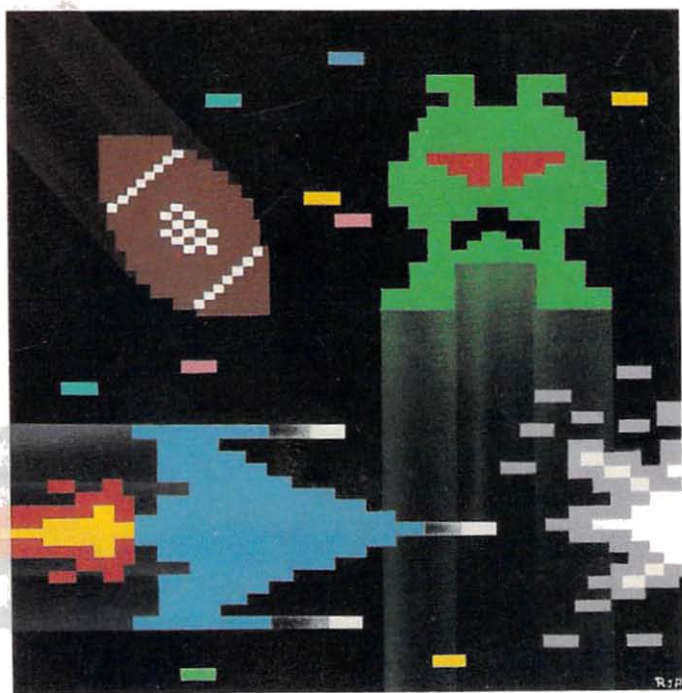




COMMODORE 64 GAME CONSTRUCTION KIT

BY WILLIAM L. RUPP AND PATRICIA A. HARTMAN



Create Your Own Commodore BASIC Games!

***Commodore 64
Game Construction Kit***



Commodore 64 Game Construction Kit

by
William L. Rupp
Patricia A. Hartman, Ph.D.

 **DATAMOST™**

20660 Nordhoff Street
Chatsworth, CA 91311-6152
(818) 709-1202



ISBN 0-88190-293-4

Copyright © 1984 by DATAMOST, Inc.
All Rights Reserved

This manual is published and copyrighted by DATAMOST, Inc. Copying, duplicating, selling or otherwise distributing this product is hereby expressly forbidden except by prior written consent of DATAMOST, Inc.

The word Commodore and the Commodore logo are trademarks of Commodore Business Machines, Inc.

Commodore Business Machines was not in any way involved in the writing or other preparation of this manual, nor were the facts presented here reviewed for accuracy by that company. Use of the term Commodore should not be construed to represent an endorsement, official or otherwise, by Commodore Business Machines, Inc.

Printed in U.S.A.

ACKNOWLEDGEMENT

We would like to express our thanks to William B. Sanders for his guidance in the course of this project.

PUBLISHER'S NOTE

Throughout the *Commodore 64 Game Construction Kit*, the Commodore 64 special keyboard graphics characters are, for the most part, not included in the program listings. Instead, these graphics characters are represented by their ASCII values. So, a program line such as

```
PRINT "  "
```

is represented thus:

```
PRINT "{#115}"
```

In these cases, look up the ASCII value in the ASCII and CHR\$ Codes chart in Appendix F, and substitute the correct graphics character for the ASCII value.

CONTENTS

Introduction.....	9
Part 1 — Planning Your Game.....	11
Chapter 1 — What's Your Game?	13
Chapter 2 — Road Mapping Your Game	17
Chapter 3 — Commodore 64 Game Construction Tools... ..	21
Part 2 — Games with Words and Numbers.....	25
Chapter 4 — Games with Numbers.....	27
Chapter 5 — What's My Number?.....	39
Chapter 6 — Take a Guess	47
Chapter 7 — Games with Words	51
Chapter 8 — LIGHTS! CAMERA! ACTION!	55
Chapter 9 — Ask the Wizard	65
Part 3 — Putting Pictures into Your Games	77
Chapter 10 — Basic Graphics Tools	79
Chapter 11 — Commodore 64 Keyboard Graphics	83
Chapter 12 — Animal Jump: A Child's Game	87
Chapter 13 — Making Your Own Characters.....	103
Chapter 14 — Ask the Wizard: With Graphics	117
Chapter 15 — Magic Cards.....	129
Part 4 — Advanced Game Tools and Techniques... ..	169
Chapter 16 — Enhancing Commodore 64 Games	171
Chapter 17 — Managing Memory	173
Chapter 18 — Animation.....	183
Chapter 19 — Using Animation in Games	197
Chapter 20 — Joysticks and Game Paddles.....	203
Chapter 21 — Sprite Graphics	211
Chapter 22 — Animating Sprites.....	223
Chapter 23 — Saving Sprite Data in Sequential Files... ..	235
Chapter 24 — Commodore 64 Sound.....	241
Part 5 — Creating Action Games.....	245
Chapter 25 — Computer Simulations	247
Chapter 26 — Monster Maze	251
Chapter 27 — Interceptor.....	263

Part 6 — Strategy and Fantasy/Adventure Games...	275
Chapter 28 — Strategy and Map Games	277
Chapter 29 — Rocket to the Green Planet	291
Chapter 30 — A Day at the Races	303
Chapter 31 — Goal to Go!	317
Chapter 32 — Fantasy/Adventure Games	341
Chapter 33 — Danger Dungeon.....	343
Chapter 34 — Danger Dungeon: Graphics and Sound ...	359
Chapter 35 — Scary Hall	379
Part 7 — Wrap-Up	411
Chapter 36 — Ideas for the Future	413
Chapter 37 — Summary	419
Appendix A — Screen Memory Map	425
Appendix B — Color Memory Map	427
Appendix C — Color Codes	429
Appendix D — Custom Character Drawing Grid....	431
Appendix E — Sprite Drawing Grid	433
Appendix F — Commodore 64 Character Sets	435
Index	439

INTRODUCTION

The *Commodore 64 Game Construction Kit* has been designed for those owners who are excited by the idea of creating their own computer games. It's likely that one reason you bought your Commodore 64 was to play games. That's easy to understand. Games, from the bone-dice throwing of ancient Roman soldiers to the video-arcade games of today, have long been an important part of human culture.

Having fun is not the only reason to play games on your Commodore 64. Games are a way for children to imitate their parents and therefore serve as a preparation for growing up. In school, students play *Hangman* to sharpen their problem solving and spelling skills. When children progress to high school and beyond, their games become more sophisticated. Role-playing simulations in social studies classes are good examples. Even a flight simulator in pilot training can be thought of as a type of game.

The *Commodore 64 Game Construction Kit* will help you take advantage of the Commodore 64's many advanced features. The C-64 has powerful capabilities in graphics and sound, as well as a large memory capacity. You will discover that these features can be used to good advantage in the design of microcomputer games.

There are many types of computer games. We have included text games, text games with graphics and sound enhancements, and games which are primarily graphics oriented. Each type will be studied carefully.

So let's get down to business. How much knowledge must you have to begin this book? We do not ask that you be an advanced programmer to use the *Construction Kit*. We have tried to make each chapter clear and easy to understand. However, any BASIC you already know will come in handy.

You have the *Commodore 64 User's Guide* which came with your computer. We also recommend that you also obtain a book such

as the *Commodore 64 Programmer's Reference Guide*, or William Sanders' *Elementary Commodore 64*. The *User's Guide* is quite detailed, whereas Sanders' book is more general. Even if you haven't read any of these books, it is helpful to keep one on hand for reference.

The only requirement is that you enjoy games and can't wait to write some of your own. If you fit this description, you are ready to program!

The *Commodore 64 Game Construction Kit* teaches you how the different game types operate and gives complete running examples of each type. We have also included complete program listings for you to study and enjoy.

The book starts with some very simple games which illustrate fundamental points of game programming. Later chapters will explain basic aspects of graphics and sound, and how these features can be added to the earlier games. As the book progresses, we will include more sophisticated programming techniques. We hope that these tools provide a basis for your game designing career.

Before beginning, we want to clarify one point. Do not expect to be able to create the next *Pac-Man* or *Asteroids* after reading this book. Those arcade-style games are all written in machine language, which is rather difficult and beyond the scope of this book. Even in BASIC, however, it is possible to create some exciting games with animation and sound. The only limitations are your imagination and enthusiasm!

Part 1

PLANNING YOUR GAME



Chapter 1

WHAT'S YOUR GAME?

What is a Game?

According to some authorities, the word "game" is related to the old Gothic word "gaman," which means companion or companionship. That's an interesting idea when you think about it. Games do keep us company in a very real sense. Games can be played by individuals wanting to pass the time enjoyably, between two people, between two teams, and among many people playing on their own.

What kind of games do you like to play? What is it about those games which make them good companions? Before we get to the specifics of constructing games on the Commodore 64, it's worthwhile to think a little about the various categories of games.

One obvious category includes football, baseball, hockey, basketball, soccer, and other team sports. These have the following common characteristics: They involve physical action, they involve two organized teams, and they are close-ended (they end at a predetermined point—60 minutes of play, 9 innings, etc.). Other physical games are tennis, running, golf, swimming, etc. While some of these are occasionally team sports, they all involve a series of one-on-one contests. Each has a predetermined ending point as in the first category.

Another major game category includes card games, checkers, chess, monopoly, as well as many other board games. These also usually have predetermined ends. Some, like chess and checkers, are contests between two persons. Many, however, involve three or more people playing independently.

Many people would stop there, but we think that the term game has a much broader definition. In the examples given, the end of the game is predetermined. What about games in which the participants play with, rather than against, each other? Throwing a Frisbee to a friend is a game, playing peekaboo with a young child is a game, and building a sand castle at the beach is a game.

And there are other games which are very serious business, indeed, although the public is largely unaware of them. These are simulation games. Business, government, and the military, for example, use this technique to plan for the future. A large corporation may use a computer model (or simulation game) to estimate the outcome of a new marketing strategy. The Navy may use a computer model to determine how a new weapons system will work in combat. It's much less costly, and safer, to do your fighting on a computer.

What types of games can be played on a computer? It may be more appropriate to ask which can't be played on a computer. Even very physical games, like football, can be simulated on a computer. Of course, some games lend themselves more readily to computer adaptation than others. Blackjack works quite nicely on a computer. The cards can be displayed graphically and the element of chance is easily provided. Basketball, unless your system has extremely powerful graphics, is not quite so good. Football is excellent, because despite the tricky running of your favorite halfback, the game is still essentially a matter of teams moving up and down a field represented by a straight line.

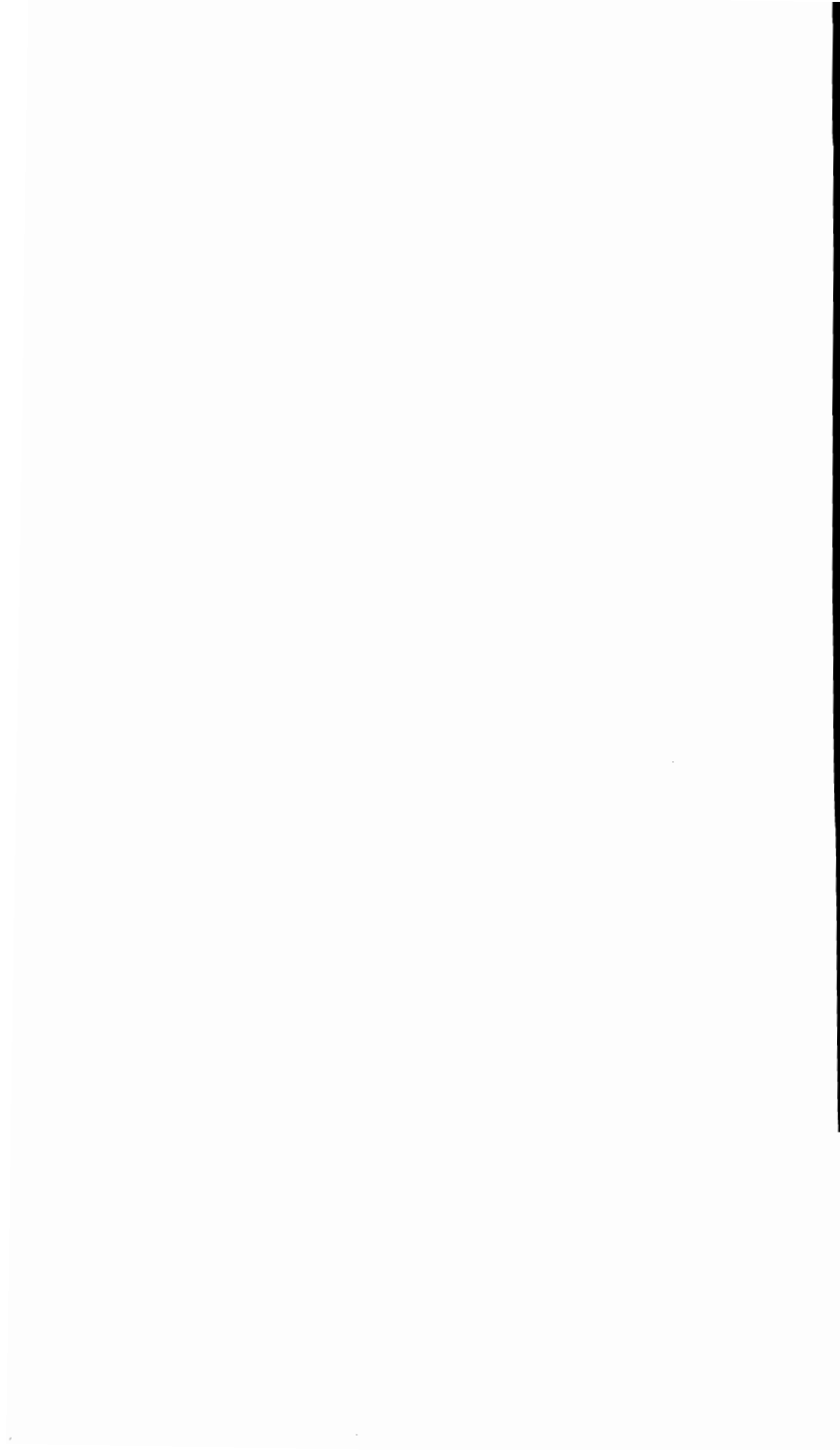
Planning Your Game

Our definition of the word game is not just a matter of idle curiosity to the Commodore 64 owner who wants to create original games. It is essential to understand the characteristics of any game before you can depict it on your computer. Let's list some of the questions you should ask yourself as you sit down to plan your game:

1. Is the game to be played by two parties? (This could be you playing against the computer, or you playing against a friend.) Or will three or more players be involved?

2. Will your game have a predetermined end point (the first one to reach 100 points, for example), or will it be an open-ended activity game (such as doodling or sand castle building) where the fun is entirely in the doing?
3. If the game is close-ended (finite), what will be the payoff? For example, in a target-shooting game, the payoff is hitting all the targets.
4. Which factors will be important as the game progresses? These factors are called variables. In football, the score is a variable, and so are the time remaining, the down, the position on the field, etc. In a target-shooting game, the number of shots fired and the number of shots left are variables. It is vital that all essential variables be identified.
5. What will be the method of play? Will players take turns? What decisions will players have to make as the game progresses? Which moves will be legal, which illegal?
6. How much information about the other participants will each player have? In chess, both players can see the entire board, which pieces are where, etc. In poker, however, players hide their cards until the decisive moment.
7. What surprises, enhancements, obstacles, sound and graphics effects do you want in your game? These factors may be window dressing, or they may critically affect the outcome of the game.

So you see, clarifying what you want your game to do before you begin to program is crucial. Do some thinking, then write a specific description of your game. To assist you in this process, we'll next examine a type of game plan called a road map.



Chapter 2

ROAD MAPPING YOUR GAME

There is an old saying among programmers that “the sooner you start writing code, the longer the program will take to complete.” This means that a program which is not well thought out will probably be rewritten again and again, because there was no clear understanding of its functions prior to writing the code. So before you get down to the actual writing of your program, it is essential that you carefully plan what you want your game to do.

We call this process road mapping. It is tempting to skip this step, just as many people skip over the introduction to a book (but not *this* book, right?). But resist the temptation, and believe us when we say that a well-conceived road map will make your game much easier to create in the long run.

The traditional approach to planning a program is to use the well-known flowchart. This method is excellent, but a flowchart is perhaps a little too detailed for our purposes. Road mapping is a very general listing of the major characteristics of a program.

Creating Road Maps

The first step in road mapping your game is to write down the answers to the following questions:

1. What is the subject of the game? Examples are card games, fantasy/adventures, mazes, and sports.
2. What is the definition of victory?

3. How does the player advance toward the goal?
4. What factors, variables, must be kept track of as the game proceeds?
5. What decisions must players make during the game?
6. What decisions must the computer make during the game?

The next step in creating your road map is to write a short scenario describing the game's action, much as a newspaper reporter would. Let's use baseball as an example. To keep things simple, we'll limit ourselves to the following elements: The pitcher throws the ball and the batter responds. We will not discuss outs, baserunners, the score, the inning, etc. Each step will be numbered for convenience.

Here goes:

1. The pitcher throws the ball.
2. The hitter decides whether or not to swing.
3. If the batter swings, skip to step 6.
4. When the batter does not swing, the pitch is either a ball or a strike.
5. Add 1 to the ball or strike count, whichever is appropriate. If that makes strike 3, the batter is out. If there are 3 outs, then the game is over. If it's ball 4, the batter walks. In either case, go to step 1 for a new batter.
6. If the ball is hit fair, go on to step 9.
7. If the batter swings and misses, add one to the strike count. If it's strike 3, the batter is out; go back to step 1.
8. If the batter hits a foul, add 1 to the strike count, unless it's already strike 2.
9. Return to step 1 for the next pitch.

Obviously there is a great deal more to baseball than what we've described (what about foul balls caught by fielders, for example?). The point is, you must have a clear idea of how your game will proceed. You must also identify the decisions to be made during the game. In the above scenario, the first decision was made by the batter, to swing or take the pitch. The decision following a swing was whether the batter missed, hit fair, or hit a foul. And so on.

Using Road Maps

The road map is not a program. Listing the decisions to be made in the course of a game does not automatically suggest program lines. For instance, the first decision in our baseball example would probably be made by the player, while the decision about whether the bat hit the ball would probably be made by the computer. What the road map does is set down the elements which must be included in your program. You must then write the code accordingly.

Another crucial factor is that of variables. It's impossible to overemphasize the importance of manipulating variables in your program. In our baseball example, there are six variables: PITCH, which can either be a swing or a take; SWING, which can either be a contact or no contact; CONTACT, which can either be fair or foul; TAKE, which can be ball or strike; BALL, which can be 0 to 4; STRIKE, which can be 0 to 3. If we were to keep track of outs, the variable OUTS would be added, with a value of 0 to 3.

Some variables change often between only two possible values (e.g., PITCH is different each time, but must always have the value of either SWING or TAKE). Other variables have a running total kept, such as BALL and STRIKE. Even these are set back to 0 with a new batter, of course.

To summarize, a road map, which is a general description of how a game is played and what decisions and variables must be considered, is a useful first step in creating a game. If your game is a simple one, your road map may be short. The road map, however, can be equally important to the final product, whether it be a short or long game.



Chapter 3

COMMODORE 64 GAME CONSTRUCTION TOOLS

Your Commodore 64 will be the setting for all the games you create. Let's take a look at the C-64 and its many capabilities. It is important to keep in mind the abilities and limitations of this or any computer as you begin designing games.

Hardware

The Commodore 64 is a microcomputer—so called because it uses an 8-bit microprocessor, the 6510. (If you have no idea what a microprocessor is, don't get discouraged. You will be able to use this book just the same.) The microprocessor chip is the device which actually does the calculations inside the computer as the program is run. The 6510 microprocessor is an 8-bit model because 8 bits is the most it can handle at one time. (A bit is a single piece, or unit, of storage information.) The Commodore 64 also has microprocessor chips which control graphics and sound generation. Quite a lot is packed into the C-64's small package.

More important to us here is the memory size of the Commodore. The C-64 holds 64K, or 64 times 1024 bytes. Each byte is comprised of 8 bits. A byte can most easily be thought of as a single character. A character can be any letter, number, or special symbol, such as punctuation. Even a blank space is a character.

When you turn on your computer, you will notice the message that 38,911 bytes are free. This means that there is room for 38,911 bytes, or characters, in memory for you to work with. Since 38,911

bytes is a great deal of memory, it is unlikely that you will create many games that are too large to fit into the computer's memory.

The Commodore 64 has some powerful hardware features which will come in handy as we think about computer games. For one thing, the C-64 has very strong graphics capabilities; these include special graphics symbols right on the keyboard, as well as sprites. Sprites are small, user-designed characters which can be moved around the screen independently.

Another nice feature of the C-64 is its four-voice sound synthesizer. You will soon see, or rather hear, what that capability can add to a game. Still another plus is the keyboard editing features which help make programming much easier.

One feature of the Commodore 64 which we like very much is its set of four special-function keys, located on the right-hand side of the keyboard. Using the SHIFT key, you can define eight distinct functions with these keys. For example, in our football game *Goal to Go!* (Chapter 31) players use the function keys to choose plays.

Input and output, or I/O, are terms you should understand. The microprocessor would be useless if there were no way to communicate with it. I/O allows you to put information into and get information out of the computer. The most important input device we will discuss is the keyboard. Other useful input devices for games are the joystick and game paddles. As to output, we will deal with the video screen (a TV or special computer monitor) as well as sound via the speaker on your video monitor or television.

