT WAS TRY NUMBER "; TRY
 80 IF X=VAL(Y$) THEN GOSUB 100
 90 IF X <> VAL(Y$) THEN GOSUB 200
100 PRINT "YOU GUESSED CORRECTLY. NOW WATCH THE
    COLOR!"
105 FOR T=1 TO 1000: NEXT T
110 IF Y$="1" THEN POKE 36879,9
```

```
115 IF Y$="2" THEN POKE 36879,25
120 IF Y$="3" THEN POKE 36879,41
125 IF Y$="4" THEN POKE 36879,57
130 IF Y$="5" THEN POKE 36879,73
135 IF Y$="6" THEN POKE 36879,89
140 IF Y$="7" THEN POKE 36879,105
145 IF Y$="8" THEN POKE 36879,121
150 FOR T=1 TO 1000: NEXT T
155 POKE 36879,27
160 PRINT "IT TOOK YOU"; TRY; "TRIES"
170 FOR T=1 TO 1000: NEXT T
180 GOTO 10
190 RETURN
200 PRINT "SORRY, TRY AGAIN"
210 FOR T=1 TO 1000: NEXT T
220 GOTO 40
230 RETURN
```

Although the game is a simple one, it includes the components of more complex and intriguing guessing games. The computer selects a random number from 1 to 8 (line 30). You select one of the VIC color keys (actually keys 1 to 8). If the value of your input, VAL(Y$), is equal to the random number generated, then the program moves to subroutine 100. If the number was not equal to the "value of Y$," then the program moves to subroutine 200.

You can change the game for other guessing games such as sound or numbers. If you want to guess a number from 1 to 100, you should change the following lines:

```
30 X=INT(RND(1) * 100) + 1
50 INPUT Y
80 IF X=Y THEN GOSUB
90 IF X>Y THEN GOSUB
95 IF X<Y THEN GOSUB
```

TIC TAC TOE

Let's try another favorite game, Tic Tac Toe. Instead of paper and pencil, you play on the screen, as shown in Fig. 9-1. Enter the program

and we'll show you how to play. This game requires two players.

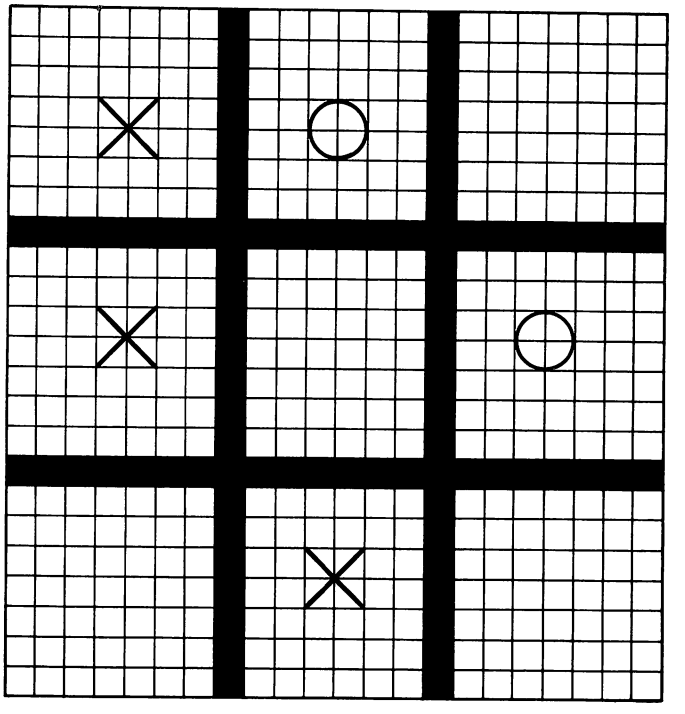


Fig. 9-1. Tic tac toe.

```
10 PRINT "■"
20 FOR Z=7944 TO 8185:
   POKE Z, 219: POKE
   Z + 30720,4: NEXT
30 PRINT: PRINT: PRINT: PR
   INT "PLAY TIC TAC TOE
   ON THE KEYBOARD"
40 PRINT: PRINT "TO STOP
   THE GAME PRESS THE
   SPACE BAR"
50 FOR T=1 TO 3500: NEXT
90 PRINT "■"
```

–20–50 Set up the instructions and introductory frame.

```
100 S=7680
110 C=38400
120 FOR G=7834 TO 7855:
    POKE G,160: POKE
    G + 30720,0: NEXT
130 FOR G=8010 TO 8031:
    POKE G,160: POKE
    G + 30720,0: NEXT
140 FOR G=7687 TO 8171
    STEP 22: POKE G,160:
    POKE G +
    30720,0: NEXT
150 FOR G=7694 TO 8178
    STEP 22: POKE G,160:
    POKE G +
    30720,0: NEXT
160 POKE S,1: POKE C,5:
    POKE S+1,61: POKE
    C+1,5: POKE S+2,86:
    POKE C+2,5
165 POKE S + 8,3: POKE
    C + 8,5: POKE S + 9,61:
    POKE C + 9,5: POKE
    S + 10,86: POKE
    C + 10,5
170 POKE S + 15,5: POKE
    C + 15,5: POKE
    S + 16,61: POKE C +
    16,5: POKE S + 17,86:
    POKE C +17,5
172 POKE S + 22,2: POKE
    C + 22,5: POKE
    S + 23,61: POKE C +
    23,5: POKE S +24,87:
    POKE C + 24,5
174 POKE S +30,4: POKE
    C + 30.5: POKE
    S + 31,61: POKE C +
    31,5: POKE S + 32,87:
    POKE C + 32,5
```

–120–150 Set up the Tic Tac Toe "grid."

–160–197 Tell you for each of the boxes what letters represent an "X" and an "O."

```
176 POKE S + 37,6: POKE
    C + 37,5: POKE
    S + 38,61: POKE C +
    38,5: POKE S + 39,87:
    POKE C + 39,5
178 POKE S + 176,7: POKE
    C + 176,5: POKE S
    +177,61: POKE
    C + 177,5: POKE S
    +178,86: POKE
    C + 178,5
180 POKE S +184,9: POKE
    C + 184,5: POKE
    S + 185,61: POKE
    C + 185,5: POKE
    S + 186,86: POKE
    C + 186,5
182 POKE S + 191,11: POKE
    C + 191,5: POKE
    S + 192,61: POKE
    C + 192,5: POKE S
    +193,86: POKE
    C + 193,5
184 POKE S + 198,8: POKE
    C + 198,5: POKE
    S + 199,61: POKE C +
    199,5: POKE S + 200,87:
    POKE C + 200,5
186 POKE S + 206,10: POKE
    C + 206,5: POKE
    S + 207,61: POKE C +
    207,5: POKE S + 208,87:
    POKE C + 208,5
188 POKE S + 213,12: POKE
    C + 213,5: POKE
    S + 214,61: POKE C +
    214,5: POKE S + 215,87:
    POKE C + 215,5
190 POKE S + 352,13: POKE
    C + 352,5: POKE
```

```
    S + 353,61: POKE C +
    353,5: POKE S + 354,86:
    POKE C + 354,5
192 POKE S + 360,15: POKE
    C + 360,5: POKE
    S + 361,61: POKE C +
    361,5: POKE S + 362,86:
    POKE C + 362,5
194 POKE S + 367,17: POKE
    C + 367,5: POKE
    S + 368,61: POKE C +
    368,5: POKE S + 369,86:
    POKE C + 369,5
195 POKE S + 374,14: POKE
    C + 374,5: POKE
    S + 375,61: POKE C +
    375,5: POKE S + 376,87:
    POKE C + 376,5
196 POKE S + 382,16: POKE
    C + 382,5: POKE
    S + 383,61: POKE C +
    383,5: POKE S + 384,87:
    POKE C + 384,5
197 POKE S + 389,18: POKE
    C + 389,5: POKE
    S + 390,61: POKE C +
    390,5: POKE S + 391,87:
    POKE C + 391,5
200 A$=CHR$(65):
    B$=CHR$(66):
    C$=CHR$(67):
    D$=CHR(68):
    E$=CHR$(69):
    F$=CHR$(70):
    G$=CHR$(71)
210 H$=CHR$(72):
    I$=CHR$(73):
    J$=CHR$(74):
    K$=CHR$(75):
    L$=CHR$(76):
```

-200-220 Define the input you can make in the keyboard.