Software

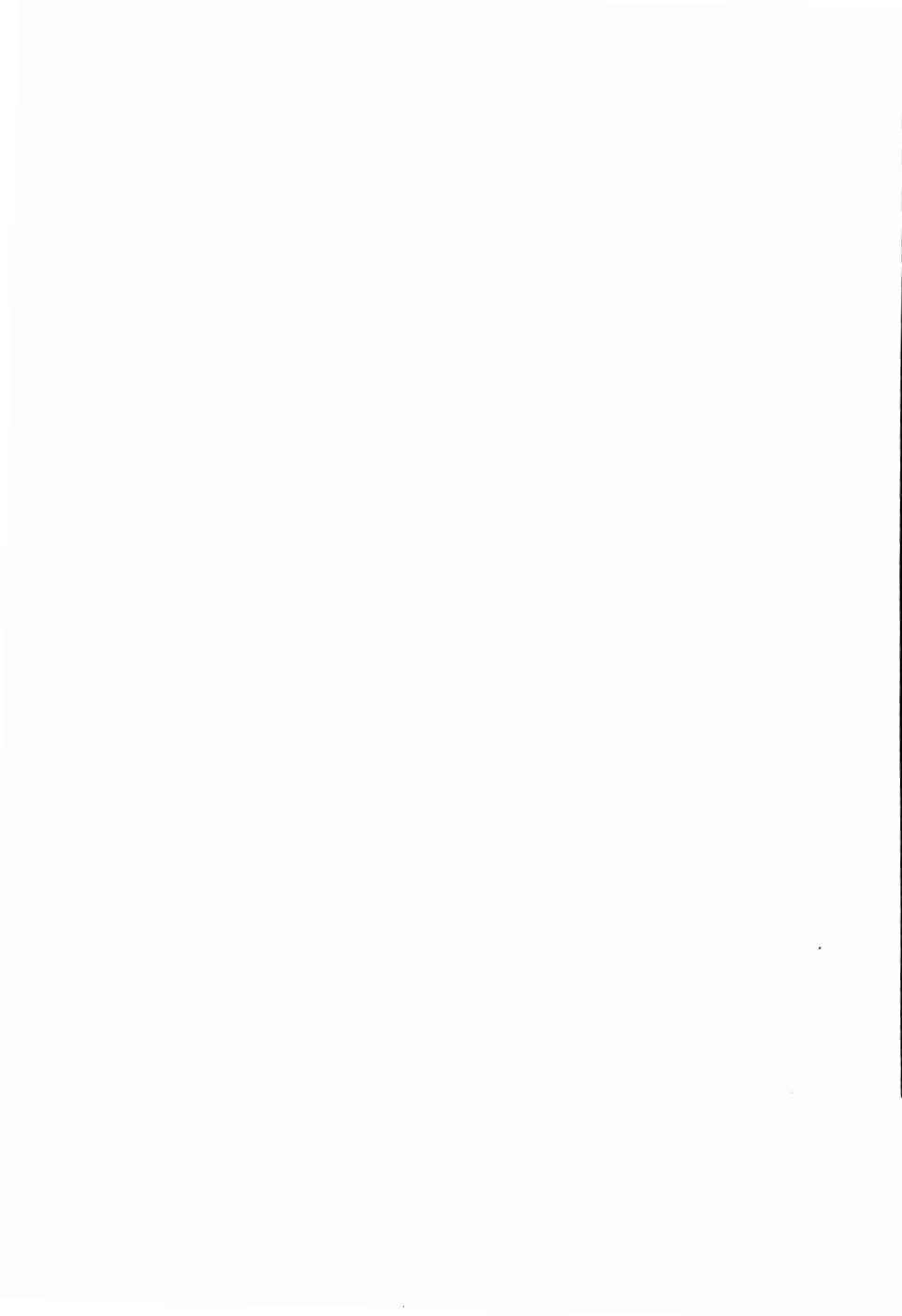
In creating games, or any programs for that matter, you must communicate with your Commodore in a language it can understand. BASIC is the language we will use exclusively in this book. Towards the end we will briefly mention machine language, which is what the 6510 microprocessor actually uses.

BASIC is called a high-level language. This simply means that it is similar to human language. Machine language, which consists of strings of 1s and 0s, is the language the computer understands. BASIC is not the most advanced computer language, but it has a number of advantages, chiefly its English-like vocabulary and syntax.

Summary

Some of the tools needed to construct games on your Commodore 64 are really concepts, such as careful definition of your game before you begin programming. We will introduce many specific programming techniques along the way as well. Some of these are variables, loops, counters, DATA statements, flags, subroutines, and IF/THEN statements. Each of these tools (as we like to call them) can help you create more complex and entertaining games.

That's a very brief look at the Commodore 64's many strong points, all of which will be put to good use in the coming chapters.



Part 2

GAMES WITH WORDS AND NUMBERS

Chapter 4

GAMES WITH NUMBERS

Numbers are the foundation of every computer program. It is appropriate, therefore, to begin our study of Commodore 64 game construction by examining how numbers function in computer programs. In this chapter we will concentrate on numeric variables and how to manipulate them in computer games.

Numeric Variables

A numeric variable is a symbol standing for a number. A numeric variable must begin with a letter of the alphabet, but may also contain numbers. Here are some possible variable names: A, X, CC, D1, F3, PAYROLL, PAYCHECK. Any of these variables can represent any number your Commodore 64 is capable of calculating. There is, however, an important warning to keep in mind. Only the first two characters of a variable name are significant. The Commodore 64 will ignore all characters following the first two.

The limitation of two significant characters does not mean that a variable must have only two characters. The variable names PAYROLL and PAYCHECK are both okay. What the two-character limitation means is that PAYROLL and PAYCHECK should not be used in the same program. Why not? Because your Commodore 64 will see them as the same variable. The PA at the beginning of each of these examples is all the Commodore cares about. Since PAYROLL and PAYCHECK obviously refer to different things, you would want to give them different names, such as PR and PC.

Integer Variables

There are three other variable types which can be used in games on the Commodore 64. One of the other types, the integer variable, is a relative of the numeric variable. The only difference is that the integer variable stores whole numbers only. The advantage of this stems from the fact that integer variables occupy less memory and allow greater speed of execution.

If you were writing a game which recorded the total of two dice ($D1+D2=DT$ or Dice Total), the total (DT) must be a whole number between 2 and 12. Rather than using a numeric variable like DT which, when used, causes the computer to look for a number and decimal places, the dice total could be stored in an integer variable. If this were done, the computer would look at the variable (say DT%), know that the variable contained no decimal places, and forego the process of checking their value. As you may have guessed, integer variables look exactly like numeric variables, except integer variables must end with a percent sign (%).

Arrays

An extremely useful type of variable is the array. An array is a variable with two or more subsections, each of which can have a different value (such as male, female being subsections of sex). Perhaps you want to calculate the points earned by three different players in a game. You could program it in this fashion:

```
10 REM *****
11 REM * ARRAY EXAMPLE *
100 REM * XY STANDS FOR A GAME *
101 REM * CONDITION OR FACTOR. *
105 REM * A, B, AND C STAND *
106 REM * FOR PLAYERS 1-3 *
107 REM *****
108 XY=9:A=5:B=3:C=12:REM- A, B, AND C
ARE INDIVIDUAL OUTCOMES-
110 P1=A * XY
```

```

120 P2=B * XY
130 P3=C * XY
135 PRINT " PLAYER 1", " PLAYER 2", " PLAYE
R 3"
140 PRINT P1,P2,P3
150 END

```

As written, the program is virtually meaningless. This is because lines 100-150 are assumed to belong in a larger program in which variables A, B, and C are defined. Instead of using the three variables A, B, and C, you could use an array. Let's call it A(3). (We will identify arrays in one of two ways: A(15), the number in parentheses indicating the total number of elements in that array; or A(), indicating that variable A is an array variable, not a numeric.)

A(3) is a one-dimensional array. A one-dimensional array may be thought of as a single-story apartment building. If the building is called Regal Apartments and has apartments 1, 2, and 3, those apartments would be, in array terms, Regal(1), Regal(2), and Regal(3).

Here is another version of the above example, this time using an array:

```

10 REM *****
11 REM * ARRAY EXAMPLE *
100 REM * XY STANDS FOR A GAME *
101 REM * CONDITION OR FACTOR. *
105 REM * A(#) STANDS FOR PLAYERS *
106 REM * 1 - 3 *
107 REM *****
108 XY=9:A(1)=5:A(2)=3:A(3)=12:REM-
    ARRAY A(#) STANDS FOR PLAYERS-
110 P1=A(1) * XY
120 P2=A(2) * XY
130 P3=A(3) * XY

```

```
135 PRINT " PLAYER 1", " PLAYER 2", " PLAYE  
R 3"  
140 PRINT P1,P2,P3  
150 END
```

One advantage of an array is that it economizes on variable names. In the first example, three variables (A, B, and C) were needed to represent game factors for three players. In the second example, the array eliminated the need for the second and third variable names. The three factors are elements (subdivisions) of the array as a whole.

Arrays and FOR/NEXT Loops

Another advantage of arrays is that they are easy to use with FOR/NEXT loops. In this sample program, 100 random numbers are generated, stored in an array, then printed out:

```
100 REM - GENERATE 100 RANDOM NUMBERS -  
110 DIM N(100)  
120 FOR I = 1 TO 100  
130 N(I)= RND(1)*11  
140 NEXT  
150 REM - DISPLAY ALL 100 NUMBERS -  
160 FOR I=1TO 100  
170 PRINT "NUMBER "I" IS "N(I)  
180 NEXT  
190 END
```

DIMensioning Arrays

Enter and run the program. That's a lot of numbers and words appearing on the monitor screen for so few program lines! That shows you some of the power of FOR/NEXT loops combined with arrays. Arrays are very powerful, especially for games and other programs. Later in this book you will see several examples of arrays used in our games.

If there are no more than 10 elements in an array, you can simply use it with no preparation. If the array is to have more than 10 elements, you will have to DIMension the array. To DIMension an array with 21 elements, you would use a statement such as:

```
100 DIM A (20)
```

This tells the Commodore 64 to expect up to 21 values to be stored as elements of an array named A. Keep in mind that array A() is not the same as the simple variable A. You can have both A and A(1) in the same program.

Even though arrays of up to 10 elements need not be DIMensioned, many programmers recommend that you DIMension all arrays. This practice clearly identifies which arrays will be used in the program. Arrays should be DIMensioned near the beginning of the program.

Multidimensional Arrays

So far we have spoken only of single-dimension arrays. It is possible to use arrays of two-, three-, four-, or more dimensions. Think of the Regal apartment building again. If a second floor is added, we would need a two-dimensional array to describe the building. Floor one's units would be Regal (floor 1, apartment 1), Regal (floor 1, apartment 2), Regal (floor 1, apartment 3); floor two's would be Regal (floor 2, apartment 1), Regal (floor 2, apartment 2), and Regal (floor 2, apartment 3). Or, more graphically:

```
REGAL: Floor 1 (apt. 1), Floor 1 (apt. 2), Floor 1 (apt. 3)  
        Floor 2 (apt. 1), Floor 2 (apt. 2), Floor 2 (apt. 3)
```

This two-dimensional array would be DIMensioned with the statement:

```
100 REGAL (2,3)
```

This means that the array called REGAL has two main elements, each of which has three subelements. Another way to think of this example of a two-dimensional array is:

```
REGAL (1,1) = 101
REGAL (1,2) = 102
REGAL (1,3) = 103
REGAL (2,1) = 201
REGAL (2,2) = 202
REGAL (2,3) = 203
```

The numbers after the equals signs represent each value of the array. Each element can have a different value, any numeric value as a matter of fact, that the C-64 can handle.

Using Arrays

Arrays are useful in many games. For example, imagine a game in which players have markers of some sort which move around a course or track on a map. The course is marked off in spaces. As each player's turn comes up, the marker will move one or more spaces, or stay put. You could store the possible moves in an array:

```
100 DIM A(5)
110 A(1)=3
120 A(2)=5
130 A(3)=-1
140 A(4)=0
150 A(5)=4
```

By use of a random-number routine, each move is selected from the five possible moves contained in the array shown above. Arrays can also be used with words, but that is a topic for the next section.

String Variables

The last variable type is perhaps the most important. String (or alphanumeric) variables can contain virtually any data (as can string array variables) including numbers, letters, words, special characters, and anything else. There are two ways to identify string variables: First, they must end with a dollar sign (\$); second, their contents must be enclosed by quotation marks.

See if you can identify the mistakes below:

```
10 A="BOY"  
20 A$=BOY  
30 A(22)=8  
40 A%=3.14
```

Here are the answers: Line 10 attempts to store a character string (BOY) into a numeric variable (A); line 20 attempts to store a character string in a string variable, which is fine, except BOY is not enclosed by quotation marks ("BOY"); line 30 attempts to store 8 in A(22)—again fine, except that A has not been DIMensioned; line 40 attempts to store a non-integer in an integer variable.

The following lines are correct:

```
10 A=5  
20 A$="CAT"  
30 DIM A$(25)  
40 A$(10)="DOG"  
50 A%=35
```

Random Numbers (RND)

Since we have mentioned the term "random number," the random number function (RND) is worth examining closely. First of all, what is randomness? To speak in very precise mathematical

terms, a random event is one which follows no predictable pattern. To say that you are picking a random name from a list of 10 names implies that each name on the list has an equal possibility of being selected. Furthermore, if 10, 100, 1000, or 1,000,000 selections are made from the list, no repeating pattern of selections will be observable.

The random (RND) function of the Commodore 64 is one of the most useful in game construction. Computer simulations rely on random number selection a great deal. When random number selection is combined with probability theory, all sorts of real world events can be simulated by the computer.

Here is how RND works. Enter this line on your Commodore in immediate mode and see what happens:

```
PRINT RND ( 1 )
```

Try it several times. You see a decimal point followed by 10 digits ranging from 0 to 9. Enter this line several more times, selecting different numbers or letters to go inside the parentheses. You still get back fractional numbers, right? The character inside the parentheses is called the argument, and in this case it is a dummy argument. That means that any number or letter would do as well as any other.

Now try this:

```
PRINT RND ( 1 ) * 10
```

Try it several times and you will get back numbers ranging from 0.00000000 to 9.99999999. Each number has a fractional part of eight places. Now try this:

```
PRINT INT ( RND ( 1 ) * 10 )
```

Suddenly you are getting whole numbers! Try the line several times. Did you notice that for the first time the zero appeared? The `INT` stands for integer. `INT` changes the fractional number delivered by `RND` into a whole number. It does this by dropping the digits to the right of the decimal point. Even 1.99999999 will be rounded off to 1 by `INT`.

RND and INT

`INT` combined with `RND` can be very handy in game construction. Let's go back to the array shown earlier. Using the `INT` and `RND` functions in conjunction with the array, we can produce random moves for the players of the hypothetical game mentioned above. Try this little program on your Commodore 64:

```
100 DIM A(5)
110 A(1)=3
120 A(2)=5
130 A(3)=-1
140 A(4)=0
150 A(5)=4
160 FOR I=1 TO 25
170 NUM= INT(RND(1)*5 +1)
180 MOVE =A(NUM)
190 PRINT"NEXT PLAYER'S MOVE IS.."
MOVE
200 NEXT I
```

The variable `NUM` is always a randomly selected whole number from 1 to 5. Did you notice that we added "+1" in line 170? That is important, because without it, you would be getting random whole numbers from 0 to 4. Whenever the `INT(RND)` function yields 0, the +1 changes the 0 to 1. On the other end, whenever the `RND(INT)` function yields 4, the +1 changes it to 5.

In this case, the only possible moves are the five elements of array `A(5)`. You could have an array of 10, 100 or even 1000 elements, each one a possible move number. Regardless of the number of

elements in the array, the same program routine (algorithm) would still work.

Lines 170 and 180 are the most important lines in this sample random/array program. In 170, a random number from 1 to 5 is assigned to variable NUM. In 180, the variable MOVE is assigned the value of the element in the array A(5) matching the numerical value of NUM. In other words, if NUM equals 3, then $MOVE = A(NUM)$ gives MOVE the value of element NUM in the array. If NUM is 3, then A(NUM) means A(3). If NUM equals 5, then A(NUM) means A(5). Run the program several times and see how different values come up.

After having said so much about the RND function of the Commodore 64, we must tell you that its results are not truly random in a mathematical sense. The numbers delivered by RND are so close to being random, however, that you will not see the difference in your games.

Random Numbers and Arrays

Arrays and random numbers can be combined with probabilities to give remarkable realism to computer games. If you are creating a war game, you can use arrays, random numbers, and probabilities to determine the outcome of battles in a realistic fashion. If the chances of two ships winning a naval battle are even, then you can figure the winner in this manner:

```
100 WINNER =INT(RND(1)*10+1)
110 IF WINNER>5 THEN PRINT "WINNER IS
SHIP ONE."
120 IF WINNER<6 THEN PRINT "WINNER IS
SHIP TWO."
```

Since the odds are 50/50 here, it is easy to select the winner on the basis of whether the random number is 1-5 or 6-10.

More complicated is the situation in which one ship is more likely to win than the other (e.g., cruiser versus gunboat). What the game designer must do is to calculate the odds. Let's say that the cruiser has a 60 percent chance of winning. Try this modified program:

```
100 WINNER= INT(RND(1)*10+1)
110 IF WINNER >6 THEN PRINT "GUNBOAT
WINS."
120 IF WINNER <7 THEN PRINT "CRUISER
WINS."
```

The variable WINNER is a random number. Six out of 10 times, on average, WINNER will have a value of 6 or less, which means that the cruiser will be declared the winner 6 out of 10 times, or 60 percent of the time. Just what we set out to simulate! By using a random number from 1 to 100, a finer scale of probability could be used.

You can use this random number feature to select outcomes from an array. One method is to assign outcomes in a graduated scale. Element 1 in an array could be the best outcome, and element 10 (or 100 depending on the size of the array) could be the worst. It is fun playing around with this technique until you design a routine which accurately reflects the probabilities you want to incorporate in your game. Don't forget, just because one side has tremendous superiority, that doesn't mean the weaker side will always lose. Probabilities just tell us what will happen most of the time. Occasionally even the gunboat wins!



Chapter 5

WHAT'S MY NUMBER?

Perhaps the easiest type of game to program is the guessing game. Here it is possible to write a set of instructions that tell the computer its part in the script and also what it must do in response to you. You are creating a companion, another intelligence according to your specifications.

For starters, key in the following lines of code, then RUN the program.

```
10 REM  WHAT'S MY NUMBER GAME
20 N=4
30 PRINT "I AM THINKING OF A NUMBER
BETWEEN 1 AND 10 ": PRINT
40 INPUT "WHAT IS THE NUMBER ";A$:
A=VAL(A$)
50 IF A=N THEN PRINT "YOU ARE
FANTASTIC! "
60 END
```

A BASIC Review

Now let's look at *What's My Number?* and review the BASIC statements we used. (If you are familiar with the fundamentals of BASIC programming, you may wish to skip this section.)

Line Numbering

In BASIC, the order in which a program is executed is determined by the line number sequence. Line 10 is executed before line 20, line 100 before line 1000, and so on. It is good practice to number by tens. In this way, it is possible to later add lines which were initially left out. Lines may be added at the bottom of the program, and the computer will insert them in numerical order.

REMark Statements

It is often important to describe what a program does. The REMark statement is used to remind the programmer what is taking place in the program. REMarks have nothing to do with how the program works. They only document what the program does. They should be inserted (1) at the beginning of a program, (2) to define variables, (3) directly before particular operations which need to be explained, and (4) to make explanations useful to the programmer or whomever will be reading the program listing.

PRINT Statements

PRINT statements tell the computer to display a message on the video monitor. PRINT statements also send data to the disk drive and the printer. The PRINT statement is followed by a double quotation mark, the phrase to be displayed, and another double quotation mark. Sometimes more than one operation is written on a line. When this is the case, the operations are each separated by a colon (see line 30). PRINT is also used by itself to move the cursor down one or more lines. The cursor scrolls down one line each time the PRINT statement is used. To scroll, PRINT may be used before or after the statement which is to be displayed. If it is used before the phrase, the scrolling will occur before the phrase is PRINTed. If it is used at the end of the phrase, the scrolling will occur after the phrase is PRINTed.

INPUT

The INPUT statement is used when data is needed by the program. The input required is in the form of a user response.

IF/THEN

The IF/THEN statement is used when a decision must be made in the program. If a certain condition is met, the program executes the THEN statement, otherwise it skips that statement and goes on. For example, line 50 of our program reads:

```
IF A=N THEN PRINT "YOU ARE FANTASTIC!"
```

The program will print "YOU ARE FANTASTIC!" if, and *only* if, variable A equals variable N; and, in this case, N equals 4, so "YOU ARE FANTASTIC!" is only printed when A equals 4.

Enhancing *What's My Number?*

Unfortunately, our game isn't too interesting. Since N always equals 4, it won't take a player too long to figure out how to win every time. So to make the game more challenging, key in and run the following lines:

```
10 REM WHAT'S MY NUMBER GAME  
20 N=INT(1+(10-1+1)*RND(1))  
30 PRINT "I AM THINKING OF A NUMBER  
BETWEEN 1 AND 10":PRINT  
40 INPUT"WHAT IS THE NUMBER?";A$:  
A=VAL(A$)  
50 IF A=N THEN PRINT"YOU ARE  
FANTASTIC! ":GOTO 70  
60 IF A<>N THEN PRINT "WRONG! WRONG!  
IT WAS ";N: GOTO 70
```

```

70 INPUT "PLAY AGAIN? (1=y 2=n) ";G$:
G=VAL(G$)
80 IF G = 1 THEN GOTO 10
90 IF G > 1 THEN GOTO 100
100 END

```

What happens now? Is the game more interesting? What else would make the program more understandable for the programmer? How about identifying the variables? Let's add some REMark statements to explain what's going on.

```

11 REM N=THE RANDOM NUMBER SELECTED
12 REM A=THE USER'S GUESS
13 REM G$=THE USER'S RESPONSE TO A
REPEATED GAME

```

These lines do not help the program run, but they identify functions of the program. Without REMark statements, it is often impossible for even an experienced programmer to decipher another programmer's work. Documentation (using REMark statements) saves a lot of trouble in the long run and is not usually time consuming during program construction.

Now our program looks like this:

```

10 REM WHAT'S MY NUMBER GAME
11 REM N=THE RANDOM NUMBER SELECTED
12 REM A=THE USER'S GUESS
13 REM G$=THE USER'S RESPONSE TO A
REPEATED GAME
20 N=INT(1+(10-1+1)*RND(1))
30 PRINT"I AM THINKING OF A NUMBER
BETWEEN 1 AND 10":PRINT
40 INPUT"WHAT IS THE NUMBER?";A$:
A=VAL(A$)
50 IF A=N THEN PRINT"YOU ARE
FANTASTIC! ":GOTO 70

```

```

60 IF A<>N THEN PRINT "WRONG! WRONG!
IT WAS ";N: GOTO 70
70 INPUT "PLAY AGAIN? (1=y 2=n) ";G$:
G=VAL(G$)
80 IF G = 1 THEN GOTO 10
90 IF G > 1 THEN GOTO 100
100 END

```

What happens when you RUN the program now? Can you think of anything to add? What about increasing the number of guesses for each number? We don't seem to have enough chances to guess the correct number. One in 10 are not great odds. What other possibilities are there? Let's experiment with some additional statements.

Let's change line 60 (the old line 60 will be replaced when you key in the new line and type RETURN). Next, add two new lines, 61 and 62.

```

60 IF A<>N THEN GOTO 61
61 IF A<N THEN PRINT "TOO LOW--TRY
AGAIN!":GOTO 40
62 IF A>N THEN PRINT "TOO HIGH--TRY
AGAIN!":GOTO 40

```

Now you can guess more than once for each number the computer selects. Does this addition make the game more interesting?

Let's review the functioning of the program. Here's the complete listing:

```

10 REM  WHAT'S MY NUMBER GAME
11 REM  N=THE RANDOM NUMBER SELECTED
12 REM  A=THE USER'S GUESS
13 REM  G$=THE USER'S RESPONSE TO A
REPEATED GAME
20 N=INT(1+(10-1+1)*RND(1))

```

```

30 PRINT "I AM THINKING OF A NUMBER
BETWEEN 1 AND 10":PRINT
40 INPUT "WHAT IS THE NUMBER?";A$:
A=VAL(A$)
50 IF A=N THEN PRINT "YOU ARE
FANTASTIC! ":GOTO 70
60 IF A<>N THEN GOTO 61
61 IF A<N THEN PRINT "TOO LOW--TRY
AGAIN!":GOTO 40
62 IF A>N THEN PRINT "TOO HIGH--TRY
AGAIN!":GOTO 40
70 INPUT "PLAY AGAIN? (1=y 2=n) ";
G=VAL(G$)
80 IF G = 1 THEN GOTO 10
90 IF G > 1 THEN GOTO 100
100 END

```

Error Trapping

Before we explain the program line by line, it is important to clarify the purpose of the VAL statements in lines 40 and 70. As mentioned, numeric variables can assume different forms. But what happens if you try to store a word, "HELLO" for example, in a numeric variable location? The program crashes, that's what! Data contained within a string variable cannot be used in any arithmetic operations. Why is this important? Look at the following program:

```

10 A$="5"
20 B$="2"
30 C=3
40 PRINT A$*B$:PRINT B$*C:PRINT A$-C

```

Lines 10-30 are perfectly valid instructions, but line 40 won't work. If you try running this program you will get an error message. This is because line 40 attempts to use string variables (A\$,B\$) in arithmetic operations. It is impossible for the computer to perform arithmetic operations on string variables, whether they are numbers or words. The two programs listed below are logically equivalent—also incorrect:

```
10 A$="2":B$="2"  
20 PRINT A$*B$
```

```
10 A$="TWO":B$="TWO"  
20 PRINT A$*B$
```

But what has this to do with *What's My Number?* Say, for example, that while playing *What's My Number?* you accidentally hit R instead of 5 for your guess at line 40. If the INPUT statement used a numeric variable, say A, then keying R would cause the program to crash; but we have eliminated that problem. In BASIC, each character has a corresponding numeric value, known as its ASCII value. The VAL function converts a string variable to a numeric variable. Letters are converted to numerics (ASCII values) thereby getting around the fatal mistake of entering a character string into a numeric variable.

This sort of mistake prevention is called "error trapping." Line 40 asks for input into a string variable (A\$). Since A\$ is arithmetically useless, a VAL statement is used to convert the string A\$ into the numeric A. Line 70 functions identically. Just so you know, if A\$ equals "2", the VALue of A\$ (VAL(A\$)) is 2.

An Analysis

Now, as promised, a line-by-line explanation of *What's My Number?*:

LINE 10: Program title.

LINE 11: Identifies N as the name for the random number variable.

LINE 12: Identifies A as the name for the variable that is the user's guess.

LINE 13: Identifies G\$ as the name for the user-response variable in which the user is asked to play again.

LINE 20: Assigns a function to N, which produces a random number between 1 and 10.

LINE 30: Produces a screen display of the phrase between quotes.

LINE 40: Asks for user INPUT. Here the user is requested to guess a number.

LINE 50: Sets up a condition for a correct response, then moves to line 70.

LINE 60: Sets up a condition for an incorrect response, then moves to line 70.

LINE 61: Sets up a condition for a guess lower than the correct number, then moves to line 40.

LINE 62: Sets up a condition for a guess higher than the correct response, then moves to line 40 to request another guess.

LINE 70: Requests user INPUT. The user is asked, "PLAY AGAIN?"

LINE 80: Sets up a condition for a "yes" response to "PLAY AGAIN?", then moves to line 10 to restart.

LINE 90: Sets up a condition for a non-"yes" response, then moves to line 100 to END the program.

LINE 100: ENDS the program.

If you're wondering how you might enhance *What's My Number?* even more, Chapter 10 ("Basic Graphics Tools") introduces some new ideas that can be incorporated into this program. Graphics increase the interest of almost any program. Even business programs can benefit greatly from graphic representation of data. More about this later.

Chapter 6

TAKE A GUESS

Take a Guess qualifies as a number game because only numeric variables are used; however, words, in the form of PRINT statements, are a big part of the game.

Here's the road map of *Take a Guess*:

1. A question is displayed on the screen, along with three possible answers.
2. The player selects an answer by pressing a single key.
3. After every question has been answered, the player's total number of correct responses is displayed.

Efficient Programming through Subroutines

Take a Guess uses subroutines to carry out functions that must be performed several times during the game. Note, in the listing included at the end of the chapter, that appropriate subroutines have been provided to follow both correct and incorrect player responses.

Lines 155 to 167 contain the five subroutines which are used with each question/answer sequence. Lines 170 to 260 contain game instructions. Lines 180, 220, and 260 are "pause loops." A pause loop is a FOR/NEXT loop which counts to a given number, then ends. Since this counting takes time, the loop functions as a delay. Pause

loops are frequently used when screen instructions must be displayed long enough to be read.

The main program routine is repeated in this game each time a new question is asked. This is the best possible programming practice. We use it here to avoid the need for alphanumeric variables, which we discuss later.

In lines 270-345 we see the question and answer routine. The question is posed in line 270, and the three possible answers come in lines 290-310. Lines 315-320 are perhaps the most important.

The subroutine at lines 162 and 163 directs the player to input an answer. The answer is stored in variable AN. If the player's answer corresponds to the correct answer, subroutines 164 and 165 follow. Subroutine 164 adds five points to the player's total. Subroutine 165 is a pause loop. Finally, the program jumps to the next question.

If the player chooses the wrong answer, the program "falls through" to line 320. Subroutine 167 gives the correct answer. After a pause loop in 345, the next question comes up. The same sequence is repeated in lines 350-445 and 450-530. The game is wound up in lines 540-550. The player's total score, stored as PTS, is displayed.

Additional programming efficiency could have been achieved by including a single subroutine to display questions and possible answers, but this was not done to avoid the use of alphanumeric variables, which will be discussed in Chapter 7.

Take a Guess

```
100 REM *****
110 REM *   TAKE A GUESS!   *
120 REM *  COPYRIGHT (C) 1983  *
130 REM *   BY WILLIAM RUPP   *
135 REM * THIS VERSION HAS SUB- *
138 REM *  ROUTINES (162-167)  *
140 REM *****
```

```

150 :
    PRINT "{CLR}{BLK}":POKE53281,1:POKE53
280,1
155 REM: 150 MAKES SCREEN & BORDER
    WHITE, CURSOR BLACK
160 GOTO 170
162 PRINT:PRINT"{DOWN 2}{RIGHT 3}{RVS} E
NTER 1, 2, OR 3, THEN PRESS RETURN ":
163 PRINT:INPUT"{RIGHT 9}";ANS$:RETURN
164 PTS=PTS+5:PRINT"{DOWN 2}{RIGHT 10}{R
VS} CORRECT {ROFF}":RETURN
165 FORI=1TO3000:NEXT:RETURN
167 PRINT"{DOWN 2}{RIGHT 3} -S O R R Y-
THE ANSWER IS";:RETURN
170 PRINT"{CLR}{DOWN 8}          HOW MUCH
DO YOU KNOW?"
175 PRINT"{DOWN 4}{RIGHT 6}{RVS} T A K E
A G U E S S "
180 FORI=1TO5000:NEXT: REM =PAUSE LOOP=
190 PRINT"{CLR}{DOWN 9}{RIGHT 7}YOU WILL
SEE A SERIES OF "
200 PRINT"{DOWN 2}{RIGHT 7}QUESTIONS, EA
CH FOLLOWED"
210 PRINT"{DOWN 2}{RIGHT 7}BY THREE POSS
IBLE ANSWERS."
220 FORI=1TO5000:NEXT: REM =PAUSE=
230 PRINT"{HOME}{DOWN 9}{RIGHT 7}TYPE TH
E NUMBER OF THE "
240 PRINT"{DOWN 2}{RIGHT 7}CORRECT ANSWE
R, THEN "
250 PRINT"{DOWN 2}{RIGHT 7}PRESS THE <RE
TURN> KEY. "
260 FORI=1TO4000:NEXT:REM =PAUSE=
270 PRINT"{CLR}{DOWN 4} WHICH OF THESE B
ECAME A STATE IN 1912?"
275 PRINT"-----
-----"
290 PRINT"{DOWN}          1. HAWAII
300 PRINT"{DOWN}          2. ARIZONA
310 PRINT"{DOWN}          3. UTAH
315 GOSUB 162:IF AN$="2"THEN GOSUB 164:G

```

```

OSUB165:GOTO 345:REM= CHECK ANSWER=
320 GOSUB 167:PRINT" 2"
345 FORI=1TO3000:NEXT: REM =PAUSE=
350 PRINT"{CLR}{DOWN 4} WHICH IS THE 'SH
OW ME' STATE?"
360 PRINT"-----
-----"
370 PRINT"{DOWN}          1. MAINE
380 PRINT"{DOWN}          2. MISSISSIPPI
390 PRINT"{DOWN}          3. MISSOURI
400 GOSUB 162: IF AN$="3" THEN GOSUB 164
:GOSUB 165:GOTO450
430 GOSUB 167:PRINT" 3"
445 FORI=1TO3000:NEXT:REM =PAUSE=
450 PRINT"{CLR}{DOWN 4} WHICH OF THESE C
OUNTRIES IS AN ISLAND"
460 PRINT"-----
-----"
470 PRINT"{DOWN}          1. PORTUGAL
480 PRINT"{DOWN}          2. PANAMA
490 PRINT"{DOWN}          3. CYPRESS
500 GOSUB 162: IF AN$="3" THEN GOSUB 164
:GOSUB 165:GOTO 540
510 GOSUB 167:PRINT" 3"
530 FORI=1TO3000:NEXT: REM =PAUSE=
540 PRINT"{CLR}{DOWN 5}{RIGHT 4}   E N
D   O F   G A M E "
545 PRINT"{DOWN 6}          YOU SCORED "PT
S" POINTS."
550 PRINT"{DOWN 7}":END

```

Chapter 7

GAMES WITH WORDS

The INPUT statement can be used to capture both numeric and string data. In both cases the following two forms may be used:

```
100 INPUT A$
200 INPUT "PLEASE ENTER NAME";N$
```

The syntax in line 100 uses no prompt (a message written to the screen which “prompts” the player to respond or act). The word INPUT is simply followed by the name of the variable or variables to be input. In line 200 we see the INPUT with a prompt, which is placed between quotes. In this case, a semicolon must appear between the close quotation mark and the variable name. While string variables were used here, INPUT statements using numeric variables would be similar.

String Variables

With this information fresh in mind, let's create a very simple game using string variables. This is a two-player game. Player one inputs a mystery word and clue (both of which utilize string variables), and player two tries to guess what the word is.

Here's the program:

```
100 PRINT "{CLR}":REM ** CLEAR SCREEN**
110 PRINT "{DOWN 5}":REM** GO DOWN FIVE
    LINES**
```

```

120 ?"===== GREAT CLUE WORD ====="
130 FOR I=1TO4000:NEXT:REM ** PAUSE **
140 PRINT {CLR}"
150 PRINT "FIRST PLAYER WILL NOW INPUT
    THE"
160 PRINT "MYSTERY WORD.  SECOND PLAYER
    WILL"
170 PRINT "PLEASE CLOSE EYES!"
180 PRINT "{RVS}PRESS ANY KEY TO CONTIN
UE{ROFF}":GETA$:IF A$= "" THEN 180
190 PRINT "{CLR}":INPUT "INPUT MYSTERY
    WORD";MYST$: REM ** SECRET WORD **
200 PRINT "{CLR}NOW INPUT CLUE:2 LINES
    OR LESS";CLUE$
210 PRINT "{CLR}"
220 PRINT "{DOWN 5}HERE IS THE CLUE FOR
    THE"
230 PRINT "SECOND PLAYER."
240 PRINT: PRINT {RVS}"CLUE${ROFF}":
REM ** CLUE DISPLAYED **
250 PRINT :INPUT "SECOND PLAYER, WHAT IS
    YOUR GUESS";ANS$:REM ** 2ND PLAYER **
260 IF ANS$=MYST$ THEN 300:REM ** IS THE
    ANSWER CORRECT? **
270 PRINT:PRINT "SORRY THAT'S NOT RIGHT.
    TRY AGAIN":
FOR I=1TO3000:NEXT
280 GOTO 220
300 PRINT "{CLR}{DOWN 5} THAT'S RIGHT!"
310 END

```

The road map of this game is very simple:

1. Player one enters a secret word and a clue sentence.
2. The clue sentence is displayed on the screen for player two to see.
3. Player two inputs a guess.
4. The Commodore 64 checks the answer; if it is correct, the message "THAT'S RIGHT!" appears on the screen. Otherwise the clue is displayed again, and player two has another chance to guess.

Comparing Strings

The key part of this program is line 260. In this line player two's guess (variable ANS\$) is compared to the secret word (variable MYST\$). If the two are identical, then a jump is made to line 300, where the "RIGHT" message is displayed. If the two variables are not identical, the flow of the program drops to the next line (line 270), where the "NOT RIGHT" routine begins. The end of this routine causes a jump back to line 240, where the clue is displayed again.

We want to stress that when two string variables are compared (as in line 260), the variables must be identical in *every* respect or your Commodore will consider them to be different. For instance, "ocean" and "OCEAN" are different because the first is written in lowercase letters while the second has all caps. Consider this pair: "moonlight" and " moonlight". They too are different because the second string begins with a space. Look at that pair again if you do not see the difference.

Remember, everything contained within quotes must be considered. Therefore, in the "moonlight" example, both words are identical, but since the second string begins with a space, the strings are different. Your Commodore 64 will compare both strings character by character, space by space. If there is any difference found in the comparisons, the strings will be judged not equal.

A more advanced technique to be studied later will enable you to evaluate a portion of a string rather than the entire string. For the time being, however, we'll utilize the all-or-nothing string comparison we've been discussing.

A Short Review

Before going on, let's review what we've learned about string variables:

1. Any character can be contained within the quotation marks of an alphanumeric (string) variable.
2. String variables cannot be computed or used in any mathematical calculation.
3. String variables can be defined either by assigning values in the program or by allowing a player to input data from the keyboard.
4. `INPUT` statements can include screen prompts (instructions enclosed in quotation marks "prompting" the player to respond or act) which come after the word `INPUT` and before the variable name. A semicolon must separate the prompt from the variable name.

Chapter 8

LIGHTS! CAMERA! ACTION!

Alphanumeric, or string, variables are the most versatile variables your Commodore 64 understands. To illustrate string variables, we turn to a quiz game which should please old movie fans (old movies or old fans, whichever you prefer).

LIGHTS! CAMERA! ACTION! benefits greatly from the use of alphanumeric variables (as well as arrays and flags). The road map of this game is as follows:

1. The player chooses one of three categories of film questions.
2. If the category has already been chosen, the player must choose again.
3. A question is displayed, and the player must enter an answer.
4. The player is told whether the answer is correct or incorrect.
5. After five questions, the player may choose another category.
6. When all categories have been played, the player's final score is displayed.

Using Subroutines and Arrays

All subroutines but one are found near the beginning of the program (listed at the end of this chapter). This is a good practice, since the Commodore 64 must go to the beginning of the program and start counting until it finds the subroutine in question. Obviously, if the subroutine is near the beginning, time is saved.

The one subroutine which is not near the beginning of the program is located in lines 1180-1480. In these lines the questions and correct answers are defined. Six arrays are used. Two arrays are defined per category—one for questions, one for answers.

LIGHTS! CAMERA! ACTION! uses the simple method of defining each element of an array with an equal sign. A\$(1) = "Who was the Duke?", for instance. In *Goal To Go!* (Chapter 31) you will see how FOR/NEXT loops read DATA statements to accomplish the same thing more efficiently. Arrays A\$(5), B\$(5), and C\$(5) contain the questions, while arrays AA\$(5), BB\$(5), and CC\$(5) contain the answers.

The next routine (380-440) draws a border of dollar signs around the screen. Notice line 410. TAB(38) places the dollar sign at the end of each line to form the right-hand border.

The title of the game is drawn in lines 450-500. The subroutine at 370, which is a pause loop, limits the time that the title is displayed on the screen. This same pause loop is also used elsewhere in the program.

Next, the screen is cleared and the border drawn again, this time followed by the subtitle and the keyboard-graphics picture of a movie camera. Lines 650-710 allow the player to select the next category of question. CAT\$ is used to store the player's input. String variables offer advantages over numeric variables (as in *Take a Guess*). With a string variable it is easier to handle illegal entries. An example of an illegal entry is when the player enters a letter where a number is requested.

In lines 720-800, the player's response causes a branch to the appropriate part of the program. Let's use 750 to show how the player's choice is analyzed:

```
750 IF CAT$="1" AND C1=0 THEN B10
```

Flags: Binary Switches

Perhaps the most important new tool for your construction kit which we explain in this chapter is the flag. A flag is simply a binary type numeric variable which acts much like an ON/OFF switch. Think of a flag as a traffic signal with only two colors: red and green. The status of a flag (yes/no, ON/OFF, 0/1, etc.) can be used to determine whether some action has already taken place or whether an action should take place in the future. Flags help programmers make sure that actions are executed only at the proper time. As soon as the action has occurred, you can set a flag variable to prevent that action from recurring until the flag is changed.

Since CAT\$ is a string (alphanumeric) variable, quotation marks must be used. C1 is a flag. Basically, line 750 says that if the player chose category one (CAT\$="1") and if flag C1 is 0, then it's okay to go on with category one's questions. If CAT\$ does not equal "1", or if the flag variable C1 equals something other than 0, then the program goes on to the next line.

The logical statements AND and OR are very powerful tools which you need to understand. In both cases you are trying to determine if something is true or false. With OR, you want to know if either one of two possibilities is true. For instance:

```
100 A=7
110 B=3
120 IF A> 5 OR B > 5 THEN X=1:GOTO 200
130 X=0:GOTO 210
```

In this example, X equals 1, since A is in fact greater than 5. Only one of the two propositions must be true; both could be true, but one is enough.

On the other hand, AND requires that both propositions are true. Change line 120 in the example to this:

```
120 IF A> 5 AND B >5 THEN X=1:GOTO 200
```

Here, only one proposition is true (A is greater than 5, but not B), so the entire statement is false. X equals 0.

Remember, in an AND statement, both conditions must be true for the following result to take place.

Assuming that the player has never tried category one (i.e., C1=0), we move to lines 810-880. The basic routine is the same for all categories. First, the title of the category is displayed. Line 830 makes flag variable C1 equal to 1 (C1=1). This is called setting the flag. The next time the player chooses a category, the second half of line 750 will be false (C1 will not equal 0). This way the player can only try a category once. Thereafter, the set flag (C1=1) prevents a repeat of that category.

General-Purpose Arrays

Lines 840 and 850 fill the general-purpose arrays Q\$(5) and QQ\$(5) with the five questions and five answers for the category in question. Line 860 signals that question one is next (Q=1) and that each question in this group is worth 5 points (P=5).

The most important routine is found in lines 200-310. Here the value of each question is stated, then the five questions are asked. There are three variables which should be understood: Q, Q\$(Q), and QQ\$(Q). Remember that the five questions (contained in A\$(5)) and their answers (contained in AA\$(5)) were stored in arrays Q\$(5) and QQ\$(5), respectively. Variable Q indicates the number of the next question. Q\$(Q), therefore, will always be the new question. The player's answer (AN\$) is compared with QQ\$(Q). If Q equals 2, then the question will be Q\$(2), and the player's response will be compared with QQ\$(2).

If the answer is not correct, that fact is announced to the player in line 280, and MISSES is incremented by 1. In this game, three wrong answers disqualify the player. If the variable MISSES equals 3, then the program branches to line 1030. If the player is correct, the running total of correct points is added to the value of P (the value of each correct answer in that category).

No FOR/NEXT loop is used to run through the five questions. Instead, after each question has been answered, variable Q is incremented by 1, then checked to see if Q is greater than 6. If it is, then all questions have been asked, and there is a return to line 870. Line 870 uses two ANDs to determine whether all categories have been completed (which is signalled only if C1, C2, and C3 all equal 1). If that is so, then we jump to 1080, where the player is told how many points have been earned.

If at least one category has not been tried (determined if at least one of the flags—C1, C2, and C3—equals 0), then it's back to line 650, so a new category may be chosen.

Summary

LIGHTS! CAMERA! ACTION! contains a number of important game ideas. Arrays have been used extensively. The practice of redefining a set of general-purpose arrays (Q\$(5) and QQ\$(5)) for each category of questions is not the only way this game could have been written. Separate routines calling for A\$(), etc., would also have worked.

The next chapter will contain an example of a Commodore 64 game which uses both numeric and alphanumeric (string) variables in a slightly different way.