```
     M$=CHR$(77):
     N$=CHR$(78)
220  O$=CHR$(79):
     P$=CHR$(80):
     Q$=CHR$(81):
     R$=CHR$(82)
230  S$=CHR$(32)
250   GET X$: IF X$="  "
      THEN GOTO 250
260  IF X$=A$ THEN POKE
     S + 91,86: POKE
     C + 91,6
270  IF X$=B$ THEN POKE
     S + 21,15: POKE
     C + 91,6
280  IF X$=C$ THEN POKE
     S + 99,86: POKE
     C + 99,6
290  IF X$=D$ THEN POKE
     S + 99,15: POKE
     C + 99,6
300  IF X$=E$ THEN POKE
     S + 106,86: POKE
     C + 106,6
310  IF X$=F$ THEN POKE
     S + 106,15: POKE
     C + 106,6
320  IF X$=G$ THEN POKE
     S + 267,86: POKE
     C + 267,6
330  IF X$=H$ THEN POKE
     S + 267,15: POKE
     C + 267,6
340  IF X$=I$ THEN POKE
     S + 275,86: POKE
     C + 275,6
350  IF X$=J$ THEN POKE
     S + 275,15: POKE
     C + 275,6
360  IF X$=D$ THEN POKE
```

-Defines the space bar as string variable, S$.

-260–430 Plots either an "X" or an "O" in one of the nine Tic Tac Toe blocks.

```
      S + 282,86: POKE C +
      282,6
370 IF X$=L$ THEN POKE S
      + 282,15: POKE C +
      282,6
380 IF X$=M$ THEN POKE S
      + 443,86: POKE C +
      443,6
390 IF X$=N$ THEN POKE S
      + 443,15: POKE C +
      443,6
400 IF X$=O$ THEN POKE S
      + 451,86: POKE C +
      451,6
410 IF X$=P$ THEN POKE S
      + 451,15: POKE C +
      451,6
420 IF X$=Q$ THEN POKE S
      + 458,86: POKE C +
      458,6
430 IF X$=R$ THEN POKE S
      + 458,15: POKE C +
      458,6
450 IF X$=S$ THEN GOTO
      10                        -Tells the program to start over
                                 with a clean slate.
500 GOTO 250.
```

The game could be simplified by reducing the grid size or including the "X" or "O" instructions in the PRINT command, but it would be more difficult to play.

TED AND DAVE'S CASINO

How about a little gambling? At your command, the VIC rolls the dice. You begin play with $10. If you roll a 7 or 11, you win $2. If you roll less than a 4 or if you roll a 12, you lose $2. If any other number comes up, the house takes $1 for the roll. (See Fig. 9-2.)

```
 5 S=7680: C=38400
 8 M=M + 10
10 PRINT "♡"
```

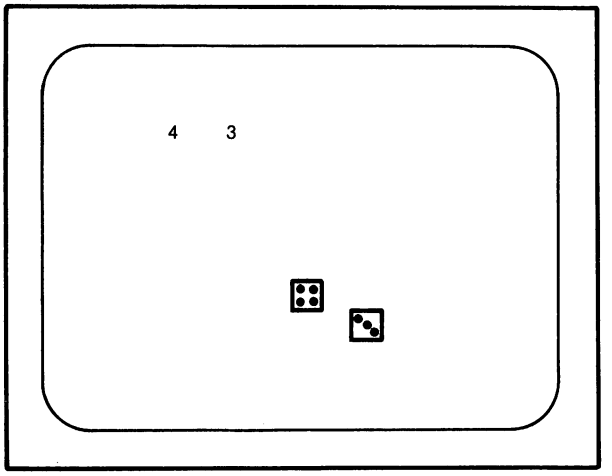


Fig. 9-2. Ted and Dave's casino.

```
15 PRINT "WELCOME TO
   TED AND DAVE'S
   CASINO"
20 PRINT: PRINT: PRINT
   "YOU HAVE 10 DOLLARS
   TO PLAY WITH—TRY
   YOUR LUCK"
30 PRINT "ROLL 7 OR 11
   AND YOU WIN $2"
40 PRINT "ROLL LESS THAN
   4 OR ROLL A 12 AND
   YOU LOSE $2"
50 PRINT "ANY OTHER
   ROLL AND YOU LOSE
   $1"
60 FOR T=1 TO 4000: NEXT
70 FOR Z=7680 TO 8185:
   POKE Z,193: POKE
   Z + 30720,5: NEXT Z
90 PRINT "■"
100 INPUT "PRESS RETURN
```

```
    TO ROLL THE DICE!";
    X$
110 POKE S + 360,85:
    POKE C + 360,0: POKE
    S + 361,64: POKE C +
    361,0: POKE
    S + 362,64: POKE
    C + 362,0
115 POKE S + 363,64:
    POKE C + 363,0: POKE
    S + 364,73: POKE C +
    364,0: POKE
    S + 382,66: POKE
    C + 382,0
120 POKE S + 386,66:
    POKE C + 386,0: POKE
    S + 404,66: POKE C +
    404,0: POKE
    S + 408,66: POKE
    C + 408,0
125 POKE S + 426,66:
    POKE C + 426,0: POKE
    S + 430,66: POKE C +
    430,0: POKE
    S + 448,74: POKE
    C + 448,0
130 POKE S + 449,64:
    POKE C + 449,0: POKE
    S + 450,64: POKE C +
    450,0: POKE
    S + 451,64: POKE
    C + 451,0
135 POKE S + 452,75:
    POKE C + 452,0: POKE
    S + 322,85: POKE C +
    322,0: POKE
    S + 323,64: POKE
    C + 323,0
140 POKE S + 324,64:
    POKE C + 324,0: POKE
```

-110-160 Plot the outside of the dice.

```
    S + 325,64: POKE C +
    325,0: POKE
    S + 326,73: POKE
    C + 326,0
145 POKE S + 344,66:
    POKE C + 344,0: POKE
    S + 348,66: POKE C +
    348,0: POKE
    S + 366,66: POKE
    C + 366,0
150 POKE S + 370,66:
    POKE C + 370,0: POKE
    S + 388,66: POKE C +
    388,0: POKE
    S + 392,66: POKE
    C + 392,0
155 POKE S + 410,74:
    POKE C + 410,0: POKE
    S + 411,64: POKE C +
    411,0: POKE
    S + 412,64: POKE
    C + 412,0
160 POKE S + 413,64:
    POKE C + 413,0: POKE
    S + 414,75: POKE C +
    414,0
200 LD=INT(RND(1) * 6) + 1
```

-Defines the left die (LD) as an integer from 1 to 6.