LIGHTS! CAMERA! ACTION!

```
100 REM *****
110 REM * LIGHTS! CAMERA! ACTION! *
120 REM *   A MOVIE QUIZ           *
130 REM * COPYRIGHT (C) 1983 BY   *
140 REM *   WILLIAM L. RUPP      *
150 REM *****
160 :
170 PRINT "{CLR}"
180 POKE53281,1:POKE53280,1:PRINT "{BLK}"
:REM   =WHITE SCREEN, BLACK CURSOR=
```

```

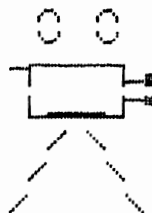
190 GOTO630
195 REM *****
200 PRINT"{CLR}{DOWN 5}"H$:REM =TITLE OF
    CATEGORY=
210 PRINT"{DOWN 3} EACH QUESTION WILL B
E WORTH "P"POINTS"
220 GOSUB 370:REM =PAUSE=
230 Q=1:REM =QUESTION COUNTER=
240 PRINT"{CLR}{DOWN 4} {RVS} QUESTION N
UMBER {ROFF}"Q
250 PRINT:PRINT" "Q$(Q)
260 PRINT:PRINT" ENTER YOUR ANSWER, THEN
PRESS {RVS}RETURN{ROFF}":INPUT ANS$
270 IF ANS$=QQ$(Q)THEN 330
280 PRINT"{CLR}{DOWN 5} SORRY, THE A
NSWER IS ":
290 PRINT:PRINT" {RVS}"QQ$(Q)"{ROFF
}":GOSUB370
300 MISSES=MISSES+1:IF MISSES=3THEN1030
310 Q=Q+1:IFQ>5 THEN RETURN
320 GOTO 240
330 PRINT"{CLR}{DOWN 6} THAT'S RIG
HT!!"
340 GOSUB 370:PTS=PTS+P:Q=Q+1:IFQ>5 AND
C1=1 AND C2=1 AND C3=1 THEN 1070
350 IF Q>5 THEN RETURN
360 GOTO240
370 FOR PA =1TO4000:NEXT:RETURN:REM
RUN =PAUSE LOOP=
380 PRINT"{CLR}":REM =$ BORDER ROUTINE=
390 PRINT"{UP}";:FORI=1TO39:PRINT"$";:NE
XT
400 PRINT
410 FORI=1TO21:PRINTTAB(38)"$":NEXT
420 PRINT"{HOME}":FORI=1TO20:PRINT"$":NE
XT
430 FORI=1TO39:PRINT"$";:NEXT
440 RETURN
450 PRINT"{HOME}"
460 PRINT"{DOWN 6}$ {RVS} L I G H
T S !!{ROFF}"

```

```

470 PRINT "{DOWN}$"           {RVS} C A M
E R A !!
480 PRINT "{DOWN}$"           {RVS} A
C T I O N !!!
490 GOSUB 370
500 RETURN
510 PRINT "{HOME}{DOWN 6}$"    A MOVIE
GUESSING GAME"
520 PRINT"$"   -----
--"
530 PRINT"
540 PRINT"$
550 PRINT"$
560 PRINT"$
570 PRINT"$
580 PRINT"$
590 PRINT"$
600 PRINT"$
610 PRINT"$
620 RETURN
630 GOSUB1180: GOSUB 380:GOSUB450:GOSUB
380:GOSUB510
640 GOSUB 370
650 PRINT"{CLR}"
660 PRINT"{CLR}{DOWN 3}"      {RVS} WHI
CH CATEGORY WOULD{ROFF}"
670 PRINT"{DOWN 2}"          {RVS} YOU LI
KE TO TRY? {ROFF}"
680 PRINT"{DOWN 2}"          1. FAMOUS STARS
& DIRECTORS"
690 PRINT"{DOWN 2}"          2. FAVORITE CLA
SSIC MOVIES"
700 PRINT"{DOWN 2}"          3. ALL-TIME MOV
IE TRIVIA"
710 PRINT"{DOWN 2} ENTER 1,2, OR 3, THE
N PRESS {RVS}RETURN{ROFF}" :PRINT"{RIGHT
2}";:INPUT CAT$
720 IF CAT$="1"THEN 750
730 IF CAT$="2"THEN 770
740 IF CAT$="3"THEN 790
750 IF CAT$="1"AND C1=0 THEN 810

```



```

760 GOTO1150
770 IF CAT$="2"AND C2=0 THEN 890
780 GOTO1150
790 IF CAT$="3"AND C3=0 THEN 960
800 GOTO1150
810 REM: =CATEGORY 1-STARS & DIRECTORS
820 H$="{RVS}FAMOUS STARS & DIRECTORS{ROFF}"
830 C1=1
840 FORI=1TO5:Q$(I)=A$(I):NEXT
850 FORI=1TO5:QQ$(I)=AA$(I):NEXT
860 Q=1:P=5:GOSUB200
870 IFC1=1ANDC2=1 AND C3=1 THEN 1150
880 GOTO 650
890 REM: =CATEGORY 2- CLASSIC MOVIES
900 C2=1:H$="{RVS} CLASSIC MOVIES{ROFF}"
910 FORI=1TO5:Q$(I)=B$(I):NEXT
920 FORI=1TO5:QQ$(I)=BB$(I):NEXT
930 Q=1:P=10:GOSUB200
940 IFC1=1ANDC2=1 AND C3=1 THEN 1150
950 GOTO 650
960 REM: =CATEGORY 3- MOVIE TRIVIA
970 C3=1:H$="{RVS}MOVIE TRIVIA{ROFF}"
980 FORI=1TO5:Q$(I)=C$(I):NEXT
990 FORI=1TO5:QQ$(I)=CC$(I):NEXT
1000 Q=1:P=15:GOSUB200
1010 IFC1=1ANDC2=1 AND C3=1 THEN 1150
1020 GOTO 650
1030 PRINT"{CLR}{DOWN 5} THAT IS YOUR
THIRD MISS."
1040 PRINT"{DOWN} YOU HAVE FINISHED WI
TH A TOTAL"
1050 PRINT"{DOWN} OF {RVS}"PT"{ROFF
F}POINTS."
1060 END
1070 GOSUB 380
1080 PRINT"{HOME}{DOWN 6}$ YOU H
AVE COMPLETED"
1090 PRINT"{DOWN}$ ALL CATEGORIES WI

```



```

TH A FINAL"
1100 PRINT"{DOWN}$          SCORE OF {RVS
}"PTS"{ROFF} POINTS."
1110 PRINT"{DOWN 3}$      {RVS}
          {ROFF}"
1120 PRINT"$          {RVS} SEE YOU AT THE
MOVIES! {ROFF}"
1130 PRINT"$          {RVS}
          {ROFF}"
1140 END
1150 PRINT"{CLR}{DOWN 4}      SORRY, YOU
HAVE ALREADY"
1160 PRINT"{DOWN 2}      TRIED THIS CA
TEGORY"
1170 GOSUB 370:GOTO 650
1180 A$(1)=" WHO WAS THE DUKE? "
1190 A$(2)=" WHO BECAME A STAR IN {RVS}T
HE GRADUATE{ROFF}?"
1200 A$(3)=" WHO DIRECTED AND STARRED IN
{RVS}ANNIE HALL{ROFF}?"
1210 A$(4)=" WHO PLAYED DIRTY HARRY?"
1220 A$(5)=" WHO DIRECTED {RVS}THE BIRDS
{ROFF} AND {RVS}PSYCHO{ROFF}?"
1230 AA$(1)="JOHN WAYNE"
1240 AA$(2)="DUSTIN HOFFMAN"
1250 AA$(3)="WOODY ALLEN"
1260 AA$(4)="CLINT EASTWOOD"
1270 AA$(5)="ALFRED HITCHCOCK"
1280 B$(1)=" WHAT FILM TOLD THE STORY OF
A ROMAN CHARIOTEER?"
1290 B$(2)=" COMPLETE THIS FILM TITLE:{R
VS}FROM HERE TO...{ROFF}"
1300 B$(3)=" {RVS}THE MISFITS{ROFF} WAS
THE LAST FILM OF WHAT ACTRESS?"
1310 B$(4)=" NAME THE FILM WHICH PORTRAY
ED THE 1924 OLYMPICS"
1320 B$(5)=" WHAT FILM MADE ALAN LADD A
STAR?"
1330 BB$(1)= "BEN HUR"
1340 BB$(2)= "ETERNITY"
1350 BB$(3)="MARILYN MONROE"

```

1360 BB\$(4)="CHARIOTS OF FIRE"
 1370 BB\$(5)="THIS GUN FOR HIRE"
 1380 C\$(1)="IN WHAT FILM DID SAM PLAY IT
 AGAIN?"
 1390 C\$(2)="IN WHAT FILM DID WOODY ALLEN
 TRAVEL TO THE FUTURE?"
 1400 C\$(3)="WHAT MOVIE STUDIO DID HOWARD
 HUGHES OWN?"
 1410 C\$(4)="WHAT WAS THE NAME OF BOGIE'S
 YACHT?"
 1420 C\$(5)="HOW MANY OSCARS HAVE RICHARD
 BURTON AND PETER O'TOOLE WON?"
 1430 CC\$(1)="CASABLANCA"
 1440 CC\$(2)="SLEEPER"
 1450 CC\$(3)="RKO RADIO PICTURES"
 1460 CC\$(4)="SANTANA"
 1470 CC\$(5)="NONE"
 1480 RETURN

The following lines show the ASCII equivalents (see Appendix F)
 of the keyboard graphics characters which comprise the illustration
 in *LIGHTS! CAMERA! ACTION!*

```

540 PRINT"$           {#213}{#201} {#
213}{#201}
550 PRINT"$           {#202}{#203} {#
202}{#203}
560 PRINT"$           {#196}{#207}{#18
3 3}{#208}{#198}{#187}
570 PRINT"$           {#204}{#175 3}{
#186}{#196}{#190}
580 PRINT"$           {#206} {#205}
590 PRINT"$           {#206}    {#205}

600 PRINT"$           {#206}          {#205
}
610 PRINT"$
  
```

Chapter 9

ASK THE WIZARD

In *Ask the Wizard* you will learn about a new kind of guessing game—the fortune telling game. This game is interactive, much like *LIGHTS! CAMERA! ACTION!*; however, the answers in *Ask the Wizard* will appear to be determined by fate.

Many of you have seen the “fortune telling lady” at a penny arcade. Usually you insert a coin in the machine, a mechanical doll moves its hand—apparently inserting a slip of paper in the return slot—and your fortune emerges. Along the same lines, you may like to read about your lucky number, or maybe you like to read the contents of fortune cookies.

In *Ask the Wizard*, the player will be presented with a selection of questions which may be asked of the wizard. The player chooses, and the wizard asks the player to pick a number from 1 to 16. After the player picks a number, the wizard displays the answer to the question on the screen. The player is then invited to continue playing or to stop.

The computer’s response is determined by the question asked and the number selected. Random numbers are not used in this game; rather, numerology is the model. The number chosen by the player supposedly has a magical effect which determines the answer. Fate, then, plays a role in *Ask the Wizard*, and though the game is pretty simple, it can be quite interesting to discover what strange or meaningful answers turn up.

Using our trusty road map, let’s get an overview of the program structure. First, we must give instructions to the player, then offer options for the player to select. Next, we need to set up

subroutines which will return answers to the questions. Finally, we must end the game.

Subroutines: GOSUB and RETURN

Subroutines allow the program to move to a specified line number to receive new instructions, reading through the lines until the RETURN command is encountered, and then RETURN to the line where the GOSUB was found. When subroutines are set up in an orderly fashion, the result is a structured program which is easily understood by another programmer. Subroutines allow complicated and repetitive tasks to appear only once in a program. When a subroutine is needed, it can be called from anywhere in the program. This eliminates the need for repetitive code which takes up unnecessary space in memory.

Let's look at an example of a subroutine used many times in the course of a program.

```
1 PRINTCHR$(147):POKE53281,1:PRINTCHR$(144)

100 PRINT"BELOW IS A SERIES OF QUESTIONS
. YOU"
105 PRINT"MUST CHOOSE A SYMBOL FROM THE
CHART"
110 PRINT"IN ORDER TO RECEIVE THE CORREC
T ANSWER."
115 PRINT"WHEN YOU ARE READY PRESS THE N
UMBER"
120 PRINT"YOU CHOSE.":PRINT:PRINT
130 PRINT:PRINT"FIRST- CHOOSE A QUESTION
TO ASK.":PRINT:PRINT
140 PRINT" 1.CAN I MAKE THE ONE I LOVE L
OVE ME?"
142 PRINT" 2.IS THE ONE I LOVE FAITHFUL
TO ME?"
144 PRINT" 3.WILL MY LOVER MAKE ME HAPPY
?"
```

```

146 PRINT" 4.WILL THE WISH I MAKE NOW CO
ME TRUE?"
148 PRINT" 5.WILL I HAVE MANY AMOROUS AD
VENTURES?"
150 PRINT:PRINT:INPUT" WHICH? ";Q$
155 IF VAL(Q$)>5 THEN GOSUB 5500:GOTO 10
0
156 IF VAL(Q$)<1 THEN GOSUB 5500:GOTO 10
0

```

Lines 1-148 clear the screen, then display the instructions for the player along with the questions the player may ask the wizard. By this time, the PRINT statement should be quite familiar. Line 150 asks for player input, defining string variable Q\$. Line 155 uses the VAL function. (Remember, error trapping is an essential part of sound programming technique, and it is important to use error-trapping routines whenever they are applicable.)

Look at subroutine 5500:

```

5500 REM ERROR TRAP
5510 PRINT"THE CHOICE IS one to five.":
RETURN

```

The program can go to this subroutine whenever an invalid response to the question is entered. Line 156 repeats the same principle to trap responses with a value less than one.

Line 160 shows another subroutine which can be accessed continually during the game.

```

160 PRINT CHR$(147):GOSUB 200:GOTO165

```

This sets up a direction for the program: moving to line 200, and upon RETURNing, GOingTO the next line (165).

```

165 IF Q$="1" THEN GOSUB 1000
170 IF Q$="2" THEN GOSUB 2000
175 IF Q$="3" THEN GOSUB 3000
180 IF Q$="4" THEN GOSUB 4000
185 IF Q$="5" THEN GOSUB 5000
190 PRINT:PRINT:INPUT"WANT TO ASK
ANOTHER ";X$
192 IF LEFT$(X$,1)<>"Y" THEN GOTO
6000
194 GOTO 100
200 REM SCREEN DISPLAY OF NUMBERS
210 PRINT"1, 2, 3, 4, 5, 6, 7, 8, 9"
220 PRINT:PRINT"10, 11, 12, 13, 14,
15, 16"
390 PRINT
400 INPUT"WHICH NUMBER DO YOU
CHOOSE";A$:PRINT
410 RETURN

```

Since line 160 directs the program to GOSUB 200, it is important to take note of the lines in this subroutine. As mentioned in the REMark statement, the subroutine displays numeric options on-screen then RETURNS to line 160. This subroutine is called each time the player wants to play again. Obviously, this saves a lot of code which would be generated if one had to repeat program lines in anticipation of the number of times a player might want to play.

Now, let's take a look at lines 165 to 194. Here, we are looking for valid codes in response to our invitation for the player to choose one of five questions. Each response (1-5) has a separate subroutine set aside for it. (For clarity, the first lines of the subroutines are incremented by 1000. Of course, this is not mandatory, but it does help the programmer remember which routines were designated for which functions.)

Now look at subroutine 1000:

```

1000 REM ANSWER SUBROUTINE
1010 IF A$="1" THEN PRINT" NOTHING. YOU AR
E ALREADY ADORED. ":RETURN

```

```

1020 IF A$="2"THEN PRINT"HIDE THE PAST &
DO BETTER.":RETURN
1030 IF A$="3"THEN PRINT" CHANGE YOUR HA
BITS.":RETURN
1040 IF A$="4"THEN PRINT" BECOME RICH AN
D YOU WILL BE LOVED.":RETURN
1050 IF A$="5"THEN PRINT" CRY LESS AND L
AUGH MORE." :RETURN
1060 IF A$="6"THEN PRINT"LOSE THE QUALIT
Y OF WHICH YOU HAVE TOO MUCH.":RETURN
1070 IF A$="7"THEN PRINT" DO EXACTLY AS
YOUR LOVER WISHES.":RETURN
1080 IF A$="8"THEN PRINT" BE MORE AMIABL
E WHEN ALONE TOGETHER.":RETURN
1090 IF A$="9"THEN PRINT" SUFFERING IN S
ILENCE IS A GREAT VIRTUE.":RETURN
1100 IF A$="10"THEN PRINT" PLAY HARD TO
GET. YOU WILL BE VALUED." :RETURN
1110 IF A$="11"THEN PRINT" BE FAITHFUL &
DON'T COMPLAIN.":RETURN
1120 IF A$="12"THEN PRINT" THOUGH IT IS
DIFFICULT BE DISCREET.":RETURN
1130 IF A$="13"THEN PRINT" BE CALM IN YO
UR LOVER'S COMPANY.":RETURN
1140 IF A$="14"THEN PRINT" BE FORGIVING
AND TOLERANT.":RETURN
1150 IF A$="15"THEN PRINT" BE ALWAYS THE
SAME.":RETURN
1160 IF A$="16"THEN PRINT"DO NOT REPROAC
H YOUR LOVER WHEN HE/SHE IS WRONG."
1170 RETURN

```

Each line identifies one of the 16 possible codes for A\$ as a different response to question one. If the player selects number one from the screen display in line 210, then "code 1" of A\$ becomes "NOTHING. YOU ARE ALREADY ADORED." Actually, each line of subroutine 1000 is a separate subroutine. If the condition stipulated is met, then the end of the subroutine is signified by the RETURN after the colon on the same line as the conditional statement. If the player chooses number 16 for question one, then the program goes to each line following line 1000 until it reads line

1160. But as soon as the selected number code for A\$ is located, the program RETURNS to the GOSUB line (line 165 for question one).

From this example, it is a relatively simple matter to construct similar sets of responses for each of the remaining questions posed in lines 165-185. Check the listing at the end of this chapter if you are stuck for ideas.

Ending the Game with LEFT\$

Lines 190-192 prompt the player to try again and, if the response is not YES or Y, end the game. Line 194 restarts the game if the response is Y, using the LEFT\$ function to pick out the first character entered.

Finally, the END routine is at lines 6000-6010.

Play the game for a few rounds and try thinking of ways to improve or enhance it. Later, in Chapter 14, you'll find an upgraded version of *Ask the Wizard*; the enhancements are in the area of graphics. In the meantime, see if you can make *Ask the Wizard* more entertaining by adding words.

Ask the Wizard

```
1 PRINTCHR$(147):POKE53281,1:PRINTCHR$(144)
10 REM WIZARD
20 PRINT:PRINT
79 PRINT"***** T H E    W I Z A R D *****"
"
90 FOR PAUSE=1 TO 2000:NEXT PAUSE:PRINTCHR$(147)
100 PRINT"BELOW IS A SERIES OF QUESTIONS
. YOU"
105 PRINT"MUST CHOOSE A SYMBOL FROM THE
CHART"
```



```

110 PRINT"IN ORDER TO RECEIVE THE CORRECT ANSWER."
115 PRINT"WHEN YOU ARE READY PRESS THE NUMBER"
120 PRINT"YOU CHOSE.":PRINT:PRINT
130 PRINT:PRINT"FIRST- CHOOSE A QUESTION TO ASK.":PRINT:PRINT
140 PRINT" 1.CAN I MAKE THE ONE I LOVE LOVE ME?"
142 PRINT" 2.IS THE ONE I LOVE FAITHFUL TO ME?"
144 PRINT" 3.WILL MY LOVER MAKE ME HAPPY?"
146 PRINT" 4.WILL THE WISH I MAKE NOW COME TRUE?"
148 PRINT" 5.WILL I HAVE MANY AMOROUS ADVENTURES?"
150 PRINT:PRINT:INPUT" WHICH? ";Q$
155 IF VAL(Q$)>5 THEN GOSUB 5500:GOTO 100
156 IF VAL(Q$)<1 THEN GOSUB 5500:GOTO 100
160 PRINTCHR$(147):GOSUB 200:GOTO 165
165 IF Q$="1"THEN GOSUB 1000
170 IF Q$="2"THEN GOSUB 2000
175 IF Q$="3"THEN GOSUB 3000
180 IF Q$="4"THEN GOSUB 4000
185 IF Q$="5"THEN GOSUB 5000
190 PRINT:PRINT:INPUT"WANT TO ASK ANOTHER ";X$
192 IF LEFT$(X$,1)<>"Y"THEN GOTO 6000
194 GOTO 100
200 REM SCREEN DISPLAY OF NUMBERS
210 PRINT:PRINT"1, 2, 3, 4, 5, 6, 7, 8, 9"
220 PRINT"10, 11, 12, 13, 14, 15, 16"
230 PRINT
400 INPUT"WHICH NUMBER DO YOU CHOOSE ";A$:PRINT:
410 RETURN
1000 REM ANSWER SUBROUTINE

```

1010 IF A\$="1"THENPRINT" NOTHING. YOU ARE
ALREADY ADORED.":RETURN
1020 IF A\$="2"THEN PRINT"HIDE THE PAST &
DO BETTER.":RETURN
1030 IF A\$="3"THEN PRINT" CHANGE YOUR HA
BITS.":RETURN
1040 IF A\$="4"THEN PRINT" BECOME RICH AN
D YOU WILL BE LOVED.":RETURN
1050 IF A\$="5"THEN PRINT" CRY LESS AND L
AUGH MORE." :RETURN
1060 IF A\$="6"THEN PRINT"LOSE THE QUALIT
Y OF WHICH YOU HAVE TOO MUCH.":RETURN
1070 IF A\$="7"THEN PRINT" DO EXACTLY AS
YOUR LOVER WISHES.":RETURN
1080 IF A\$="8"THEN PRINT" BE MORE AMIABL
E WHEN ALONE TOGETHER.":RETURN
1090 IF A\$="9"THEN PRINT" SUFFERING IN S
ILENCE IS A GREAT VIRTUE.":RETURN
1100 IF A\$="10"THEN PRINT" PLAY HARD TO
GET. YOU WILL BE VALUED." :RETURN
1110 IF A\$="11"THEN PRINT" BE FAITHFUL &
DON'T COMPLAIN.":RETURN
1120 IF A\$="12"THEN PRINT" THOUGH IT IS
DIFFICULT BE DISCREET.":RETURN
1130 IF A\$="13"THEN PRINT" BE CALM IN YO
UR LOVER'S COMPANY.":RETURN
1140 IF A\$="14"THEN PRINT" BE FORGIVING
AND TOLERANT.":RETURN
1150 IF A\$="15"THEN PRINT" BE ALWAYS THE
SAME.":RETURN
1160 IF A\$="16"THEN PRINT"DO NOT REPROAC
H YOUR LOVER WHEN HE/SHE IS WRONG."
1170 RETURN
2000 REM ANSWER SUB
2010 IF A\$="1"THEN PRINT"SOMETIMES FOR A
T LEAST 2 DAYS.":RETURN
2020 IF A\$="2"THEN PRINT" YES.":RETURN
2030 IF A\$="3"THEN PRINT" WISHES TO;BUT
CANNOT.":RETURN
2040 IF A\$="4"THEN PRINT"THEY SAY YES. T
HE WIZARD SAYS NO.":RETURN

```

2050 IF A$="5"THEN PRINT"A QUALITY YOU H
AVE CAUSED TO BE LOST.":RETURN
2060 IF A$="6"THEN PRINT"LOSS OF LOVE--L
OSS OF FAITHFULNESS.":RETURN
2070 IF A$="7"THEN PRINT"ALWAYS HAS BEEN
& ALWAYS WILL BE.":RETURN
2080 IF A$="8"THEN PRINT"HE/SHE IS FAITH
FUL TO PLEASURE.":RETURN
2090 IF A$="9"THEN PRINT"CAN'T SAY MUCH
ABOUT IT.":RETURN
2100 IF A$="10"THEN PRINT"NOT NOW--BUT MA
YBE IN THE FUTURE.":RETURN
2110 IF A$="11"THEN PRINT"HAS BEEN BUT M
AYBE NOT AGAIN.":RETURN
2120 IF A$="12"THEN PRINT"YOU GET THE VA
LUE OF YOUR MONEY.":RETURN
2130 IF A$="13"THEN PRINT"AS MUCH AS YOU
ARE TO HIM/HER.":RETURN
2140 IF A$="14"THEN PRINT"YES. UNABLE TO
DO OTHERWISE.":RETURN
2150 IF A$="15"THEN PRINT"HE/SHE LOVES Y
OU TOO MUCH.":RETURN
2160 IF A$="16"THEN PRINT"ASK AND BELIEV
E WHAT YOU HEAR.":RETURN
3000 REM ANSWER 3 SUBROUTINE
3010 IF A$="1" THEN PRINT"FOR A COUPLE O
F DAYS MAYBE.":RETURN
3020 IF A$="2" THEN PRINT"NOT AS MUCH AS
YOU DESERVE.":RETURN
3030 IF A$="3" THEN PRINT"MUCH MORE THAN
YOU THINK.":RETURN
3040 IF A$="4" THEN PRINT"PARADISE! PARA
DISE!":RETURN
3050 IF A$="5" THEN PRINT"YES. IF LOVE
IS ENOUGH FOR YOU.":RETURN
3060 IF A$="6" THEN PRINT"YES. IF YOU A
RE CONTENT WITH LITTLE.":RETURN
3070 IF A$="7" THEN PRINT"HAPPINESS IS N
OT FOR YOU.":RETURN
3080 IF A$="8" THEN PRINT"HOME WILL GIVE
A TASTE OF HELL.":RETURN

```

```

3090 IF A$="9" THEN PRINT"AFTER THE AGE
OF FORTY ALL IS WELL.":RETURN
3100 IF A$="10" THEN PRINT"ALL THE TIME
AND EVERYWHERE.":RETURN
3110 IF A$="11" THEN PRINT"MORNING;NOON
AND NIGHT.":RETURN
3120 IF A$="12" THEN PRINT"MORE HAPPINES
S THAN WISDOM.":RETURN
3130 IF A$="13" THEN PRINT"YOUR LOVER CA
RES MORE FOR SELF THAN YOU.":RETURN
3140 IF A$="14" THEN PRINT"ONCE IN A WHI
LE.":RETURN
3150 IF A$="15" THEN PRINT"DEPENDS UPON
WHAT HAPPENS LATER.":RETURN
3160 IF A$="16" THEN PRINT"TOO MUCH TOO
SOON.":RETURN
4000 REM ANSWER 4 SUBROUTINE
4010 IF A$="1" THEN PRINT"WHEN YOU LEAST
EXPECT IT.":RETURN
4020 IF A$="2" THEN PRINT"IMPOSSIBLE DRE
AM.":RETURN
4030 IF A$="3" THEN PRINT"NOT FOR A LONG
TIME.":RETURN
4040 IF A$="4" THEN PRINT"PARTLY BUT NEV
ER COMPLETELY.":RETURN
4050 IF A$="5" THEN PRINT"YOU SHOULD NOT
WISH FOR THIS.":RETURN
4060 IF A$="6" THEN PRINT"IT WILL BE SOO
N OR NEVER.":RETURN
4070 IF A$="7" THEN PRINT"YES, AND YOU W
ILL WISH IT DID NOT.":RETURN
4080 IF A$="8" THEN PRINT"NEVER, UNTIL Y
OU CHANGE WHAT YOU WISH.":RETURN
4090 IF A$="9" THEN PRINT"YOU WILL BE RE
WARDED AFTER A LONG WAIT.":RETURN
4100 IF A$="10" THEN PRINT"GRADUALLY OVE
R TIME.":RETURN
4110 IF A$="11" THEN PRINT"ONLY IF YOU B
ECOME LESS SELF CENTERED.":RETURN
4120 IF A$="12" THEN PRINT"SOONER, RATHE
R THAN LATER.":RETURN

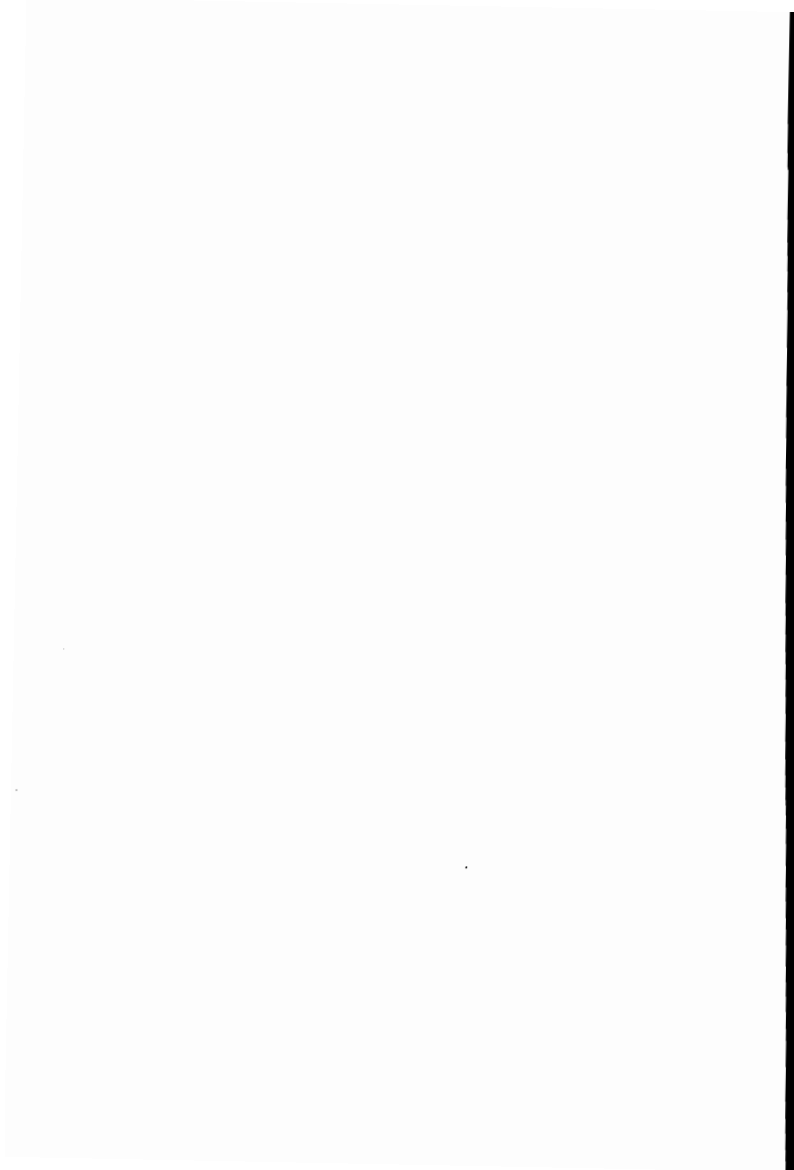
```

```
4130 IF A$="13" THEN PRINT"NO. THROUGH N
O FAULT OF YOURS.":RETURN
4140 IF A$="14" THEN PRINT"YES. BUT YOU
WILL NO LONGER WANT IT.":RETURN
4150 IF A$="15" THEN PRINT"THE FATE IS H
ANGING IN THE BALANCE.":RETURN
4160 IF A$="16" THEN PRINT"IT'S ENTIRELY
UP TO YOU.":RETURN
5000 REM ANSWER 5 SUBROUTINE
5010 IF A$="1" THEN PRINT"SO MANY LOVERS
- SO LITTLE TIME.":RETURN
5020 IF A$="2" THEN PRINT"YES, BUT FEW H
APPY ENDINGS.":RETURN
5030 IF A$="3" THEN PRINT"YES, AND ALL W
ILL END WELL.":RETURN
5040 IF A$="4" THEN PRINT"YES, BUT ONLY
AFTER NIGHTFALL.":RETURN
5050 IF A$="5" THEN PRINT"YES, BUT ONLY
BEFORE DAWN":RETURN
5060 IF A$="6" THEN PRINT"WHEN THEY ARE
4, TIME TO GET MORE.":RETURN
5070 IF A$="7" THEN PRINT"THE PATH OF LO
VE IS NOT ROSE STREWN.":RETURN
5080 IF A$="8" THEN PRINT"ENOUGH TO CAUS
E REGRET.":RETURN
5090 IF A$="9" THEN PRINT"THE FIRST WILL
BE THE LAST.":RETURN
5100 IF A$="10" THEN PRINT"THE WISE ALWA
YS PLAN AHEAD.":RETURN
5110 IF A$="11" THEN PRINT"TOO MANY BRIN
GS DISGUST.":RETURN
5120 IF A$="12" THEN PRINT"THEY WAIT AND
YOU WILL NOT RESIST.":RETURN
5130 IF A$="13" THEN PRINT"YOU WILL LEAR
N MUCH ABOUT LIFE.":RETURN
5140 IF A$="14" THEN PRINT"THE FIRST WIL
L WHET YOUR APPETITE.":RETURN
5150 IF A$="15" THEN PRINT"SOME BY DAY,
BUT MANY BY NIGHT.":RETURN
5160 IF A$="16" THEN PRINT"YOU WILL LOOK
, BUT SELDOM FIND.":RETURN
```

```
5500 REM ERROR TRAP
5510 PRINT"{RVS}THE CHOICE IS 1-5.{ROFF}"
    ":RETURN
6000 REM END
6010 PRINT"GAME ENDED"
6020 END
```

Part 3

PUTTING PICTURES INTO YOUR GAMES



Chapter 10

BASIC GRAPHICS TOOLS

Graphics is one of the most exciting areas in the world of microcomputers. In fact, you probably had the Commodore 64's powerful graphics capabilities in mind when you decided to buy it. The Commodore 64 has three graphics modes. The first mode we will call keyboard graphics, the second sprite mode, and the third high-resolution graphics. There are also three color modes: standard, extended, and multicolor.

Graphics

Let's take a look at keyboard graphics. In a broad sense, every computer has keyboard graphics. That is, visual effects can be created on any computer by displaying the text characters on the screen. The Commodore 64 is different in that, in addition to regular text, it has an entire set of special graphics symbols which can be displayed by pressing keys.

What is really exciting about the Commodore's keyboard graphics is that you are not limited to the standard character set. In Chapter 13 you will see how it is possible to create new keyboard characters, an entirely new set if you wish. The uses for custom keyboard graphics are many. For instance, you might want to create special characters for foreign languages. In a game, custom characters could be used as playing pieces or to build interesting background illustrations.

All keyboard characters, whether standard or custom, are displayed on the Commodore 64's text screen. This screen consists of 25 lines, each line comprised of 40 character-columns.

Sprites comprise the second graphics mode. Sprites are custom-designed shapes which can be moved around the screen without erasing the background illustrations. Your Commodore 64 can display up to eight sprites at a time. Among other uses, sprites are perfect for fast-moving rockets, airplanes, etc. But sprites do not have to move. They can be very effective as stationary screen illustrations.

The final graphics mode we will mention is the high-resolution mode. Whereas the keyboard graphics mode uses a 25-by-40 grid, the hi-res mode uses a 320-by-200 grid! That allows for some very detailed drawing. Unfortunately, the high-resolution mode is somewhat difficult to use from BASIC, so we will not focus on it as much as we would in a book devoted to graphics alone.

Color

Besides the different graphics modes, you can choose between three color modes. The first is standard mode, in which each character can be either the background color or a specific character color. This mode is fine for most purposes.

If you want more variety, the extended color and multicolor modes are useful. In extended color mode, the background color as well as the character color can be different for each character position. In the multicolor mode, even more variety is possible: Each dot in a character can be any of four colors!

The added flexibility of color in the extended and multicolor modes is somewhat offset by the lower resolution required. In effect, each character cell is made up of four wider dots across each row instead of the eight dots across in the standard mode. The reason for the larger dots is that it takes pairs of dots to keep track of all those colors. In the standard mode an individual dot, or pixel, is either in background or in character color. This two-way choice is handled nicely in binary (which computers like to use).

Look at the following eight-cell representation:

0 0 0 1 1 0 1 1

In multicolor mode this line, read left to right, represents background color 0, background color 1, background color 2, and character color (all of which can be different). In standard color mode, the same line would be read as eight individual pixels, 0 standing for the background color, and 1 standing for the character color of that particular character cell.

As you can see, the different color modes can be a little confusing at first. So rather than going into too much detail, we will focus on the standard color mode.

To summarize, the Commodore 64 has three graphics modes, two of which we will discuss in the following pages: keyboard graphics and sprites.



Chapter 11

COMMODORE 64 KEYBOARD GRAPHICS

Perhaps the most exciting feature of the Commodore 64 is its easy-to-use graphics, and the keyboard graphics characters are the machine's number one graphics tool. Alone among personal computers, the Commodore offers the simplicity of graphics symbols right on the keyboard. The symbols are displayed on the face of each key (not on top).

For example, if you want to create a smiling face, the circles shown on the W and Q keys can be produced by holding down the SHIFT key while you press W or Q (SHIFT-W or SHIFT-Q). The smile might also be produced by using a combination of straight and curved lines shown on the C, J and K keys. If, instead of a smiling face, you want to use the symbol pictured to the left of the circle on the Q key, you can obtain this by typing CMD-Q (pressing the CMD and Q keys simultaneously).

Keyboard Graphics

To include these keyboard characters in your programs, you can use either PRINT statements or POKE commands. The simplest way to begin is to use PRINT statements. Suppose that you would like to display a smiling face on the screen each time a player masters some feat or earns points in a game. This is how you might do it:

```
90 PRINT CHR$(147)
100 PRINT "    (SHIFT) W (SPACE)
```

```

(SHIFT) W "
110 PRINT " (SHIFT) B (SPACE) (SPACE)
(SPACE) (SPACE) (SPACE) (SHIFT) - "
120 PRINT " (SHIFT) J (SPACE) (SPACE)
(SPACE) (SPACE) (SHIFT) K "
130 PRINT " (SHIFT) * (SHIFT) *
(SHIFT) * (SHIFT) * (SHIFT) * "
140 END

```

Enter the program and notice that the smiling face is easy to see when the program is listed, without necessarily having to run the program. That is, it is easy to see when listed, provided that you are using a Commodore serial interface printer. If you have a parallel interface printer, you will see only typical typewriter keyboard characters in the printout, and it will be difficult for you to recognize your creative keyboard characters.

The following code produces a square:

```

90 PRINT CHR$(147)
100 PRINT " (SHIFT) O (SHIFT) P "
110 PRINT " (SHIFT) L (SHIFT) @"
120 END

```

Notice how the square is produced. Try making your own outline for a square or rectangle. The trick is finding characters which will "match up" with each other, so there are no unwanted gaps or overlaps in the drawing. In our example, the corners of the square are produced by holding down the `SHIFT` while pressing the O, P, L, and P keys.

Now suppose you want to make a rectangle. The task is to find characters which will match the horizontal bars in the corners of the square. For example:

```

90 PRINT CHR$(147)
100 PRINT " (SHIFT) O (CMD KEY) Y

```

```
(SHIFT) (SHIFT) P "  
110 PRINT " (SHIFT) L (CMD KEY) P  
(SHIFT) P "  
120 END
```

See how the square has been elongated? Try turning the rectangle around so that it is vertical, rather than horizontal.

```
90 PRINT CHR$(147)  
100 PRINT " (SHIFT) O (SHIFT) P "  
105 PRINT " (CMD KEY) H (CMD KEY) N"  
110 PRINT " (SHIFT) L (SHIFT) @"  
120 END
```

Now that you have mastered the rectangle, try your hand at the other characters. How about making a playing card by placing the characters on the A, S, Z, and X keys within the rectangle you have just created?

Once you feel comfortable using these characters, you will find that you can think of countless ways to incorporate them into your games. Chapter 13 and subsequent chapters show you how to put these characters into game programs, including the games we have already discussed.

Color and Keyboard Graphics

Much of the fun of keyboard graphics comes from the use of color. All that is involved in adding color to your pictures is pressing the appropriate key for the color you want when you are creating the PRINT statement shown above. For example, retype line 100 of the program shown above, but this time, just before you type SHIFT-O, type CTRL-6 (hold down the CTRL key and press the number 6 key). You should see a character on the screen that looks like an up-arrow. Continue typing the line as before. When RUN, the program should give you a green square.

Now, retype line 100 again, but this time just before closing the quotes, type CTRL-1. Now, only the top of the square is green, while the rest is black. If you do nothing to change it, your cursor will remain black with anything you type or run. The line at the beginning of the games (POKE 53281,1:PRINT CHR\$(144)) sets the background and cursor colors. Try the program with each color by using the SHIFT key, then try them again by pressing the CMD key. You should be able to produce 16 different colors this way.

A note of caution: It is best to place the colors where you want them as you are typing the line, because it is difficult to backspace without disrupting the color codes. Also, you will notice that placing a color character code in a line seems to move the picture that is on that line. If you tried to move it back by deleting spaces, then you know it doesn't work. The appearance of the picture in your program listing is skewed, but it will be centered when RUN. If you "fix" the line to correspond with the rest of the program as it appears on the screen, and then RUN the program, the picture will be off center.

Chapter 12

ANIMAL JUMP: A CHILD'S GAME

If you have a small child around the house, *Animal Jump* is designed specifically for that little boy or girl. The game does not require the ability to read or type. The child merely presses keys at random to make various animals and colors appear on the monitor. But there is much of interest to the beginning game programmer in *Animal Jump*. In fact, several very important techniques introduced here will be invaluable in coming chapters.

First, here's the road map of *Animal Jump*:

1. Game instructions are displayed.
2. The child presses keys.
3. If the key pressed is a function key, an animal appears on the screen.
4. If the key pressed is any other key (but not the STOP key; the child will have to learn not to touch that one!), the screen and border colors change.

That's it! It doesn't sound like much from an adult's point of view. Small children, however, will delight in being able to control changes in color and shapes which appear on the screen. And you, the beginning programmer, will find that the game utilizes many sophisticated and powerful game techniques.

More BASICs: POKE, PEEK, READ, and DATA

Four of the instructions we want to focus on are POKE, PEEK, READ, and DATA. They figure prominently in the next chapter which covers custom characters. POKE and PEEK are statements that directly access specific locations in your Commodore 64's memory. POKE places a value in a memory location. PEEK, on the other hand, determines what value is currently stored in a particular byte of memory. A practical analogy for PEEK and POKE is as close as your desk. Sometimes you open a drawer and drop an item in; that equates with POKE. Other times you open a drawer to see what is inside; that corresponds with PEEK.

Enter this line on your Commodore 64 and press return: POKE 53281,2. Your screen turned to red (on a black-and-white monitor you will at least see a change). Now type 53280,2 followed by return. The border now turned to red. Locations 53280 and 53281 are like switches. By POKEing color codes (0-15) into these locations you can change the border and screen colors. Not all memory locations in your Commodore 64 are "soft switches," as they are called. Most of the Random Access Memory (RAM) area is like those drawers in your desk: You can put in them whatever you want.

Now enter this line (we will assume from now on that you will automatically press RETURN): PRINT PEEK(53281). If your screen color was still red, a 2 was printed on the screen. You have used PEEK to look into location 53281 and PRINT to display whatever value was stored there. POKE changes the contents of a memory location, but PEEK merely looks without altering the contents of the byte. Being able to discover the contents of a memory location with the PEEK statement has more uses than you can now imagine. But let's continue with *Animal Jump*, where we will see PEEK and POKE, as well as READ, DATA, and other interesting instructions in action.

The structure of *Animal Jump* is rather simple. After the initial REMark statements, the subroutine at line 1000 is called. This subroutine displays the game's title and instructions. The main

program routine is located in lines 600-700. Here the key being pressed, if any, is determined. Color or shape change is made on the screen, depending on which key is pressed. If a new animal shape is to be drawn, the appropriate subroutine is called.

The animal-display subroutines are in lines 100-505. Another subroutine, at line 2000, causes a brief pause during the game's introduction.

Let's go back to the setup subroutine in lines 1000-1120. `READ` and `DATA`, two important statements mentioned earlier, are found in lines 1010, 1012, and 2050. If `DATA` refers to numerical values included in your program, they are always in decimal form. In Chapter 13, `DATA` statements are used to hold numbers which define custom characters. In *Animal Jump* we will use `DATA` statements to define character codes. In line 1008 two numeric arrays, `D(11)` and `D1(7)`, are `DIMENSIONED`.