```
210 IF LD=1 THEN GOSUB
    450
220 IF LD=2 THEN GOSUB
    500
230 IF LD=3 THEN GOSUB
    550
240 IF LD=4 THEN GOSUB
    600
250 IF LD=5 THEN GOSUB
    650
260 IF LD=6 THEN GOSUB
    700
```

```
300 RD=INT(RND(1) * 6)
    + 1
310 IF RD=1 THEN GOSUB
    750
320 IF RD=2 THEN GOSUB
    800
330 IF RD=3 THEN GOSUB
    850
340 IF RD=4 THEN GOSUB
    900
350 IF RD=5 THEN GOSUB
    950
360 IF RD=6 THEN GOSUB
    1000
370 PRINT LD, RD
375 IF LD + RD=7 OR LD +
    RD=11 THEN M=M + 2:
    POKE 36878,15: POKE
    36876,220
376 FOR T=1 TO 200: NEXT:
    POKE 36878,0: POKE
    36876,0
377 IF LD + RD<4 THEN
    M=M-2
379 IF LD + RD=12 THEN
    M=M-2
380 IF LD + RD=4 OR LD +
    RD=5 OR LD + RD=6
    OR LD +  RD=8 OR LD
    + RD=9 OR LD + RD=
    10 THEN M=M-1
390 PRINT: PRINT "YOU
    NOW HAVE "M"DOL
    LARS"
395 IF M<1 THEN GOSUB
    3000
400 IF M>20 THEN GOSUB
    4000
```

–Defines the right die (RD) as an integer from 1 to 6.

375–380 define the payoff conditions.

```
437 FOR T=1 TO 2000:
    NEXT
449 GOTO 90
450 POKE S + 406,81:
    POKE C + 406,0

490 RETURN
500 POKE S + 383,81:
    POKE C + 383,0: POKE
    S + 429,81: POKE C +
    429,0
540 RETURN
550 GOSUB 450: GOSUB
    500
590 RETURN
600 GOSUB 500
610 POKE S + 385,81:
    POKE C + 385,0:POKE
    S + 427,81: POKE C +
    427,0
640 RETURN
650 GOSUB 600: GOSUB
    450
690 RETURN
700 GOSUB 600
710 POKE S+ 405,81: POKE
    C + 405,0: POKE S+
    407,81: POKE C +
    407,0
740 RETURN
750 POKE S+ 368,81: POKE
    C + 368,0
790 RETURN
800 POKE S+ 345,81: POKE
    C + 345,0: POKE S+
    391,81: POKE C +
    391,0
840 RETURN
850 POKE S+ 368,81: POKE
    C + 368,0: POKE S+
```

–450–1000 Plot the faces of each die depending upon the value of LD and RD.

```
      345,81: POKE C +
      345,0: POKE S+ 391,81:
      POKE C + 391,0
 890  RETURN
 900  GOSUB 800
 910  POKE S+ 347,81: POKE
      C + 347,0 : POKE S+
      389,81: POKE C +
      389,0
 940  RETURN
 950  GOSUB 900: GOSUB
      750
 990  RETURN
1000  GOSUB 900
1010  POKE S+ 367,81: POKE
      C + 367,0: POKE S+
      369,81: POKE C +
      369,0
1040  RETURN
3000  PRINT: PRINT: PRINT:
      PRINT "YOU HAVE
      NOW LOST ALL YOUR
      MONEY!"
3010  POKE 36878,15
3020  FOR Y=210 TO 128
      STEP-1
3030  POKE 36874,Y
3040  POKE 36875,Y
3050  FOR T=1 TO 5: NEXT
3060  NEXT Y
3070  POKE 36878,0: POKE
      36874,0: POKE 36875,0
3080  FOR Z=7856 TO 8185:
      POKE Z,164: POKE
      Z + 30720,4: NEXT Z
3085  END
3090  RETURN
```

-Lets you know when you have run out of money and ends the games.

```
4000 PRINT: PRINT: PRINT:
     PRINT: YOU HAVE
     WON TOO MUCH
     MONEY"
4005 PRINT "TED AND
     DAVE'S CASINO IS
     CLOSED"
4010 FOR T=1 TO 3000:
     NEXT
4020 END
4030 RETURN
```

Ends the game when you have doubled your money ($20). Some of the ways to adapt the game are to: 1) include sound each time the dice are rolled; 2) change the win/lose conditions for the game, such as *If LD=RD then M=M + 2* which gives the player two dollars for rolling doubles; 3) change the amount of money for the betting in line 8; or 4) add more color and sound for the ending of the game. When you are tired of gambling, try this friendly little game of Hangman.

HANGMAN

The VIC plays you in Hangman.(See Fig. 9-3.) Guess the secret word in the number of tries and you avoid the hanging. When you guess the word correctly, a person appears with a "THANKS!" message. If you don't guess the word correctly, he is hung.

```
10 PRINT "■"
20 PRINT "QQQ[][][][]
   WELCOME"
30 PRINT "[][][][][]TO"
40 PRINT "[][][][][]
   HANGMAN"
50 GOSUB 1000: GOSUB
   1200
60 FOR T=1 TO 3000: NEXT
70 X=INT(RND(1) * 10) +1     -Defines X as a random number
                              from 1 to 10 (10 being the num-
                              ber of words in the DATA state-
80 FOR A=1 TO X               ment).
```

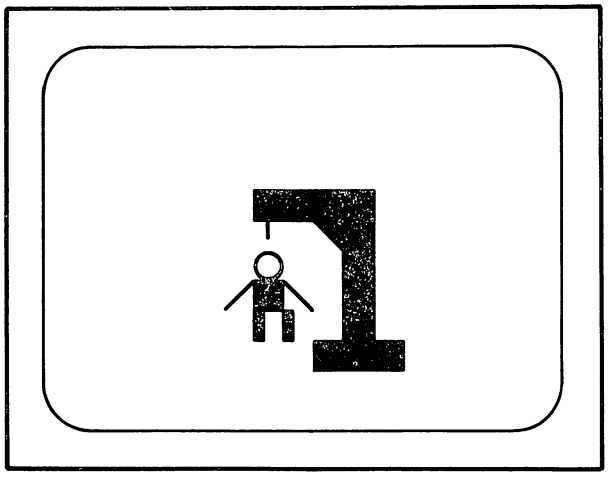


Fig. 9-3. Hangman.

```
 90 READ A$
100 NEXT A
110 N=LEN(A$)                    -Defines N as the number of
                                  characters in the word READ.
120 FOR A=1 TO N
130 L(A)=ASC(MID$(A$,A,1))       -Sets the array L(A) as the ASCII
                                  of any 1 letter in the word.
140 O(A)=L(A)
150 NEXT A
160 T=N + 4                      -Defines T as the length of the
                                  word plus 4 more.
170 PRINT "■ GUESS THIS
    WORD IN "T" TRIES"
180 FOR S=1 TO T
190 GOSUB 450
200 IF C=N THEN 340              -If the number of correct letters
                                  (C) equals the number of letters
                                  in the word (N)—the game is
                                  completed.
210 PRINT T + 1−S "TRIES         -Reduces the number of tries (T)
    TO GO!"                       for each incorrect guess.
220 INPUT X$                     -Asks you to input a letter guess.
230 X=ASC(X$)
```

```
240 FOR A=1 TO N
250 IF O(A)=X THEN
    O(A)=0: S=S-1
```

-If a correct letter is guessed (X), then the value in the array O(A) is set at 0. This allows the correct letter to be printed amid the dashes. S is reduced by one, meaning that the number of tries is *not* reduced when a correct letter is guessed.

```
260 NEXT: NEXT
270 GOSUB 450
280 FOR T=1 TO 2000: NEXT
290 PRINT "■"
```

-290-320 Set up the losing condition for Hangman.