Loading DATA Statements with READ

In line 1010 a `FOR/NEXT` loop `READS` the items held in the `DATA` statement in line 2050 one item at a time. The variable name `A` is used to refer to each item as it is `READ` from the `DATA` statement. After `A` has been defined by `READING` from the `DATA` statement, the value of `A` is stored as an element of an array. In line 1010, the array is `D()`, in 1012 the array is `D1()`. As the loop is performed again, the next item in the `DATA` statement is `READ`, giving `A` a new value. The next element in the array is then assigned whatever value `A` has at that moment. This continues until the loop has been performed the correct number of times (11 times in line 1010, 7 times in line 1012).

Since this is a lot to understand, let's go through it again. The first time through the loop in line 1010, the first item in the `DATA` statement in line 2050 is `READ`. That item is the decimal number 65. The value 65 is given to the numeric variable `A` (i.e., `A=65`). Next, the first element in array `D()`, which is `D(1)`, is assigned the value of `A`, which is, don't forget, 65. When the loop has been gone through once, we have the following facts: `A=65`, `D(1)=65`, and

I=2. The second time through the loop, the second item in line 2050 is READ. That number is 32. Variable A is now given the value 32. Then 32 is assigned to the second element in array D(). After the loop has been run through twice, A=32, A(2)=32, and I=3.

The program continues to loop until the first 11 DATA items have been READ and assigned to the 11 elements of array D(). In other words, the array now has these values:

```
D(1)=65
D(2)=32
D(3)=78
D(4)=32
D(5)=73
D(6)=32
D(7)=77
D(8)=32
D(9)=65
D(10)=32
D(11)=76
```

A new FOR/NEXT loop in line 1012 continues the process by READING the remaining DATA items and assigning their values to the elements of array D1(). We want to stress the fact that your Commodore 64 always remembers the last DATA item READ. When we tell it to READ more items (as in line 1012), it does not go back and read the first item again. The C-64 starts READING DATA with the next item which follows the last one READ. (Sometimes you may actually want the computer to start READING the DATA again from the beginning. To do that, you must use the statement RESTORE before READING the DATA a second time.)

Now we have two arrays containing the numbers contained in the DATA statement in line 2050. Those numbers each represent a character code, referred to as a character string. The term CHR\$(read as "character string") followed by a number inside parentheses is the code for a screen character. Letters, numbers, punctuation, and keyboard graphics symbols all have unique character

string numbers. A space is `CHR$(32)`, the letter Z is `CHR$(90)`, a spade is `CHR$(97)`, and so on. Some character string codes refer to functions rather than screen symbols. For instance, `CHR$(29)` means “move cursor one space right.” `CHR$(18)` means “print the following characters in reverse (white character on a black background).”

PRINTing with FOR/NEXT Loops

The two arrays set up in lines 1010 and 1012 contain nothing but screen characters, and they will be used to display the name of the game, “ANIMAL JUMP.” Line 1015 clears the screen (here represented by a reverse heart), moves the cursor down 14 lines, then 6 spaces right. The semicolon which follows the close quotation marks means that whatever is `PRINTed` next will appear on the next space to the right, not on the next line down. The remaining portion of line 1015 is a `FOR/NEXT` loop which prints the screen characters symbolized by the numbers contained in array `D()`.

Try this on your Commodore 64. Type `PRINT CHR$(82)`. The letter R appears on the screen. Now enter `PRINT CHR$(82) CHR$(69)CHR$(68)`. The word “RED” should appear. `CHR$(82)` is R, `CHR$(69)` is E, and `CHR$(68)` is D. In line 1015, however, the statement `PRINT CHR$(D(I))` is used. The `D(I)` refers to the first element of array `D()`. The first time through the loop, I equals 1, so `CHR$(D(I))` has the meaning `CHR$(D(1))`. This is read, “`CHR$(whatever number is contained in the first element of the array variable D)`.” That number is 65 which is the character code for the letter A. The next time through the loop, the meaning is `CHR$(whatever is contained in the second element)`. With `CHR$`, the item within parentheses can be a number (`CHR$(55)`), a numeric variable (`CHR$(X)`), or an array element identified either by a number (`CHR$(D(3))`) or by another numeric variable, as in lines 1015 and 1020. If you are going to `PRINT` a series of character strings, using a `FOR/NEXT` loop and an array is a very efficient way to do it.

After each letter has been `PRINTed`, subroutine 2000 is called before the `NEXT` statement. Since the subroutine at 2000 is a

pause loop, this has the effect of slowly displaying the letters of the title one by one instead of all at once. Sometimes you may wish to use this technique to emphasize certain words, or to gain a dramatic effect.

The same sort of FOR/NEXT loop used in line 1015 is used in 1020 to display the letters of the word "JUMP." Line 1030 draws a line of dots across the screen. Again a pause loop is used to draw the dots slowly to achieve a dramatic effect. Lines 1040-1100 are simple PRINT statements. Line 1110 uses GET to set up a loop which is only broken when a key is pressed. The "" means that no key has been pressed. This technique is handy when you want the program to pause until the player is ready to proceed.

Scanning the Keyboard

The main program routine, although short, contains several items worth noting. In line 610, the keyboard is read. This is done by using PEEK. PEEK(197) tells us the number code of the key, if any, currently being pressed. This number is given to variable A (which earns its keep in this program). Lines 620-660 are IF statements which evaluate the keypress and act accordingly. In lines 620, 630, 640 and 650, the program is diverted to the animal-drawing subroutines. Keypress values 4, 5, 6, and 3 correspond to function keys 1, 3, 5, and 7. Each of these subroutines begins with a clear screen (reverse heart) and black cursor symbol (black square). Then the strange looking creatures are drawn with simple PRINT statements using standard keyboard graphics characters.

The RETURN at the end of each subroutine returns the program to the main program routine, where the keyboard is READ once more. If no key has been pressed, A=PEEK(197) will still return a value: the number 64. If this situation is detected in line 660, the program is sent back to 610, where the keyboard is READ once more. On the other hand, if any other key is pressed, the program falls through to lines 665-700. Variable X, in line 665, keeps track of the color code. Whatever the color code was previously, that code is incremented by 1. When X reaches 16, X is made equal

to 1 (we avoid 0 because 0 is black). In line 670, the screen and border colors are both changed. First, the screen color code is made equal to the current border color code. Here is where PEEK is again useful:

```
POKE 53281,PEEK(53280)
```

The above line means “POKE the value found in location 53280 into 53281.” This changes the screen color to whatever the border color was previously. Next, the border color is changed by:

```
POKE 53280,X
```

X, keep in mind, was just incremented by 1. Therefore, if the border color was 3 and the screen color was 2, the border color will become 4 and the screen color will become 3.

Summary

Animal Jump can be a lot of fun for youngsters, and it can be very educational for you. Here are the main things to remember:

1. POKE stores a value in a memory location (e.g., POKE 5000,38 stores 38 in RAM byte 5000).
2. PEEK determines the current value stored in a particular RAM byte (e.g., B=PEEK(12288) assigns to variable B whatever value the PEEK found in location 12288).
3. READ looks at the next item listed in DATA statements. After one item is READ, the following item is always READ next.
4. DATA contains decimal numbers, separated by commas, which are identified in the READ operation.

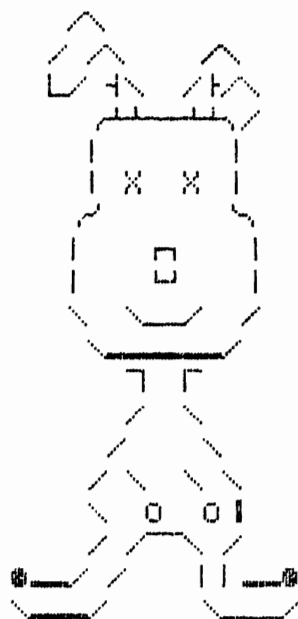
In the next chapter we'll learn how to design custom keyboard graphics characters.

Animal Jump

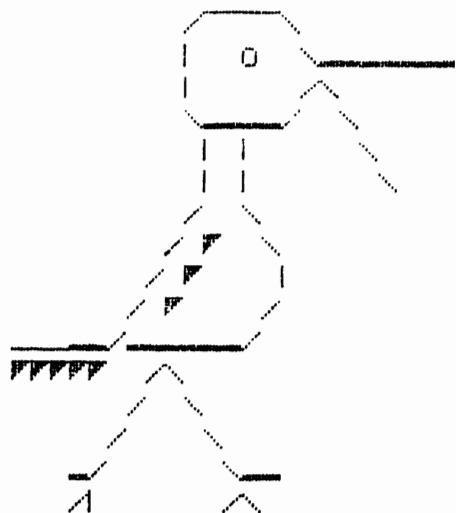
```
1 POKE53280,0
2 POKE53281,0
10 REM *****
11 REM * ANIMAL JUMP *
12 REM * A CHILD'S GAME *
13 REM * PROGRAM ILLUSTRATES *
14 REM * <READ>, <DATA>, <POKE>, *
15 REM * AND <PEEK> *
16 REM *****
20 GOSUB 1000
30 GOTO600
100 PRINT" {CLR} {BLK} "
105 PRINT"
110 PRINT"
115 PRINT"
120 PRINT"
125 PRINT"
130 PRINT"
135 PRINT"
140 PRINT"
145 PRINT"
150 PRINT"
155 PRINT"
160 PRINT"
165 PRINT"
170 PRINT"
175 PRINT"
180 PRINT"
190 PRINT"
195 RETURN
200 PRINT" {CLR} {BLK} "
```



```
206 PRINT"  
208 PRINT"  
210 PRINT"  
212 PRINT"  
214 PRINT"  
216 PRINT"  
218 PRINT"  
220 PRINT"  
222 PRINT"  
224 PRINT"  
226 PRINT"  
228 PRINT"  
230 PRINT"  
232 PRINT"  
234 PRINT"  
236 PRINT"  
238 PRINT"  
240 PRINT"  
242 PRINT"
```



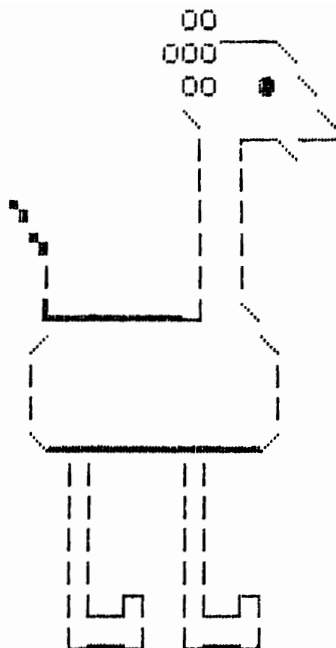
```
250 RETURN  
300 PRINT" {CLR} {BLK} "  
320 PRINT"  
325 PRINT"  
330 PRINT"  
335 PRINT"  
340 PRINT"  
345 PRINT"  
350 PRINT"  
355 PRINT"  
360 PRINT"  
365 PRINT"  
370 PRINT"  
375 PRINT"  
380 PRINT"  
385 PRINT"  
390 PRINT"  
395 PRINT"  
398 RETURN  
400 PRINT" {CLR} {BLK} "
```



```

405 PRINT"
410 PRINT"
415 PRINT"
420 PRINT"
425 PRINT"
430 PRINT"
435 PRINT"
440 PRINT"
445 PRINT"
450 PRINT"
455 PRINT"
460 PRINT"
465 PRINT"
470 PRINT"
475 PRINT"
480 PRINT"
485 PRINT"
490 PRINT"
495 PRINT"
500 PRINT"
505 RETURN

```



```

600 REM *****
601 REM * MAIN PROGRAM ROUTINE *
602 REM *****
610 A=PEEK(197)
620 IF A=4 THEN GOSUB 100:GOTO 610
630 IF A=5 THEN GOSUB 200:GOTO 610
640 IF A=6 THEN GOSUB 300:GOTO 610
650 IF A=3 THEN GOSUB 400:GOTO 610
660 IF A= 64 GOTO 610
665 X=X+1:IF X>15 THEN X=1
670 POKE53281,PEEK(53280):POKE53280,X
700 GOTO 610
999 END
1000 REM *****
1001 REM * SETUP AND INSTRUCTIONS *
1002 REM *****
1003 :
1005 POKE53280,0:POKE53281,0:REM - SET
      SCREEN AND BORDER COLORS TO BLA
CK

```

```

1007 PRINT"{WHT}":REM -CURSOR TO WHITE
1008 DIM D(11),D1(7)
1010 FORI=1TO11:READ A:D(I)=A:NEXT
1012 FORI=1 TO 7:READ A:D1(I)=A:NEXT
1015 PRINT"{CLR}{DOWN 14}{RIGHT 6}";:FOR
I=1TO11:PRINTCHR$(D(I));:GOSUB 2000:NEXT
1020 PRINT"{DOWN}{RIGHT 8}";:FORI=1TO7:P
RINTCHR$(D1(I));:GOSUB 2000:NEXT:PRINT
1030 PRINT"{DOWN}";:FORI=1TO40 :PRINT"."
;:FORJ=1TO 25:NEXTJ,I
1040 PRINT"{DOWN 2}{RIGHT 2}THIS IS A LI
TTL E GAME FOR VERY SMALL":GOSUB 2000
1050 PRINT"{DOWN 2}{RIGHT 2}CHILDREN. W
HEN THE CHILD PRESSES ":GOSUB 2000
1060 PRINT"{DOWN 2}{RIGHT 2}A DIFFERENT
FUNCTION KEY, A":GOSUB 2000
1070 PRINT"{DOWN 2}{RIGHT 2}NEW ANIMAL A
PPEARS. PRESSING":GOSUB 2000
1080 PRINT"{DOWN 2}{RIGHT 2}OTHER KEYS C
HANGES SCREEN AND BORDER"
1090 PRINT"{DOWN 2}{RIGHT 2}COLORS.
HAVE FUN!"
1100 PRINT"{DOWN}{RIGHT 5}{RVS} PRESS AN
Y KEY TO BEGIN "
1110 GETA$:IFA$="" GOTO1110
1120 PRINT"{CLR}":GOTO 600
2000 FOR P=1TO150:NEXT:RETURN
2050 DATA 65,32,78,32,73,32,77,32,65,32,
76,74,32,85,32,77,32,80

```

The following line ranges show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustrations in *Animal Jump*.

```

105 PRINT"                                {#206}{#205}

110 PRINT"                                {#206}{#180}
{#205}

115 PRINT"                                {#205 2} {#17
0}

```

```

120 PRINT"                {#205 2}{#20
6}
125 PRINT"                {#208}{#207
}{#183 3}{#205}
130 PRINT" {#213}{#201}      (
#206) {#213}{#201} {#205}
135 PRINT" {#205} {#205}      (#
206) {#202}{#203} {#205}
140 PRINT" {#205}{#175}{#205}{#210 12}(
#206) {#182}
145 PRINT" {#206}          (
#205) {#206}
150 PRINT" {#206}          {#20
6}{#183 4}
155 PRINT" {#206}          {#206
}
160 PRINT" {#205} {#175 11} {#203}
165 PRINT" {#205} {#205}      {#20
5} {#205}
170 PRINT" {#205} {#205}      {#2
05} {#205}
175 PRINT" {#206} {#206}      {#
205} {#205}
180 PRINT" {#206} {#183}{#208}
{#205} {#183}{#208}
190 PRINT" {#204}{#175 3}{#186}
{#205}{#175 2}{#186}

206 PRINT " {#206}{#205}
208 PRINT " {#206} {#206}{#205}
{#206}{#205}
210 PRINT " {#204}{#206} {#179}{#
205} {#206}{#171}{#206}{#205}
212 PRINT " {#213}{#177 2}{#192
2}{#177 2}{#201}{#206}
214 PRINT " {#199} {#200}
216 PRINT " {#199} X X {#200}
218 PRINT " {#213}{#203} {#
202}{#201}
220 PRINT " {#212} {#176}{#174}
{#200}

```

```

222 PRINT "          {#212}    {#173}{#189}
      {#200}
224 PRINT "          {#205}    {#205}{#164 2
}{#206}    {#206}
226 PRINT "          {#205}{#175 6}{#206}

228 PRINT "          {#208}    {#207}
230 PRINT "          {#206}    {#205}
232 PRINT "          {#206}    {#205}
234 PRINT "          {#206} {#205}    {#205
} {#205}
236 PRINT "          {#205}    {#215}    {#21
5}{#182}
238 PRINT "          {#206} {#206}{#183 2
}{#205} {#206}
240 PRINT "          {#209}{#175 2}{#206} {#2
06}    {#170} {#180}{#175 2}{#209}
242 PRINT "          {#205}{#175 3}{#206}
      {#205}{#164 3}{#206}

320 PRINT"          {#206}{#183
  4}{#205}
325 PRINT"          {#180}    {#2
15}    {#205}{#175 7}
330 PRINT"          {#165}
{#206}{#205}
335 PRINT"          {#205}{#175
  4}{#206}    {#205}
340 PRINT"          {#165} {#1
65}    {#205}
345 PRINT"          {#165} {#1
65}    {#205}
350 PRINT"          {#206}    {#2
05}
355 PRINT"          {#206} {#169
} {#205}
360 PRINT"          {#206} {#169}
      {#165}
365 PRINT"          {#206} {#169}
      {#206}

```

```

370 PRINT"                {#164 3}{#175 2}{#20
6}{#175 6}{#206}
375 PRINT"                {#169 5}  {#206}{#20
5}
380 PRINT"                {#206}  {#205}

385 PRINT"                {#206}  {#205
}
390 PRINT"                {#175}{#206}
  {#205}{#175 2}
395 PRINT"                {#206}{#180}
  {#206}{#205}

405 PRINT"                {#215 2}
410 PRINT"                {#215 3}{#183 3}
{#205}
415 PRINT"                {#215 2}  {#209
} {#205}
420 PRINT"                {#205}          {#2
05}
425 PRINT"                {#170}  {#207}{
#183}{#205}{#183 2}
430 PRINT"                {#170}  {#180}
435 PRINT"                {#191}          {#170}  {#
180}
440 PRINT"                {#191}          {#170}  {#
180}
445 PRINT"                {#170}          {#170}  {#
180}
450 PRINT"                {#182}{#175 7}{#186}  {
#205}
455 PRINT"                {#206}          {#205}

460 PRINT"                {#165}          {#170}

465 PRINT"                {#165}          {#170}

470 PRINT"                {#205}{#175 11}{#206}
475 PRINT"                {#180 2}    {#180 2}
480 PRINT"                {#180 2}    {#180 2}

```

```
485 PRINT"          {#180 2}      {#180 2}
490 PRINT"          {#180 2}      {#180 2}
495 PRINT"          {#180}{#204}{#186}{#2
08} {#180}{#204}{#186}{#208}
500 PRINT"          {#204}{#175 2}{#186}
      {#204}{#175 2}{#186}
```


Chapter 13

MAKING YOUR OWN CHARACTERS

We have already seen how much keyboard graphics can add to your Commodore 64 games. There is yet another graphics surprise in store for first-time Commodore game designers. In addition to the full set of graphics characters found on the keyboard, it is possible to create your own custom characters which can be used quite easily in your games.

At the end of this chapter there is a program which will help you create custom characters. This character editor program even generates the numeric codes which define the characters, and which you must include in your games. Nevertheless, the method by which these custom characters can be created is worth outlining in some detail. Every computer has as part of its permanent, Read Only Memory (ROM) a character generator which contains the information for sending letters, numbers, etc., to the screen. The user usually has no choice in this matter; the standard character set must be used.

With the C-64, however, you can make the computer ignore this built-in base of information. Instead, the computer can use data of your own creation. In this way, the characters displayed on the screen can be any of those included in the basic graphic character matrix itself.

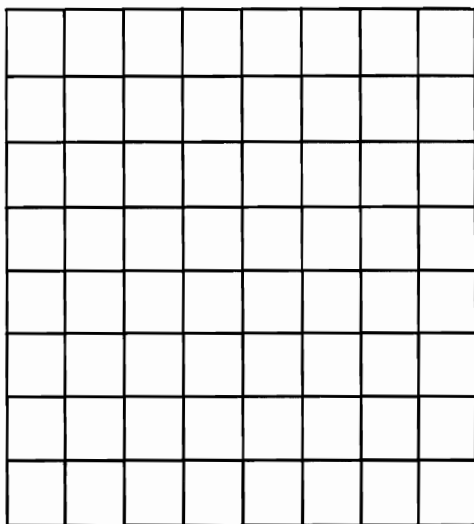
Designing Custom Characters

The basic Commodore graphic character is a matrix made up of eight rows of eight dots each. Each dot can be OFF (blank) or ON

(lighted). Here is an example of such a character, in this case the letter A (each X represents an ON dot; each period an OFF dot):

```
. . . XX . . .  
. . XXXX . .  
. XX . . XX .  
. XXXXXX .  
. XX . . XX .  
. XX . . XX .  
. XX . . XX .  
. . . . . . .
```

The first step in designing a character is to make some 8-by-8 grids on paper, as in this example:



Experiment with these grids, putting an X wherever you want an individual dot to be ON. Look closely at your video screen, and you will see how each letter is made up of individual dots. In the same manner, you can design a pattern of OFF and ON dots to create whatever design you want.

After you have designed a character, you are ready for step two: converting that pattern of ON and OFF dots to numbers. To do

this you must assign numerical values to each column in each row. If a dot in a given column is ON, it takes the numerical value of that column. If the dot is OFF, its value is 0.

This process of digitizing your character information will be more easily understood with this visual example:

128	64	32	16	8	4	2	1
	X		X	X	X		X

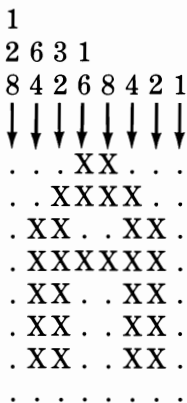
In the example, each column of the 8-by-8 grid was given a specific number value. The 128, 32, and 2 columns are OFF (blank). Each column therefore takes the value of 0. The 64, 16, 8, 4, and 1 columns are ON (filled with an X). Each of these takes the numerical value of its particular column. Let's add up the numerical values which correspond to the ON squares:

$$64 + 16 + 8 + 4 + 1 = 93$$

This row of dots, therefore, has a numerical value (or code) of 93. In other words, a line in which the second, fourth, fifth, sixth, and eighth dots (reading left to right) are ON has a numerical value of 93. How about a line in which only the last dot (i.e., the right-most) is ON? Since the numerical value for the right-most dot is 1, that whole line will have a numerical code of 1.

A line with only the left-most dot ON would have a value of 128. A line with only the last two dots ON (i.e., the seventh and eighth dots) would have a value of 3, and so on. Once you get used to the idea of each column in a character grid having a specific numerical value, it should be easy to calculate the numerical code for an entire character.

Before we go to the next step, let's go back to our example of the letter A. Using the number coding discussed above, we will calculate the numerical code for that character. (The number for each column is shown vertically: 128, 64, etc.)



In the top row (row one), the 16 and 8 dots (or bits) are ON; hence a value of 24 for that line. In line two, the 32, 16, 8, and 4 bits are ON. Add those up and you get a total of 60 for line two. And so it goes.

In your program, you would include all eight totals as the data which represent and define the letter A. Your program line would be:

```
100 DATA 24, 60, 102, 126, 102, 102, 102, 0
```

Notice that lines three, five, six, and seven have the same value, since in each of those lines the same dots are ON. The last code, zero, is for the bottom line of the character in which no dots are ON. Just for a little practice, figure out for yourself the numerical value of a line in which all eight dots are ON.

You may have been wondering, "Where did the numbers 128, 64, 32, 16, 4, 2, and 1 come from?" That's an important question if you really want to understand how your Commodore 64 operates. The reason for the 128, 64, etc., is that they are the decimal (base 10) equivalents of the binary (base 2) values which the Commodore 64 actually uses as it runs your program. Notice that each of the numbers is divisible by 2 (except 1 and 0, of course).

Let's recapitulate. First, make a drawing of your custom character on an 8-by-8 square grid. Remember, each square must be either

empty or completely filled. Next, assign a numerical value to each horizontal line by adding the values of all the dots which are ON—that is, filled in. Any dot (square on the grid) which is OFF has a value of 0.

Displaying Custom Characters

Now that you have designed your custom character, you must be wondering how you are going to get your Commodore 64 to display it on the screen. This takes a number of steps to accomplish.

As stated, when you power up the Commodore 64, it will automatically use the built-in character set stored in ROM. To use your custom characters, you must make the C-64 ignore the built-in characters and use the ones that you designed instead. To do this, you first copy the character data from ROM into a portion of RAM where changes can then be made.

Here is a short program that will copy the data for all 256 characters from character set one into RAM (note that character set one contains capital letters and graphic characters; set two contains upper- and lowercase):

```
10 REM *****
11 REM * A PROGRAM TO MOVE DATA FOR *
12 REM * ALL 256 CHARACTER OF SET#1 *
13 REM * TO RAM, STARTING AT 12288. *
14 REM * END OF BASIC MOVED DOWN, *
15 REM * LEAVING 10237 BYTES FREE *
16 REM * FOR BASIC PROGRAM. *
17 REM * INCLUDES EXAMPLE OF TWO *
18 REM * CUSTOM CHARACTERS. *
19 REM *****
100 PRINTCHR$(142):REM-SELECT CHAR SET#1

105 POKE53281,1:POKE53280,1:PRINT"{BLK}"
:REM -SCREEN & BORDER WHITE, CURSOR BLAC
K-
108 PRINT"{CLR}{DOWN 5}{RIGHT 8}PLEASE W
AIT."
```

```

110 POKE52,48:POKE56,48:CLR:REM-LOWERS
    END OF BASIC: 10237 BYTES LEFT FOR PRG

120 POKE56334,PEEK(56334)AND 254:REM -TU
RN    OFF KEYBOARD-
130 POKE1,PEEK(1)AND 251:REM -SWITCH IN
    CHARACTER ROM-
140 FORI=0TO2047:POKEI+12288,PEEK(53248+
I)
150 NEXT:REM-TRANSFERS CHAR DATA TO RAM-

160 POKE1,PEEK(1)OR4:REM-SWITCH I/O IN-
170 POKE 56334,PEEK(56334)OR1: REM -
    KEYBOARD BACK ON -
180 POKE53272,(PEEK(53272)AND 240)+12
200 FORI=0TO15:READ A:POKEI+12288,A:NEXT

210 PRINT"{CLR}":A$="@":B$="A"
220 FORI=1TO10:PRINTSPC(X);:X=X+1
230 FORJ=1TO10:PRINTA$;:NEXT:PRINT
240 NEXT I
250 FORI=1TO10:PRINTSPC(X);:X=X-1
260 FORJ=1TO10:PRINTA$;:NEXT:PRINT
270 NEXT I
300 POKE53281,0:REM -SCREEN TO BLACK
310 POKE53280,0:REM -BORDER TO BLACK
320 PRINT"{WHT}":REM -CURSOR TO WHITE
330 PRINT"{CLR}"
340 FORI=1TO10:PRINTSPC(X);:X=X+1
350 FORJ=1TO10:PRINTB$;:NEXT:PRINT
360 NEXT I
370 FORI=1TO10:PRINTSPC(X);:X=X-1
380 FORJ=1TO10:PRINTB$;:NEXT:PRINT
390 NEXT I
395 GOTO 210
400 DATA 0,31,56,124,126,62,28,120
410 DATA 0,248,28,62,126,124,56,30

```

Run this program. If you have made no typing mistakes, your Commodore will ignore you for about one minute. You will then see

a pattern display using two custom characters, the symbol @ and the letter A in modified form. Now clear the screen and press the @ and A keys. You now have two custom characters!

If you now type a short program, or load one from tape or disk, it will run normally. Try pressing the C= (Commodore) and SHIFT keys together. Anything on the screen turns to garbage! Remember, we are only using the character data which was copied into RAM. We only copied the 256 characters of set one. Pressing the C= and SHIFT keys switches the Commodore to character set two. Since we did not copy the data for set two, the Commodore cannot display it.

Do not fear! Simply press the C= and SHIFT keys again, and you will be back in business. Since we want to use the graphic characters in character set one, we will not feel the loss of the lowercase letters in set two.

Customized Characters in Action

Now you are ready to do some character customizing of your own. First, we must consider which characters to modify. This decision is based on the nature of the game you are creating. In most cases you would not wish to modify the letters of the alphabet. Many other characters out of the basic 256 can be dispensed with, however. As stated before, you must plan your game in advance. In this instance, that means determining which characters are indispensable and which can be done without.

After you decide which characters you wish to change, you must redefine those characters to suit your needs. Let's say you want to change the @ symbol. It so happens that the eight bytes defining the @ begin at location 12288 (which begins the large block of memory we put off limits to the BASIC program). To modify this character, which is CHR\$(60), simply store new data in locations 12288-12295. The following line, which can be typed in the immediate mode, will change the @ to a solid black square:

```
FOR I=0 TO 7:POKE 12288 + I,255:NEXT
```

This loop puts (POKES) the decimal value 255 in the appropriate locations. Examine the line closely. Since the loop starts at 0, the first address is 12288 plus 0 (the initial value of I). The second time through the loop, I has a value of 1, so the location is 12288 plus 1, which is location 12289. The next time through the loop, I has a value of 2, so the address being POKED is 12288 plus 2. And so it goes.

This example produces a solid colored square. The value of 255 is POKED into each 8 bytes of memory since all dots in each line are turned ON. Here is a short program which POKES different data into those same locations. Try it and see what results you get when you press the @ key.

```
10 FOR I=0 TO 7:READ A:POKE 12288+ I,A:
NEXT
20 DATA 24,24,60,90,24,60,36,102
```

If you saw a row of little men as you pressed the @ key several times, you have successfully created a custom character! It's a good idea to experiment with this character a few times. Design different characters using your 8-by-8 grids, then modify the DATA statement in the short program above. Each time you run the program with new values, the @ key will display something different. It's fun! Remember, each number in the DATA line is a decimal value for a horizontal row of dots in the character cell.

After some experimenting, you will want to do serious character customizing for a specific game. If possible, select a consecutive group of characters to modify. This has at least two advantages: First, you can easily keep track of which characters are to be modified, and second you can POKES all the necessary bytes into memory with one loop.

The following program will help you keep track of where the data for the various characters is stored in memory. Run the program and it will ask you to input the starting address of the character data memory. If you are using the setup program shown above, you would input 12288. If you are using a program which moves

character memory to the beginning of memory bank two, you would input 32768. The first memory location of each screen character will then appear on your monitor. If you have a printer, you can make a hard copy for your files.

To customize a given character, you will set up a FOR/NEXT loop which POKES the data into the 8 bytes for that character. Be sure that the first memory location you POKE to is the correct starting byte for the desired character. If you are off by just a single byte, you will end up with a garbaged character.

```
10 REM *****
15 REM * THIS PROGRAM LISTS *
20 REM * THE STARTING LOCATION *
25 REM * OF ALL 255 CHARACTERS *
30 REM * OF CHAR. SET #1 WHEN *
40 REM * THAT SET HAS BEEN *
45 REM * COPIED INTO RAM *
50 REM * STARTING AT LOCATION *
55 REM * 12288. SEE PAGE 132 *
60 REM * OF THE C-64 USER'S *
65 REM * GUIDE FOR IDENTITY *
70 REM * OF EACH SCREEN CODE. *
80 REM *****
100 PRINT "{CLR}"
110 PRINT "{DOWN 5}{RIGHT 5}USE THIS PROG
RAM TO CREATE A"
120 PRINT "{DOWN}{RIGHT 3}LIST OF THE STA
RTING LOCATIONS"
130 PRINT "{DOWN}{RIGHT 3}FOR ALL 255 CHA
RACTERS IN CHARACTER"
140 PRINT "{DOWN}{RIGHT 3}SET 1 AFTER THE
Y HAVE BEEN COPIED"
150 PRINT "{DOWN}{RIGHT 3}INTO RAM. CHEC
K PAGE 132 OF THE "
160 PRINT "{DOWN}{RIGHT 3}C-64 USER'S GUI
DE TO IDENTIFY EACH"
170 PRINT "{DOWN}{RIGHT 3}CHARACTER. "
180 PRINT: PRINT "{RVS} PRESS A K
EY TO CONTINUE"
```

```

185 GET A$:IF A$="" THEN 185
188 PRINT"{CLR}";:
190 PRINT"{DOWN 5}{RIGHT 3}TO CHANGE A S
TANDARD CHARACTER"
200 PRINT"{DOWN}{RIGHT 3}INTO A CUSTOM O
NE, USE A LOOP TO "
210 PRINT"{DOWN}{RIGHT 3}POKE 8 DATA BYT
ES STARTING AT THE"
220 PRINT"{DOWN}{RIGHT 3}STARTING LOCATI
ON OF THE STANDRAD "
230 PRINT"{DOWN}{RIGHT 3}CHARACTER TO BE
MODIFIED."
240 PRINT: PRINT" {RVS}PRESS A K
EY TO CONTINUE"
245 GET A$:IF A$="" THEN 245
10000 :
10010 :
10012 PRINT"{CLR}"
10014 INPUT"STARTING LOCATION OF CHAR ME
MORY?";SL:M=SL:X=SL
10015 INPUT"TO SCREEN(S) OR COM. PRINTER
(P)";SP$
10016 IF SP$="S" THEN QQ=1: GOTO 10040
10018 IF SP$<>"P" THEN 10012
10020 OPEN1,4
10040 PRINT"{CLR}"
10041 FOR I=0TO255:IF QQ THEN 10043
10042 CMD1:PRINT#1
10043 IF I<33 THEN XX=1
10045 IF I>129 AND I<149 THEN XX=1
10048 IF I>32 AND I <130 OR I>148 THEN X
X=0
10050 PRINT"START LOCATION OF CHAR "I" I
S "X:
10052 IF XX=1 THEN PRINT:PRINT " -NO
T A DISPLAYED CHARACTER-":PRINT
10054 IF XX=0 THEN PRINT:PRINT "CHR "
I" IS... "CHR$(I)"(BLK)":PRINT
10055 PRINT"{RVS}PRESS ANY KEY TO CONTIN
UE{ROFF}":PRINT
10057 GET C$:IF C$="" THEN 10057

```

```

10058 X=X+8
10060 PRINT"{CLR} ":NEXT I:REM FOLLOWING
LISTS DATA
10090 PRINT"DATA FOR THIS CHAR ARE :"
10092 FOR Y=0T07:PRINTPEEK(X+Y);:NEXT:PR
INT
10100 X=X+8
10110 NEXT I
10858 X=X+8

```

And Finally . . .

A few more points need to be understood before we proceed. When customizing a character, you have only changed its screen appearance. As far as your Commodore is concerned, the character still functions as it always did. You may display your customized character as part of a PRINT statement, as a string variable, or you can POKE it into a screen memory location. In short, a modified character just looks different on the screen (which is what you want!). It can be manipulated as are all keyboard characters. We will see some specific examples of custom characters in the games which you will find in pages yet to come.

Character Editor

```

0 REM *****
1 REM * CHARACTER EDITOR *
2 REM *****
3 POKE53280,1:POKE53281,1:PRINT"{BLK}":P
RINT"{CLR}";:
4 PRINT"{DOWN 3}{RIGHT 3}{RVS}CUSTOM CH
ARACTER EDITOR {ROFF}":PRINT"{DOWN} GE
NERATES DATA STATEMENTS."
5 FORI=1T05000:NEXT:PRINT"{CLR}";:FORI=1
T08:FORJ=1T08
6 PRINT".":NEXT J:PRINT:NEXT I
8 PRINT"{HOME}{DOWN 2}{RIGHT 10}PRESS {R
VS}F1{ROFF} TO PLOT A POINT."

```



```

<50 THEN Y=50:PY=PY+1
230 IF X$="{F1}" THEN PLOT=1
240 IF X$="{F3}" THEN PLOT=0
250 IF X$="{F5}" THEN PLOT=-1
260 IF PLOT=1 THEN GOSUB 400:GOTO 300
270 IF PLOT=-1 THEN GOSUB 500:GOTO 300
300 GOSUB 120:GOTO200:RETURN
400 P$="{#209}":GOSUB 600:RETURN
500 P$=" ":GOSUB 600:RETURN
600 IF PX<0 THEN PX=0
601 IF PX>23 THEN PX=23
602 IF PY<0 THEN PY=0
603 IF PY>20 THEN PY=20
605 POKE781,PY:POKE782,PX:POKE783,0
610 SYS65520:PRINTP$: RETURN
1000 REM *****
1001 REM * CALCULATE DATA STATEMENTS *
1002 REM *   "." IS CHR$(46)           *
1003 REM *****
1005 PRINT"{HOME}{DOWN 16}{RIGHT 4} {RVS
}PLEASE WAIT "
1010 IF XX=777 GOTO 1020
1012 DIM SD(505),BY(64),DE(8):XX=777
1020 FORI=1 TO 24 STEP 3:POKEBY(I),0:NEX
T:X1=1
1025 FORI=1024 TO 1824 STEP 40
1030 FORJ=I TO I+23
1040 SD(X1)= PEEK(J):X1=X1+1:
1050 NEXT J:NEXT I
2000 :
2005 DE(1)=128:DE(2)=64:DE(3)=32:DE(4)=1
6:DE(5)=8:DE(6)=4:DE(7)=2:DE(8)=1
2100 X1=1
2110 FOR I=1 TO 63
2120 FOR J=1 TO 8
2130 IF SD(X1)=81 THEN SD(X1)=DE(J):GOTO
2160
2140 IF SD(X1)<>81 THEN SD(X1)=0
2160 BY(I)=BY(I)+SD(X1)
2170 X1=X1+1
2180 NEXT J,I

```

```
3000 FORI=0 TO 62:POKEI+832,BY(I+1):NEXT
3010 POKE 53248,50:POKE53249,130
4000 :
4070 PRINT"{HOME}{DOWN 13}{RIGHT 4}COPY
THESE DATA FOR YOUR PROGRAM."
4100 PRINT;:
4140 FORI=1TO24 STEP 3
4150 PRINTBY(I)" "":NEXT
5000 PRINT:PRINT"{RVS} TO RUN PROGRAM AG
AIN, PRESS Y"
5010 GETA$:IFA$="" THEN 5010
5020 IF A$<>"Y" THEN END
5030 CLR:GOTO 1
```

Chapter 14

ASK THE WIZARD: WITH GRAPHICS

You might remember thinking that the screen display of numbers 1 through 16 was not a particularly fascinating part of *Ask the Wizard*, and you might have thought that a picture of the Wizard would have been a nice enhancement. Well, *Ask the Wizard: With Graphics* uses the keyboard graphics discussed in Chapter 11 to illustrate the wizard and to create symbols for the 16 numbers used in the game.

If you keyed in *Ask the Wizard* and SAVED it on disk, LOAD it and add these lines:

```
20 PRINT"
22 PRINT"
24 PRINT"
26 PRINT"
28 PRINT"
30 PRINT"
40 PRINT"
42 PRINT"
44 PRINT"
46 PRINT"
48 PRINT"
50 PRINT"
60 PRINT"
70 PRINT"
72 PRINT"
74 PRINT"
76 PRINT"
78 PRINT"
":PRINT:PRINT
79 PRINT"***** T H E   W I Z A R D   *****"
```



PRINTing Colors

Be sure to pay attention to the symbols in the PRINT statements which determine the different colors in the portrait. The character next to the wizard's eyes is made by pressing CTRL-5. Do this before inserting the characters for the eyes. First the > is pressed, then CTRL-5. After pressing keys for the eye characters, press CTRL-1; this makes the < sign black. In other words, the color must be changed back to black before the next character appears which is supposed to be black. Remember, the POKE at the beginning of the program sets the screen background color to white and the cursor to black. Therefore, any picture you draw with keyboard graphics characters will be black unless you change them.

To make the wizard's mouth red, you must press CTRL-3. At the beginning of the wizard's beard, you must reset the cursor color to black, or he will have a red beard. The little characters appearing on the screen and in your VIC 1525 printer listings will not show when the program is RUN; the colors these characters represent will show instead.

Do not try to back up when editing a line with a color character on it. It is better to decide in advance what colors you want in an illustration and insert them as you write the PRINT statements. If you attempt to go back and add the color characters, you will find it difficult to avoid messing up your illustration. Also, the picture will appear to be off center on the screen and in a printer listing on those lines where the color characters are placed. While this makes screen editing a little difficult, you will catch on once you try it a few times.

Keyboard Characters at Work

The Commodore 64's keyboard graphics capability allows your creativity full rein. Just sitting and staring at the keyboard (the front faces of the keys) will give you lots of ideas. It's like going back to kindergarten and making pictures with blocks or puzzles. You have at your disposal two keyboard characters per letter key,

besides the +, -, @, *, and other keys. Each of these characters can be displayed in each of the 16 colors available on the Commodore.

Now, let's replace those numbers in the display.

```

210 PRINT"
220 PRINT"
230 PRINT"
240 PRINT" 1 | 2 | 3 | 4 |
250 PRINT"
260 PRINT"
270 PRINT"
280 PRINT" 5 | 6 | 7 | 8 |
290 PRINT"
300 PRINT"
310 PRINT"
320 PRINT"
330 PRINT" 9 | 10 | 11 | 12 |
340 PRINT"
350 PRINT"
360 PRINT"
370 PRINT"
380 PRINT" 13 | 14 | 15 | 16 |
390 PRINT"

```

These lines create a chart with a series of symbols accompanying the numbers. These symbols, incidently, were created by Agrippa von Nettesheim, a real fifteenth century wizard. But unless you believe they have magical powers, you can make these symbols to suit your own needs just as well.

The horizontal lines can be made by pressing SHIFT-T (or any key with a horizontal line shown on its face). The vertical lines can be produced by holding down either the SHIFT or CMD key with one of the keys displaying a vertical line. The triangles shown in

some of the squares are produced by using a combination of the triangle symbols found on the * and the British pound sign keys.

This game offers you a good chance to experiment and learn by allowing you to create virtually any illustration you like.