```
300 PRINT: PRINT "YOU
    LOSE! TO THE GAL
    LOWS WITH YOU!"
310 GOSUB 1000: GOSUB
    1200
320 GOTO 600
340 PRINT "■YOU GOT IT!
    NO HANGING TODAY"
350 PRINT "THE WORD IS"
    A$
360 GOSUB 1000
370 PRINT "QQQQQQQQQQ
    Q]]]]]] THANKS!"
380 END
390 DATA "WAKE", "RID
    DLE", "TOUCH",
    "EXTRA", "JOKE",
    "PLEASANT"
400 DATA "COWARD",
    "TOOLKIT", "DELAY",
    "SIMPLEST"
```

-340-380 Set up the winning condition for Hangman.

```
450 C=0: PRINT "Q"
```

-Sets the number of correct letters at 0.

```
460 FOR Z=1 TO N
470 IF L(Z)=O(Z) THEN
    PRINT "*";
```

-Checks to see if any letters in A$ have been selected. If each

```
480 IF L(Z)<>O(Z) THEN
    PRINT CHR$(L(Z));:C=C
    + 1
490 NEXT: PRINT: PRINT
500 RETURN
600 PRINT "TOUGH LUCK"

610 PRINT "THE WORD IS"
    A$
620 PRINT "QQQQQQQQQ
    ]]]]]]]] AAAGH"
690 END
1000 REM**MAN
1005 F=7680: D=38400
1010 POKE F + 342,85: POKE
     D + 342,5: POKE F +
     343,73: POKE D +
     343,5: POKE F + 364,74:
     POKE D + 364,5
1020 POKE F + 365,75: POKE
     D + 365,5: POKE F +
     385,78: POKE D +
     385,2: POKE F +
     386,102: POKE D +
     386,2
1030 POKE F + 387,102:
     POKE D + 387,2: POKE
     F + 388,77: POKE D +
     388,2: POKE F + 406,78:
     POKE D + 406,2
1040 POKE F + 408,102:
     POKE D + 408,2: POKE
     F + 409,102: POKE D +
```

character in the array L(Z) equals O(Z) then the star is printed.

-Checks to see if any letters have been selected in A$. If a correct letter has been selected L(Z) is no longer equal to O(Z) and the program prints the correct letter in the right space. The number correct increases by one for each correct letter.

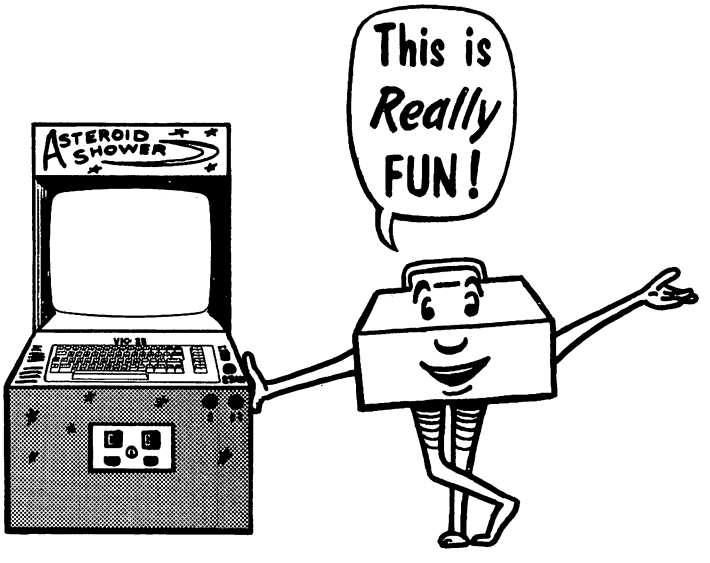
-600–690 Set up the losing condition for Hangman.

```
      409,2: POKE F + 411,77:
      POKE D + 411,2
1050  POKE F + 430,97: POKE
      D + 430,6: POKE F +
      431,118: POKE D +
      431,6: POKE F + 452,97:
      POKE D + 452,6
1060  POKE F + 453,118:
      POKE D + 453,6
1100  RETURN                    -Draws the man on the screen.
1200  POKE F + 299,160:
      POKE D + 299,0: POKE
      F + 300,
      160: POKE D + 300,0:
      POKE F + 301,160:
      POKE D + 301,0
1210  POKE F + 302,160:
      POKE D + 302,0: POKE
      F + 303,160: POKE D +
      303,0:POKE F +
      304,160: POKE D +
      304,0
1220  POKE F + 325,95: POKE
      D + 325,0: POKE F +
      326,160: POKE D +
      326,0: POKE F +
      348,160: POKE D +
      348,0
1230  POKE F + 370,160:
      POKE D + 370,0: POKE
      F + 392,160: POKE D +
      392,0: POKE F +
      414,160: POKE D +
      414,0
1240  POKE F + 436,160:
      POKE D + 436,0: POKE
      F + 458,160: POKE D +
      458,0: POKE F +
      479,160: POKE D +
      479,0
```

```
1250 POKE F + 480,160:
     POKE D + 480,0: POKE
     F + 481,160: POKE D +
     481,0: POKE F +
     321,117: POKE D +
     321,0
1300 POKE 36878,15
1310 FOR Y=200 TO 128
     STEP-1
1320 POKE 36876,Y
1330 POKE 36875,Y
1340 FOR T=1 TO 10: NEXT
1350 NEXT Y
1360 POKE 36878,0: POKE
     36876,0: POKE 36875,0
1999 RETURN
```

-Draws the gallows and POKEs a losing sound routine as well.

We hope you have enjoyed the games in this chapter. Try some of your own favorites and use our games as models. We use some of the logic for the arcade type games in our next chapter.



Arcade Games 10

Creating arcade games can be a lot of fun, but it can also be extremely frustrating if you are not careful. Begin with simple games and improve them. We wrote the games in this chapter by planning the simple game logic (car driving up screen, creature moving through maze) and then adding color, sound, scoring, and other enhancements. We hope you enjoy these arcade games.

SAUCER BLASTER

Two spacecraft appear on the screen, at the left, the escort ship, and at the right, the enemy saucer. (See Fig. 10-1.) You shoot your missile from the green "silo" by pressing function key 1 (F1).

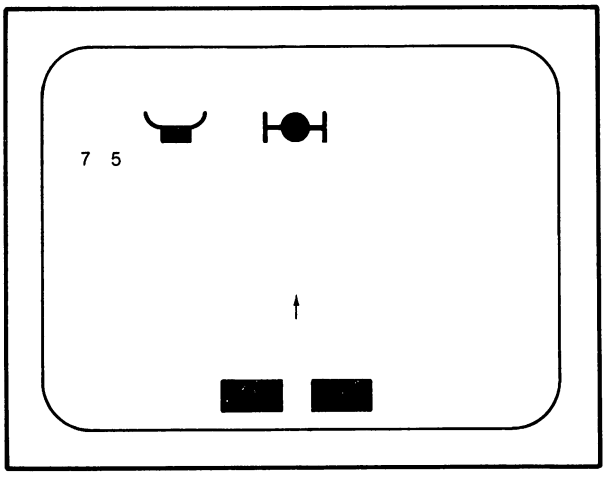


Fig. 10-1. Saucer blaster.

If you hit the enemy saucer, you earn 25 points. If you accidentally hit the friendly escort ship, you lose 15 points.

```
10 GOSUB 1000
30 PRINT"■"
40 POKE 36879,11
50 Y=0: X=0: ZX=1: C=8174:
   R=0
60 POKE 8172,160: POKE
```

```
    8173,160: POKE
    8175,160: POKE 8176,160
 65 POKE 38892,5: POKE
    38893,5: POKE 38895,5:
    POKE 38896,5
 70 S=0
 80 PRINT "QQQQ"R
 90 K=PEEK(197)
100 IF K=39 AND S=0 THEN
    S=1
110 IF S<>0 THEN GOSUB
    500
120 POKE 7711+X+22*Y,32:
    POKE 7712+X+22*Y,32:
    POKE 7713+X+22*Y,32
135 POKE 7707+X+22*Y,32:
    POKE 7708+X+22*Y,32:
    POKE 7709+X+22*Y,32
150 X=X+ZX
160 IF X=-6 OR X=10 THEN
    ZX=-ZX
180 POKE 7711+X+22*Y,107:
    POKE 7712+X+22*Y,81:
    POKE 7713+X+22*Y,115
185 POKE 7707+X+22*Y,74:
    POKE 7708+X+22*Y,98:
    POKE 7709+X+22*Y,75
200 GOTO 90
500 IF S=1 THEN POKE C,30:
    S=2: RETURN
510 POKE C,32
520 IF S =23 THEN S=0:
    C=8174: RETURN
530 C=C-22: S=S+1
540 IF PEEK(C)=107 OR
    PEEK(C)=81 OR
    PEEK(C)=115 THEN
    GOSUB 800
```

-60 & 65 POKEs silo green.

-If F1 is pressed, S is set at 1.

-Subroutine 500 is called if S is not equal to 0.

-POKEs the enemy saucer blank space.

-POKEs the friendly escort ship blank space.

-If the escort ship gets too close to the left side or the saucer too close to the right, direction is reversed.

-POKEs the saucer.

-POKEs the escort ship.

```
550 IF PEEK(C)=74 OR
    PEEK(C)=98 OR
    PEEK(C)=75 THEN
    GOSUB 900
560 POKE C,30
590 RETURN
```

–Plots the missile moving up the screen by changing the S value, alternating movement from missile to spacecraft.

```
800 REM**HIT SPACECRAFT
810 S=0: POKE C,86: POKE
    36874,200
820 FOR T=1 TO 200: NEXT
830 FOR Q=1 TO 15
835 POKE 36878,Q
840 FOR T=1 TO 50: NEXT
845 NEXT Q
850 POKE 36878,0: POKE
    36874,0
860 R=R+25
870 PRINT "□"R
880 C=8174
890 RETURN
```

–Sets S back to 0 and plays a sound for scoring 25 points.

```
900 REM**HIT ESCORT SHIP
905 POKE 36878,15: POKE
    36876,180: POKE
    36877,180
910 FOR T=1 TO 100: NEXT
915 POKE 36878,0: POKE
    36877,0: POKE 36876,0
950 FOR V=123 TO 11
    STEP-16
960 POKE 36879,V
970 FOR T=1 TO 50
970 NEXT V
980 R=R-15
990 PRINT "□"R
999 RETURN
```

–A sound is made when the escort ship is hit and the score is reduced by 15 points.

```
1000 REM***OPENING
1005 PRINT "■"
1010 FOR V=123 TO 11
     STEP-16
1020 POKE 36879,V
1030 FOR T=1 TO 90: NEXT
1040 NEXT V
1050 PRINT "■QQQQQQ
     Q]]]]]] SAUCER
     BLASTER"
1060 PRINT "Q]]]]]]PRESS
     F1 TO FIRE"
1080 FOR T=1 TO 2500:
     NEXT
1090 RETURN                    -Welcomes you to the game.
```

Saucer Blaster has all the elements of a good arcade game—graphics, animation, color, sound, and a scoring mechanism. Change some of the lines in this program to improve it. One of the best ways to test your understanding of this game's logic is to change it to a horizontal game. Have the ships move up and down the screen with the missile firing across. Or change the missile into bombs being dropped down the screen.

POTHOLE DERBY

The object of this game is to drive the car (♦) up the road avoiding the potholes and the side of the road. (See Fig. 10-2.)

You can control the car by function keys:

Press F1–the car moves left
Press F3–the car moves right

You're given three laps on this road. The score is computed like this:

+20 - for reaching the top of the road
− 5 - for hitting a pothole.
−10 - for running off the side of the road.

60 is a perfect score.

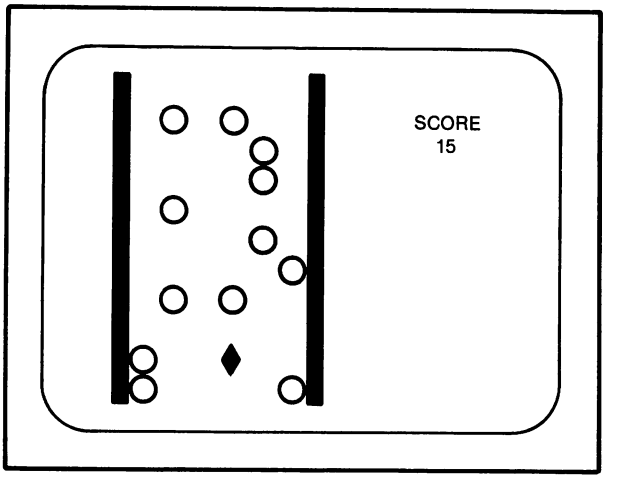


Fig. 10-2. Pothole derby.

```
 5 S=0
10 GOSUB 800
15 FOR L=1 TO 3                  -Sets up the 3 laps of the pothole
                                  derby.
20 PRINT "♥"
30 GOSUB 700: GOSUB
   500: GOSUB 600
50 A=0: F1=39: F3=47:
   F0=64
55 POKE 36878,8: POKE            -Turns the car on with an engine
   36874,128                      noise.
60 FOR B=0 TO -23 STEP
   -1
70 K=PEEK(197)
80 IF K=F1 THEN A=A-1            -If function key 1 (F1) is pressed,
                                  then the car moved to the left.
90 IF K=F3 THEN A=+1             -If function key 3 (F3) is pressed,
                                  then the car moves to the right.
120 IF PEEK(8173+A+22*B)=        -Tells if the car has hit the left-
    118 THEN GOSUB 400            hand side of the road.
130 IF PEEK(8173+A+22*B)=        -Tells if the car has hit the right-
    117 THEN GOSUB 400            hand side of the road.
```

```
135 IF PEEK(8173+A+22*B)=
    87 THEN GOSUB 300
140  IF B=-23 THEN S=S+20

180 POKE 8173+A+22*B,90

190 POKE 38893+A+22*B,2
200 FOR T=1 TO 100
210 POKE 8173+A+22*B,32
260 NEXT B
280 NEXT L
290 PRINT "♥QQQQ]]]]]
    R↑YOUR FINAL"
292 PRINT "]]]]]R↑SCO
    RE IS"
294 PRINT "]]]]]R↑"S
299 POKE 36874,0: END
300 REM**HIT POTHOLE
310 POKE 36877,200
320 FOR T=1 TO 10: NEXT
330 S=S-5
340 POKE 36877,0
350 RETURN
400 REM**HIT SIDE
410 POKE 36876,180
420 FOR T=1 TO 20: NEXT
430 POKE 36876,0
440 S=S-10
490 RETURN
500 REM**ROAD BORDER
510 FOR Z=7685 TO 8169 ST
    EP 22: POKE Z,118: POK
    E Z+30720,6
515 FOR Z=7685 TO 8169 ST
    EP 22: POKE Z+8,117: P
    OKE Z+30728,6: NEXT
550 RETURN
600 REM**POTHOLES
605 FOR P=1 TO 40
610 X=INT(RND(1)*12)
```

-Tells if a pothole is hit.

-Tells if the car has reached the top of the screen.

-180 & 190 POKE the car into position.

-Erases the car.

-290-299 End the game with a final score.

```
620 Y=INT(RND(1)*21)
630 Z=7680+X=22*Y
650 IF X>5 AND X<13 THEN
    POKE Z,87: POKE Z+
    30720,4
660 NEXT
690 RETURN
700 PRINT "[]]]]]]]]]]]]]]
    ]]]]] SCORE"
710 PRINT "[]]]]]]]]]]]]]]
    ]]]]]"S
720 RETURN
800 REM**OPENING FRAME
810 PRINT "♥QQ]]]]]R]
    WELCOME TO ■←"
820 PRINT "]]]]]R] POT
    HOLE DERBY■←"
830 FOR T=1 TO 2000: NEXT
840 PRINT "QQ]]]]] PRESS
    F1 TO MOVE LEFT"
850 PRINT "]]]]] PRESS F3
    TO MOVE RIGHT"
860 FOR T=1 TO 1000: NEXT
870 FOR Z=7680 TO 8185:
    POKE Z,214: POKE
    Z+30720,4: NEXT
899 RETURN
```

You can easily increase the difficulty of the game by changing the number of potholes in lines 610 and 620. Change line 15 if you want more laps or fewer laps.

ASTEROID SHOWER

Using the same program logic as in Pothole Derby, we can create an entirely different arcade game, Asteroid Shower (shown in Fig 10-3). You are a space ship, cleaning asteroids from outer space. Each time you remove an asteroid, you earn 10 points.

You have only the time warps to fear. If you accidentally touch

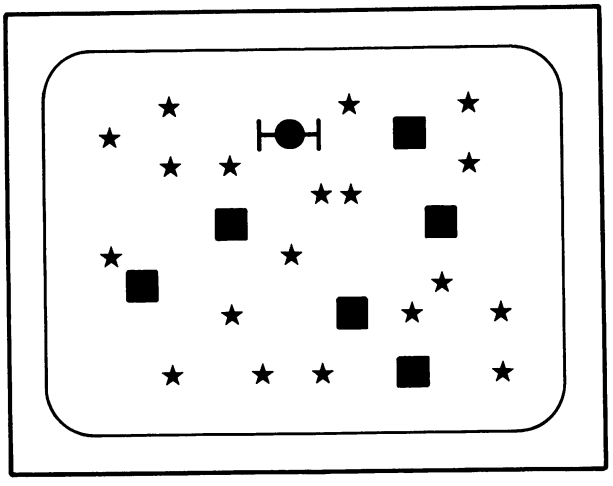


Fig. 10-3. Asteroid shower.

them, you lose 20 points. You have four turns to clean the skies of asteroids.

You control the space ship with the function keys:

F1–moves the ship to the right
F3–moves the ship to the left
F5–jumps the ship 5 spaces to the right
F7–jumps the ship 5 spaces to the left.

```
 5 PRINT "♥": POKE 36879,8
10 S=0: GOSUB 500: GOSUB
   600                          -Starts the score at 0 and the program shows you the asteroids and time warps.

12 PRINT "QQQQQQQ]]]]
   ]]]]WELCOME TO"
14 PRINT "]]]] ASTEROID
   SHOWER"
16 FOR T=1 TO 2000: NEXT        -12-16 Additional opening remarks.

20 PRINT "♥"
30 GOSUB 500: GOSUB 600         -POKEs the asteroids and time warps.

40 FOR A=1 TO 4                 -Gives the player 4 turns.
50 X=0
```

```
 60 FOR Y=0 TO 23 STEP 1
 70 F1=39: F3=47: F5=55: F7
    =63
 80 K=PEEK(197)
 90 IF K=F1 THEN X=X+1
100 IF K=F3 THEN X=X-1
110 IF K=F5 THEN X=X+5
120 IF K=F7 THEN X=X-5
130 IF PEEK(7689+X+22*Y)=
    42 OR PEEK(7690+X+22*
    Y)=42 OR PEEK(7691+X+
    22*Y)=42 THEN GOSUB
    800
140 IF PEEK(7689+X+22*Y)=
    102 OR PEEK(7690+X+
    22*Y)=102 OR PEEK
    (7691+X+22*Y)=102
    THEN GOSUB 900
170 POKE 7689+X+22*Y,107:
    POKE 7690+X+22*Y,81:
    POKE 7691+X+22*Y,115
180 GOSUB 1000
190 POKE 7689+X+22*Y,32:
    POKE 7690+X+22*Y,32:
    POKE 7691+X+22*Y,
    32
210 NEXT Y

400 NEXT A

410 PRINT "■YOUR FINAL
    SCORE WAS" S
420 END
500 REM**ASTEROIDS
510 FOR Q=1 TO 50
515 V=7746+INT(RND(1)*
    480)
520 POKE V,42: POKE V+
    30720,2
```

-Directs movement down the screen.

-90-120 move the space ship.

-Looks to see if an asteroid is hit by the space ship.

-Looks to see if a warp is hit by the space ship.

-POKEs the space ship.

-Erases the space ship.

-The space ship moves down Step 1-Y.

-Sends the program to the next turn (4 turns).

```
530 NEXT Q
590 RETURN
600 REM**TIME WARPS
610 FOR W=1 TO 10
615 V=7746+INT(RND(1)*
    480)
620 POKE V,102: POKE V+
    30720,7
630 NEXT W
690 RETURN
800 REM**TAKE ASTEROID
810 POKE 36878,15: POKE
    36876,180
820 FOR T=1 TO 15: NEXT
830 POKE 36878,0: POKE
    36876,0
840 S=S+10
850 PRINT "■"S
890 RETURN
900 REM**HIT WARP
910 POKE 36877,200
915 FOR V=15 TO 1 STEP-1
920 POKE 36878,V
930 FOR T=1 TO 20: NEXT
    T
940 NEXT V
950 S=S-20
960 PRINT "■"S
980 POKE 36878,0: POKE
    36877,0
990 RETURN
1000 FOR T=1 TO 1000:
    NEXT
1090 RETURN
```

You could easily add new ideas to Asteroid Shower. POKE a space station on the screen. If the ship lands on the space station for refueling, additional points are earned. Hide invisible black holes that take points away from the score. Use your imagination and have fun.

RAT TRAP

The green rat is trapped in a maze. The object of the game is to score points by capturing the prizes (hearts) while not hitting the maze walls. (See Fig. 10-4.) Thirty-five points are earned for capturing a prize, and 10 points are lost by hitting the wall. The rat has two chances to move through the maze.

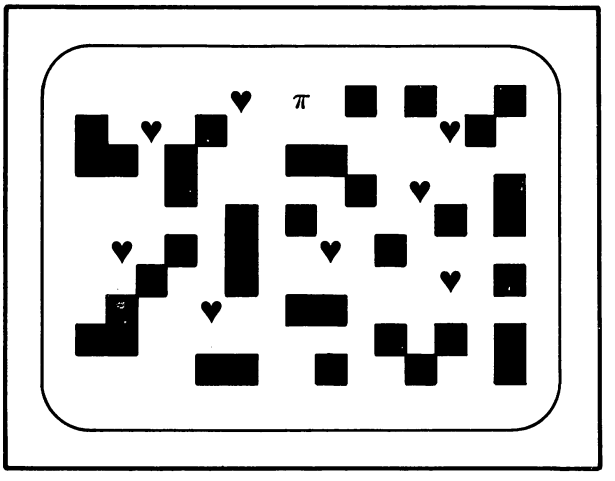


Fig. 10-4. Rat trap.

You control the rat with these keys:

R–to move up
F–to move down
<–to move left
>–to move right.

```
10 PRINT "■"
20 GOSUB 1000          -Sends the program to the opening subroutine.
25 S=0                 -Starts the score at 0.
30 PRINT "■"
40 GOSUB 500: GOSUB    -Sends the program to the subroutines that plot the maze and prizes.
   600
```

```
 50 FOR A=1 TO 2

 60 X=0
 65 FOR Y=0 TO 22 STEP 1

 70 K=PEEK(197)
 80 IF K=29 THEN X=X-1

 90 IF K=37 THEN X=X+1
100 IF K=10 THEN Y=Y-2
110 IF K=42 THEN Y=Y+1
120 IF PEEK(7685+X+22*Y)=
    160 THEN GOSUB 700
140 IF PEEK(7685+X+22*Y)
    =83 THEN GOSUB 800
160 POKE 7685+X+22*Y,
    222: POKE 7685+30720
    +X+22*Y,5
170 FOR T=1 TO 500: NEXT
180 POKE 7685+X+22*Y,32
190 NEXT Y
210 NEXT A
220 PRINT
    "♥R↑QQQQ]]]]]
    FINAL SCORE ■←"
230 PRINT"R↑]]]]]]]]]
    "S"■←"
250 END
500 REM**MAZE
510 FOR W=1 TO 120
520 V=7724+INT(RND(1)*
    480)
530 POKE V,160: POKE V+
    30720,0
540 NEXT W
590 RETURN
600 REM**PRIZES
610 FOR P=1 TO 10
620 V=7724+INT(RND(1)*
    480)
```

-Give the rat two trips through the maze.

-Gives the downward movement to the game.

-80-110 Asks if a movement control key is being pressed.

-Asks if the maze wall is hit by the rat.

-Asks if the prize is captured.

-POKEs the rat in its starting position.

-Erases the rat.

-220-250 End the game with final scores.

```
630 POKE V,83: POKE V+
    30720,6
640 NEXT P
690 RETURN
700 REM**HIT MAZE WALL
710 POKE 36878,15: POKE
    36877,150
720 FOR T=1 TO 50: NEXT
730 POKE 36878,0: POKE
    36877,0
740 S=S-10
750 PRINT "[●][R][↑]" S "[■][←]"
790 RETURN
800 REM**CAPTURE
    TREASURE
810 POKE 36878,8
820 FOR L=200 TO 225
    STEP 1
830 POKE 36876,L
840 FOR M=1 TO 50: NEXT
850 NEXT L
860 POKE 36878,0: POKE
    36878,0
870 S=S+35
880 PRINT "[●][R][↑]" S "[■][←]"
890 RETURN
1000 REM**OPENING
1010 GOSUB 500: GOSUB
     600
1020 PRINT "[Q][Q][ ][ ][ ][ ][ ][R][ ]
     RAT TRAP [■][←]"
1030 FOR T=1 TO 1500:
     NEXT
1040 PRINT "[♥][Q][Q][Q][Q][ ][ ]
     KEY R TO MOVE UP"
1050 PRINT "[Q][ ][ ] KEY F TO
     MOVE DOWN"
1060 PRINT "[Q][ ][ ] KEY < TO
     MOVE LEFT"
```

```
1070 PRINT "[Q][ ] KEY > TO
     MOVE RIGHT"
1080 FOR T=1 TO 2500:
     NEXT
1090 RETURN
```

You can customize this game further if you like. Hide a cat in the maze to chase the rat. Increase the difficulty of the game by adding more maze walls.

After working with the programs in this chapter, we hope you have learned how to successfully put all the VIC tools together to make enjoyable games. Use our subroutines and programs as examples to develop your knowledge of VIC BASIC and to construct your own educational and recreational programs.

APPENDIX A

Screen and Border Combinations

Screen	Border							
	BLK	WHT	RED	CYAN	PUR	GRN	BLU	YEL
BLACK	8	9	10	11	12	13	14	15
WHITE	24	25	26	27	28	29	30	31
RED	40	41	42	43	44	45	46	47
CYAN	56	57	58	59	60	61	62	63
PURPLE	72	73	74	75	76	77	78	79
GREEN	88	89	90	91	92	93	94	95
BLUE	104	105	106	107	108	109	110	111
YELLOW	120	121	122	123	124	125	126	127
ORANGE	136	137	138	139	140	141	142	143
LT. ORANGE	152	153	154	155	156	157	158	159
PINK	168	169	170	171	172	173	174	175
LT. CYAN	184	185	186	187	188	189	190	191
LT. PURPLE	200	201	202	203	204	205	206	207
LT. GREEN	216	217	218	219	220	221	222	223
LT. BLUE	232	233	234	235	236	237	238	239
LT. YELLOW	248	249	250	251	252	253	254	255

APPENDIX B

Musical Notes

Note	Low Octave	Middle Octave	High Octave	Top Octave
C	135	195	225	240
C#	143	199	227	241
D	147	201	228	
D#	151	203	229	
E	159	207	231	
F	163	209	232	
F#	167	212	233	
G	175	215	235	
G#	179	217	236	
A	183	219	237	
A#	187	221	238	
B	191	223	239	

APPENDIX C

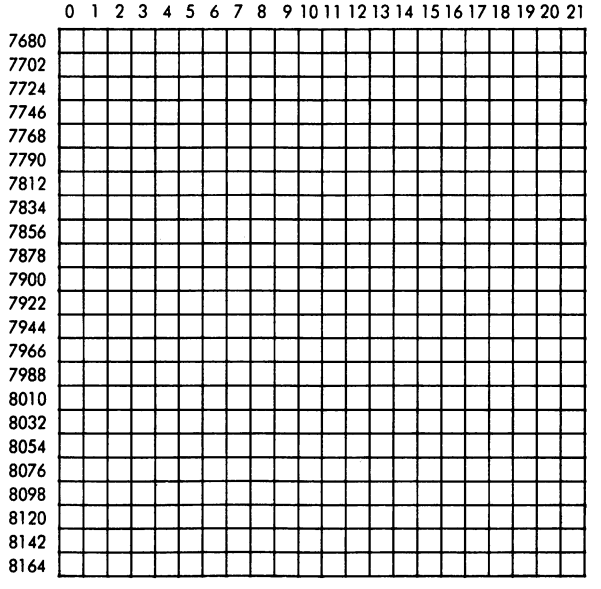
Screen Values for Poke

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	U	u	21	*		42
A	a	1	V	v	22	+		43
B	b	2	W	w	23	,		44
C	c	3	X	x	24	−		45
D	d	4	Y	y	25	.		46
E	e	5	Z	z	26	/		47
F	f	6	[		27	0		48
G	g	7	£		28	1		49
H	h	8	]		29	2		50
I	i	9	↑		30	3		51
J	j	10	←		31	4		52
K	k	11	SPACE		32	5		53
L	l	12	!		33	6		54
M	m	13	"		34	7		55
N	n	14	#		35	8		56
O	o	15	$		36	9		57
P	p	16	%		37	:		58
Q	q	17	&		38	;		59
R	r	18	'		39	<		60
S	s	19	(		40	=		61
T	t	20	)		41	>		62

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
?		63		U	85			107
		64		V	86			108
	A	65		W	87			109
	B	66		X	88			110
	C	67		Y	89			111
	D	68		Z	90			112
	E	69			91			113
	F	70			92			114
	G	71			93			115
	H	72			94			116
	I	73			95			117
	J	74	SPACE		96			118
	K	75			97			119
	L	76			98			120
	M	77			99			121
	N	78			100			122
	O	79			101			123
	P	80			102			124
	Q	81			103			125
	R	82			104			126
	S	83			105			127
	T	84			106			

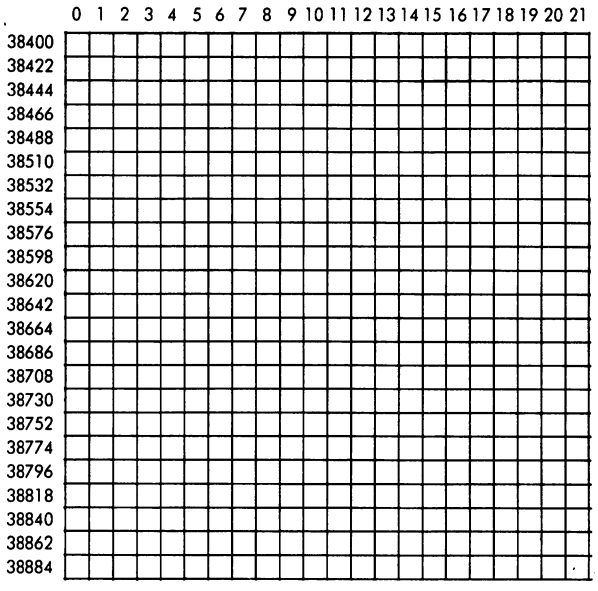
APPENDIX D

Screen Character Code Memory Map



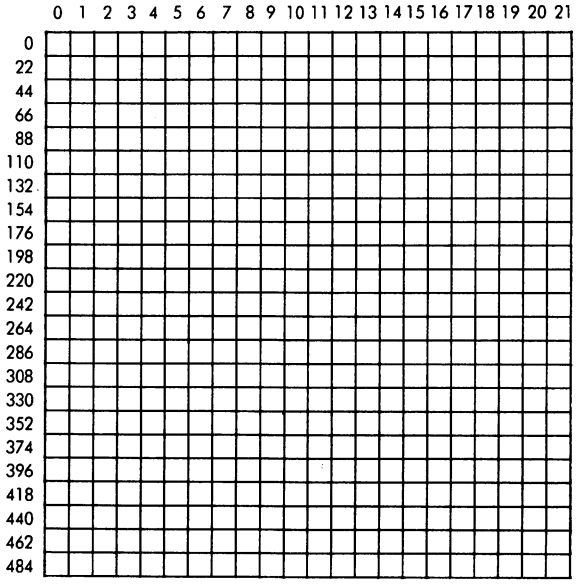
APPENDIX E

Color Code Memory Map



APPENDIX F

Memory Map Graph Paper



APPENDIX G

ASCII and CHR$ Codes

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		27	6	54
	1	RED	28	7	55
	2	CRSR →	29	8	56
	3	GRN	30	9	57
	4	BLU	31	:	58
WHT	5	SPACE	32	;	59
	6	!	33	<	60
	7	"	34	=	61
	8	#	35	>	62
	9	$	36	?	63
	10	%	37	@	64
	11	&	38	A	65
	12	'	39	B	66
RETURN	13	(	40	C	67
SWITCH TO LOWER CASE	14	)	41	D	68
	15	*	42	E	69
	16	+	43	F	70
CRSR ↓	17	,	44	G	71
RVS ON	18	–	45	H	72
CLR HOME	19	.	46	I	73
INST DEL	20	/	47	J	74
	21	0	48	K	75
	22	1	49	L	76
	23	2	50	M	77
	24	3	51	N	78
	25	4	52	O	79
	26	5	53	P	80

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
Q	81		112		143
R	82		113	BLK	144
S	83		114	↓ CRSR	145
T	84		115	RVS OFF	146
U	85		116	CLR HOME	147
V	86		117	INST DEL	148
W	87		118		149
X	88		119		150
Y	89		120		151
Z	90		121		152
[	91		122		153
£	92		123		154
]	93		124		155
↑	94		125	PUR	156
←	95		126	↑ CRSR	157
	96		127	YEL	158
	97		128	CYN	159
	98		129	SPACE	160
	99		130		161
	100		131		162
	101		132		163
	102	f1	133		164
	103	f3	134		165
	104	f5	135		166
	105	f7	136		167
	106	f2	137		168
	107	f4	138		169
	108	f6	139		170
	109	f8	140		171
	110	SHIFT RETURN	141		172
	111	SWITCH TO UPPER CASE	142		173

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	174		180		186
	175		181		187
	176		182		188
	177		183		189
	178		184		190
	179		185		191

APPENDIX H

Peek Values for the Keyboard

When you want your program to see (or PEEK) if a Key is being pressed, use these codes in your program.

←	= 8	@	= 53	↓	= 31
1	= 0	*	= 14	→	= 23
2	= 56	↑	= 54	F1	= 39
3	= 1	RETURN	= 15	F3	= 47
4	= 57	A	= 17	F5	= 55
5	= 2	S	= 41	F7	= 63
6	= 58	D	= 18	SPACE	= 32
7	= 3	F	= 42		
8	= 59	G	= 19	No key	
9	= 4	H	= 43	pressed	= 64
10	= 60	J	= 20		
+	= 5	K	= 44		
−	= 61	L	= 21		
£	= 6	:	= 45		
CLR	= 62	;	= 22		
INS	= 7	=	= 46		
Q	= 48	Z	= 33		
W	= 9	X	= 26		
E	= 49	C	= 34		
R	= 10	V	= 27		
T	= 50	B	= 35		
Y	= 11	N	= 28		
U	= 51	M	= 36		
I	= 12	,	= 29		
O	= 52	.	= 37		
P	= 13	/	= 30		

Index

TO THE READER

Sams Computer books cover Fundamentals — Programming — Interfacing — Technology written to meet the needs of computer engineers, professionals, scientists, technicians, students, educators, business owners, personal computerists and home hobbyists.

Our Tradition is to meet your needs and in so doing we invite you to tell us what your needs and interests are by completing the following:

1. I need books on the following topics:

_____

_____

_____

_____

_____

_____

2. I have the following Sams titles:

_____

_____

_____

_____

_____

_____

3. My occupation is:

_____ Scientist, Engineer
_____ Personal computerist
_____ Technician, Serviceman
_____ Educator
_____ Student
_____ D P Professional
_____ Business owner
_____ Computer store owner
_____ Home hobbyist
Other _____

_____

Name (print) _____

Address _____

City _____ State _____ Zip _____

Mail to: **Howard W. Sams & Co., Inc.**
Marketing Dept. #CBS1/80
4300 W. 62nd St., P.O. Box 7092
Indianapolis, Indiana 46206

22309

SAMS

The Tool Kit Series VIC 20™ Edition

To become computer literate and to keep your VIC 20 computer purring along properly, you must have the correct tools. The "tools" in this book are short 5 to 15 minute subroutines that combine color, sound and graphics to form a variety of educational programs and computer games.

- Forget the structured method of learning programming. Forget commands, statements, and data structures.
- Forget those frustrating evenings with your User's Manual.
- Learn to look at programs in terms of their working parts—their subroutines—and learn how to write simple programs.
- Learn how to use color, sound and music, graphics, animation, and computational subroutines as "tools" to build programs.
- Discover the modular form of programming. Learn what each subroutine will do, how it can be changed, and what the variables control.

Don't spend long frustrating hours poring over programming books and manuals. Follow the "Tool Kit" approach to programming and learn to design your own games and quizzes. Keep your VIC 20 computer humming happily.

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

$8.95/22309

ISBN: 0-672-22309-0