Ask the Wizard: With Graphics

```
1 PRINTCHR$(147):POKE 53281,1:PRINTCHR$(
144)
10 REM WIZARD
20 PRINT:PRINT"                {#206}{#205}
"
22 PRINT "                {#206} {#205}
"
24 PRINT "                {#206} {#205}
"
26 PRINT "                {#206} {#205}
"
28 PRINT "                {#213}{#192 12}{#201}
"
30 PRINT "                {#202} {#202}{#201}{#2
03}{#213}{#202}{#201}{#203}{#202} {#203
} "
40 PRINT "                "
42 PRINT "                {#213}{#201} {#213
}{#201} "
44 PRINT "                >{PUR}{#209} {#20
9}{BLK}< "
46 PRINT "                {#213} {#201}
"
48 PRINT "                {#202} {#203}
"
50 PRINT "                {RED}{#203}<{#192 4}
>{#202} "
60 PRINT "                {BLK}{#201}{#202}{#203
}{#213}{#202}{#213}{#202}{#213}{#202}{#2
13}{#202}{#201} "
70 PRINT "                {#202}{#201}{#213}{#2
```

```

03} {#201} {#202} {#213} {#203} {#201} {#202} {
#213} {#203}      "
72 PRINT      "          {#202} {#213} {#203} {#
201} {#202} {#213} {#203} {#201} {#202}
"
74 PRINT      "          {#203} {#213} {#203} {
#201} {#202} {#213} {#203} {#202}      "
76 PRINT      "          {#203} {#213} {#203}
{#213} {#202} {#201}      "
78 PRINT      "          {#203} {#202} {#201
} {#202}      ":PRINT:PRINT
79 PRINT"***** T H E      W I Z A R D *****
"
90 FOR PAUSE=1 TO 2000:NEXT PAUSE:PRINTC
HR$(147)
100 PRINT"BELOW IS A SERIES OF QUESTIONS
105 PRINT"MUST CHOOSE A SYMBOL FROM THE
CHART"
110 PRINT"IN ORDER TO RECEIVE THE CORREC
T ANSWER."
115 PRINT"WHEN YOU ARE READY PRESS THE N
UMBER"
120 PRINT"NEXT TO THE SYMBOL YOU CHOSE."
:PRINT:PRINT
130 PRINT:PRINT"FIRST- CHOOSE A QUESTION
TO ASK.":PRINT:PRINT
140 PRINT" 1.CAN I MAKE THE ONE I LOVE L
OVE ME?"
142 PRINT" 2.IS THE ONE I LOVE FAITHFUL
TO ME?"
144 PRINT" 3.WILL MY LOVER MAKE ME HAPPY
?"
146 PRINT" 4.WILL THE WISH I MAKE NOW CO
ME TRUE?"
148 PRINT" 5.WILL I HAVE MANY AMOROUS AD
VENTURES?"
150 PRINT:PRINT:INPUT" WHICH? ";Q$
155 IF VAL(Q$)>5 THEN GOSUB 5500:GOTO 10
0
156 IF VAL(Q$)<1 THEN GOSUB 5500:GOTO 10
0

```

```

160 PRINTCHR$(147):GOSUB 200:GOTO 165
165 IF Q$="1"THEN GOSUB 1000
170 IF Q$="2"THEN GOSUB 2000
175 IF Q$="3"THEN GOSUB 3000
180 IF Q$="4"THEN GOSUB 4000
185 IF Q$="5"THEN GOSUB 5000
190 PRINT:PRINT:INPUT"WANT TO ASK ANOTHE
R ";X$
192 IF LEFT$(X$,1)<>"Y"THEN GOTO 6000
194 GOTO 100
200 PRINT:PRINT
210 PRINT"      {#217}  {#162}  {#217}
      {#217}      "
220 PRINT"      {#223}{#169} {#217}  {#184}
      {#217} {#209} {#209} {#217}  {#169}{#1
80}{#169}{#180}"
230 PRINT"      {#217} {#162} {#162} {#2
17}      {#217}  {#183} {#183}  "
240 PRINT" 1      {#217}2{#184} {#184} {#2
17}3      {#217}4      "
250 PRINT" {#183 10}{#217}{#183 5}{#217
}{#183 6}"
260 PRINT" {#209} {#209} {#217}      {#2
17}{#223}{#169} {#223}{#169}{#217}  {#2
09}  "
270 PRINT" {#209} {#209} {#217}  {#169}
{#180} {#217}{#223}{#169} {#223}{#169}{#
217}  {#209} {#209}  "
280 PRINT" 5      {#217}6 {#183}  {#217}7
      {#217}8      "
290 PRINT" {#183 23}"
300 PRINT"      {#217} {#169}{#180}{#169
}{#180}{#217}      {#217}      "
310 PRINT" {#162} {#162} {#217} {#183}
{#183} {#217}  {#162}  {#217} {#223}{#16
9} {#223}{#169}"
320 PRINT" {#184} {#184} {#217} {#169}{
#180}{#169}{#180}{#217}  {#184}  {#217}
      "
330 PRINT" 9      {#217}10 {#183} {#217}11
      {#217}12      "

```

```

340 PRINT"  {#183 23}"
350 PRINT"      {#217}  {#223}{#169} {#2
17} {#162}  {#162}{#217}  {#169}{#180}
"
360 PRINT"  {#209}  {#217}  {#223}{#169}
{#223}{#169}{#217}  {#184}  {#184}{#217}
  {#169}{#207}{#169}{#180}"
370 PRINT"      {#217}      {#217} {#162}
  {#162}{#217}  {#183}  {#183}  "
380 PRINT" 13  {#217}14  {#217}15  {#1
84}{#217}16  {#175}"
390 PRINT"  {#183 23}"
400 INPUT"WHICH SYMBOL DO YOU CHOOSE ";A
$:PRINT:
410 RETURN
1000 REM ANSWER SUBROUTINE
1010 IF A$="1"THENPRINT" NOTHING. YOU AR
E ALREADY ADORED.":RETURN
1020 IF A$="2"THEN PRINT"HIDE THE PAST &
DO BETTER.":RETURN
1030 IF A$="3"THEN PRINT" CHANGE YOUR HA
BITS.":RETURN
1040 IF A$="4"THEN PRINT" BECOME RICH AN
D YOU WILL BE LOVED.":RETURN
1050 IF A$="5"THEN PRINT" CRY LESS AND L
AUGH MORE." :RETURN
1060 IF A$="6"THEN PRINT"LOSE THE QUALIT
Y OF WHICH YOU HAVE TOO MUCH.":RETURN
1070 IF A$="7"THEN PRINT" DO EXACTLY AS
YOUR LOVER WISHES.":RETURN
1080 IF A$="8"THEN PRINT" BE MORE AMIABL
E WHEN ALONE TOGETHER.":RETURN
1090 IF A$="9"THEN PRINT" SUFFERING IN S
ILENCE IS A GREAT VIRTUE.":RETURN
1100 IF A$="10"THEN PRINT" PLAY HARD TO
GET. YOU WILL BE VALUED." :RETURN
1110 IF A$="11"THEN PRINT" BE FAITHFUL &
DON'T COMPLAIN.":RETURN
1120 IF A$="12"THEN PRINT" THOUGH IT IS
DIFFICULT BE DISCREET.":RETURN
1130 IF A$="13"THEN PRINT" BE CALM IN YO

```

```

UR LOVER'S COMPANY.":RETURN
1140 IF A$="14"THEN PRINT" BE FORGIVING
AND TOLERANT.":RETURN
1150 IF A$="15"THEN PRINT" BE ALWAYS THE
SAME.":RETURN
1160 IF A$="16"THEN PRINT"DO NOT REPROAC
H YOUR LOVER WHEN HE/SHE IS WRONG."
1170 RETURN
2000 REM ANSWER SUB
2010 IF A$="1"THEN PRINT"SOMETIMES FOR A
T LEAST 2 DAYS.":RETURN
2020 IF A$="2"THEN PRINT" YES.":RETURN
2030 IF A$="3"THEN PRINT" WISHES TO;BUT
CANNOT.":RETURN
2040 IF A$="4"THEN PRINT"THEY SAY YES. T
HE WIZARD SAYS NO.":RETURN
2050 IF A$="5"THEN PRINT"A QUALITY YOU H
AVE CAUSED TO BE LOST.":RETURN
2060 IF A$="6"THEN PRINT"LOSS OF LOVE--L
OSS OF FAITHFULNESS.":RETURN
2070 IF A$="7"THEN PRINT"ALWAYS HAS BEEN
& ALWAYS WILL BE.":RETURN
2080 IF A$="8"THEN PRINT"HE/SHE IS FAITH
FUL TO PLEASURE.":RETURN
2090 IF A$="9"THEN PRINT"CAN'T SAY MUCH
ABOUT IT.":RETURN
2100 IF A$="10"THEN PRINT"NOT NOW--BUT MA
YBE IN THE FUTURE.":RETURN
2110 IF A$="11"THEN PRINT"HAS BEEN BUT M
AYBE NOT AGAIN.":RETURN
2120 IF A$="12"THEN PRINT"YOU GET THE VA
LUE OF YOUR MONEY.":RETURN
2130 IF A$="13"THEN PRINT"AS MUCH AS YOU
ARE TO HIM/HER.":RETURN
2140 IF A$="14"THEN PRINT"YES. UNABLE TO
DO OTHERWISE.":RETURN
2150 IF A$="15"THEN PRINT"HE/SHE LOVES Y
OU TOO MUCH.":RETURN
2160 IF A$="16"THEN PRINT"ASK AND BELIEV
E WHAT YOU HEAR.":RETURN
3000 REM ANSWER 3 SUBROUTINE

```

```

3010 IF A$="1" THEN PRINT"FOR A COUPLE O
F DAYS MAYBE.":RETURN
3020 IF A$="2" THEN PRINT"NOT AS MUCH AS
YOU DESERVE.":RETURN
3030 IF A$="3" THEN PRINT"MUCH MORE THAN
YOU THINK.":RETURN
3040 IF A$="4" THEN PRINT"PARADISE! PARA
DISE!":RETURN
3050 IF A$="5" THEN PRINT"YES. IF LOVE
IS ENOUGH FOR YOU.":RETURN
3060 IF A$="6" THEN PRINT"YES. IF YOU A
RE CONTENT WITH LITTLE.":RETURN
3070 IF A$="7" THEN PRINT"HAPPINESS IS N
OT FOR YOU.":RETURN
3080 IF A$="8" THEN PRINT"HOME WILL GIVE
A TASTE OF HELL.":RETURN
3090 IF A$="9" THEN PRINT"AFTER THE AGE
OF FORTY ALL IS WELL.":RETURN
3100 IF A$="10" THEN PRINT"ALL THE TIME
AND EVERYWHERE.":RETURN
3110 IF A$="11" THEN PRINT"MORNING;NOON
AND NIGHT.":RETURN
3120 IF A$="12" THEN PRINT"MORE HAPPINES
S THAN WISDOM.":RETURN
3130 IF A$="13" THEN PRINT"YOUR LOVER CA
RES MORE FOR SELF THAN YOU.":RETURN
3140 IF A$="14" THEN PRINT"ONCE IN A WHI
LE.":RETURN
3150 IF A$="15" THEN PRINT"DEPENDS UPON
WHAT HAPPENS LATER.":RETURN
3160 IF A$="16" THEN PRINT"TOO MUCH TOO
SOON.":RETURN
4000 REM ANSWER 4 SUBROUTINE
4010 IF A$="1" THEN PRINT"WHEN YOU LEAST
EXPECT IT.":RETURN
4020 IF A$="2" THEN PRINT"IMPOSSIBLE DRE
AM.":RETURN
4030 IF A$="3" THEN PRINT"NOT FOR A LONG
TIME.":RETURN
4040 IF A$="4" THEN PRINT"PARTLY BUT NEV
ER COMPLETELY.":RETURN

```

```
4050 IF A$="5" THEN PRINT"YOU SHOULD NOT  
WISH FOR THIS.":RETURN  
4060 IF A$="6" THEN PRINT"IT WILL BE SOO  
N OR NEVER.":RETURN  
4070 IF A$="7" THEN PRINT"YES, AND YOU W  
ILL WISH IT DID NOT.":RETURN  
4080 IF A$="8" THEN PRINT"NEVER, UNTIL Y  
OU CHANGE WHAT YOU WISH.":RETURN  
4090 IF A$="9" THEN PRINT"YOU WILL BE RE  
WARDED AFTER A LONG WAIT.":RETURN  
4100 IF A$="10" THEN PRINT"GRADUALLY OVE  
R TIME.":RETURN  
4110 IF A$="11" THEN PRINT"ONLY IF YOU B  
ECOME LESS SELF CENTERED.":RETURN  
4120 IF A$="12" THEN PRINT"SOONER, RATHE  
R THAN LATER.":RETURN  
4130 IF A$="13" THEN PRINT"NO. THROUGH N  
O FAULT OF YOURS.":RETURN  
4140 IF A$="14" THEN PRINT"YES. BUT YOU  
WILL NO LONGER WANT IT.":RETURN  
4150 IF A$="15" THEN PRINT"THE FATE IS H  
ANGING IN THE BALANCE.":RETURN  
4160 IF A$="16" THEN PRINT"IT'S ENTIRELY  
UP TO YOU.":RETURN  
5000 REM ANSWER 5 SUBROUTINE  
5010 IF A$="1" THEN PRINT"SO MANY LOVERS  
- SO LITTLE TIME.":RETURN  
5020 IF A$="2" THEN PRINT"YES, BUT FEW H  
APPY ENDINGS.":RETURN  
5030 IF A$="3" THEN PRINT"YES, AND ALL W  
ILL END WELL.":RETURN  
5040 IF A$="4" THEN PRINT"YES, BUT ONLY  
AFTER NIGHTFALL.":RETURN  
5050 IF A$="5" THEN PRINT"YES, BUT ONLY  
BEFORE DAWN":RETURN  
5060 IF A$="6" THEN PRINT"WHEN THEY ARE  
4, TIME TO GET MORE.":RETURN  
5070 IF A$="7" THEN PRINT"THE PATH OF LO  
VE IS NOT ROSE STREWN.":RETURN  
5080 IF A$="8" THEN PRINT"ENOUGH TO CAUS  
E REGRET.":RETURN
```



```
5090 IF A$="9" THEN PRINT"THE FIRST WILL  
BE THE LAST.":RETURN  
5100 IF A$="10" THEN PRINT"THE WISE ALWA  
YS PLAN AHEAD.":RETURN  
5110 IF A$="11" THEN PRINT"TOO MANY BRIN  
GS DISGUST.":RETURN  
5120 IF A$="12" THEN PRINT"THEY WAIT AND  
YOU WILL NOT RESIST.":RETURN  
5130 IF A$="13" THEN PRINT"YOU WILL LEAR  
N MUCH ABOUT LIFE.":RETURN  
5140 IF A$="14" THEN PRINT"THE FIRST WIL  
L WHET YOUR APPETITE.":RETURN  
5150 IF A$="15" THEN PRINT"SOME BY DAY,  
BUT MANY BY NIGHT.":RETURN  
5160 IF A$="16" THEN PRINT"YOU WILL LOOK  
, BUT SELDOM FIND.":RETURN  
5500 REM ERROR TRAP  
5510 PRINT"{RVS}THE CHOICE IS 1-5.{ROFF}  
":RETURN  
6000 REM END  
6010 PRINT"GAME ENDED"  
6020 END
```

In the listing of *Ask the Wizard: With Graphics*, the keyboard graphics characters which comprise the illustrations are represented by their ASCII equivalents (see Appendix F).

Chapter 15

MAGIC CARDS

In this chapter we will construct *Magic Cards*, a game complete with graphics. The graphics will evolve as part of the game, rather than being added as an enhancement as in *Ask the Wizard*.

Magic Cards is similar to tarot card fortune telling. When reading your cards, a tarot reader makes you attach special meaning to the messages associated with each card drawn. *Magic Cards* uses random number selection to pick four cards in response to the player's questions.

In an actual reading, the human fortune teller synthesizes the often contradictory messages each card presents. The computer program can act similarly if it is sufficiently complex. Although *Magic Cards* does not synthesize multiple-card messages, it will offer the information necessary to construct a sophisticated card-reading game. And as the game stands, it is a fascinating way to spend time, and it often seems to read your mind because it leaves enough to your imagination to provide meaningful answers to your questions.

The structure of *Magic Cards* is similar in many ways to *Ask the Wizard*. The beginning of the game presents instructions, a set of questions, and subroutines which include responses to the questions. However in *Magic Cards*, four separate random numbers are selected, and there are 52 subroutines (one for each card in a standard deck). Each card is assigned a message to be displayed, but the card selection itself is four draws of 52, with replacement. That is, the same card could be drawn more than one time. The card messages are typical of the meanings assigned to cards in a tarot deck.

Line 1 shows the usual screen clearing character string (147). The first POKE sets the screen color to white, as in the other games, but this time there is a POKE for border color.

```
PRINT CHR$(147):POKE53281,1;  
POKE53281,10:PRINT CHR$(144)
```

The POKE 53280,10 gives us a pretty pink border, which will look good with the cards we will construct later. The CHR\$(144) gives us a black cursor, so the printing is in black throughout the game, except when the program changes the color.

Lines 10-90 provide instructions and display the questions very much like *Ask the Wizard*. Line 100 begins the routine to select random numbers for each of the four cards to be displayed. This process is exactly like shuffling a deck of cards and drawing a single card, reshuffling and drawing another card, and so on until four cards have been drawn. The variables are named F1, F2, F3, and F4. The statements in lines 110-140 contain the algorithm for selection of a number from 1 to 52, each number representing a card in a standard deck. Lines 155-170 are the by-now-familiar prompt for another question and the provision for a Y(es) versus non-Y response. If the player does not want to play again, the program is directed to line 20000, which is the end routine. If the player does want to go around again, the program goes to line 75, which displays the questions again.

Illustrating Cards with Subroutines

For the time being, let's ignore the tests for direction to the subroutines, and go about constructing the card images. Obviously, if you are randomly drawing cards (even when spiritual powers control each draw), you will want to tap the graphic capabilities of the Commodore 64 by displaying those cards selected. The card images are contained in lines 10000-20000, with each subroutine being allocated a range of 100. The first subroutine begins on line 10000, and includes lines 10010, 10020, 10030, 10040, 10050, 10060, 10070, 10080, 10090, 10100, and 10110. If you look at the listing, each subroutine stands out with its display of a card image.

We produced the pictures by using SHIFT-O, then CMD-T three times, then SHIFT-P to make the top edge of the card. The next line should make the symbols on the card. For the deuce of spades, we press the number 2 key, move over one space, then press SHIFT-A. Complete the line by spacing once, then pressing 2, and finally pressing SHIFT-N for the border of the card. Line 10050 consists of just SHIFT-H and SHIFT-N, because there is no symbol between the edges of the card and no other characters on the line. Line 10060 is the same as 10050, except that after completing the border of the card, you need to space once and then type the message, finishing with a closed quote. Line 10070 is the same as 10060, and you finish the message begun on line 10060 if necessary; if not, you simply close the quote after typing SHIFT-N for the edge of the card. Line 10080 is exactly like 10040, but the last line (10090) is the opposite of line 10030. Line 10090 begins with SHIFT-L immediately after the quote, then three SHIFT- Ps, finishing with SHIFT-@.

Continue making each card as you did the first, being careful that each is the same size (seven lines by seven columns). It might help to get a deck of cards and lay them out. The face cards offer a different challenge, and you may be able to improve upon those you see here.

Incidentally, it is not necessary to completely retype each line in a program on the Commodore 64. The screen editor allows you to copy lines which are the same or nearly the same just by retyping the new line number over the old one. For example, line 10130 can easily be recopied from line 10030 by moving the cursor up to line 10030 and typing 10130 on top of 10030 then pressing RETURN. You can also copy lines which are similar, such as line 10040, and make only the changes necessary to line 10140 by changing the deuce of spades to a three of spades. This saves a lot of typing and will make the code writing a lot simpler.

In a program with so many subroutines, it is helpful to put any repetitious elements into additional subroutines. Unfortunately, each card is different. It is possible to save some code, however, by placing the top and bottom of each card in separate subroutines, which would save two lines for each card. If you decide to do this, each subroutine here would call two separate subroutines (for the

top and the bottom of the cards). This means that you will have nested subroutines. Such a procedure would save 104 lines (2 times 52). The program listing in this chapter displays the full card images so you can more readily understand their construction. If you feel more comfortable using the program as it is shown here, do so.

The lines immediately following the bottom edge of each card

```
10100 FOR P=1 TO 2000:NEXT P
```

are responsible for the pause loop. This sets up a pause equivalent to 2000 cycles, or about 30 seconds. This is necessary to allow the player time to read the message and view the card. After 2000 cycles, the program goes to the next line, which is:

```
10110 R=R+1:RETURN
```

This line increments counter R, which keeps track of the number of cards shown on the screen and RETURNS to the start of the subroutine.

Selecting Cards with Random Numbers

The integral part of *Magic Cards* is contained in lines 200-710. These lines are tests for the selection of numbers 1 to 52, which represent the cards in the deck.

```
200 IF F1=1 OR F2=1 OR F3=1 OR F4=1  
THEN GOSUB 10030:IF R=>4 THEN  
GOTO 160
```

Since there were four random numbers selected (F1 to F4), the test looks for any of them equaling one. If any of the four random numbers selected was one, the program moves to subroutine 10030. Because the last card selected might well be a number four, there must be another test to see if the counter has been incremented so that it now totals four. This would mean that four cards have already been displayed.

In the event that four cards have been displayed (only four random numbers were selected by the program), the program moves to line 160, which contains the prompt "DO YOU HAVE ANOTHER QUESTION."

Each line from 200 to 710 tests to determine what value is taken by F1, F2, F3, or F4. Since there are 52 possible values, there are 52 IF statements. The possibility of F1, F2, F3, or F4 taking on each value is considered in each program line. If any of them meet the criterion, then the program is directed to the appropriate subroutine. Each subroutine (and each random number selection) corresponds to each card in a 52 card deck.

The END routine wishes the player "LOTS OF LUCK!" and the program ENDS. But this is not the end of the ideas that can come from the use of cards in computer games. What about poker? Black-jack? Tarot? War? The possibilities are almost endless.

Magic Cards

```
1 PRINTCHR$(147):POKE 53281,1:POKE53280,
10:PRINTCHR$(144)
10 *** REM MAGIC CARDS ***
20 PRINT"TO PLAY MAGIC CARDS, YOU TYPE A
NO.OF A"
30 PRINT"PROBLEM TO WHICH YOU SEEK A SOL
UTION"
40 PRINT"FROM THE LIST BELOW. PRESS RET
URN"
50 PRINT"WHEN YOU HAVE COMPLETED YOUR EN
TRY."
```

```

60 PRINT"YOU WILL SEE 4 CARDS, EACH WITH
A"
70 PRINT"MESSAGE WHICH WILL HELP YOU."
75 PRINT:PRINT"1 WILL I WIN HIM/HER?"
77 PRINT "2 WILL I FIND A JOB?"
78 PRINT "3 WILL I BECOME HANDSOME/PRET
TY?"
79 PRINT "4 HOW MAY I BECOME RICH?"
80 PRINT "5 HOW CAN I DEFEAT MY ENEMIES
?"
81 PRINT "6 WHAT DOES THE NEXT MONTH HO
LD?"
82 PRINT "7 WHAT DOES THE NEXT YEAR HOL
D?"
83 PRINT "8 WILL HE/SHE CALL ME AGAIN?"
84 PRINT "9 WILL I BE PROMOTED?"
85 PRINT "10 IS {#198 7} TO BE TRUSTED?"
86 PRINT"11 WILL I SOLVE MY PROBLEM?"
87 PRINT "12 WILL HE/SHE AND I RECONCILE
?"
88 PRINT "13 WHAT WILL MY LIFE BE?"
90 PRINT:INPUT"WHICH NUMBER ";Q
100 REM MAGIC CARDS
105 R=0
110 F1=INT(RND(1)*52)+1
120 F2=INT(RND(1)*52)+1
130 F3=INT(RND(1)*52)+1
140 F4=INT(RND(1)*52)+1
155 PRINT"{CLR}": GOTO 195
160 PRINT"{CLR}":INPUT"DO YOU HAVE ANOTH
ER QUESTION ";X$
165 IF LEFT$(X$,1)="Y" THEN GOTO 75
170 IF LEFT$(X$,1)<>"Y" THEN GOTO 20000
195 REM TESTS
200 IF F1=1 OR F2=1ORF3=1ORF4=1THEN GOSU
B10030:IF R>4 THEN GOTO 160
210 IF F1=2 OR F2=2ORF3=2ORF4=2THEN GOSU
B10130:IF R>4THEN GOTO 160
220 IF F1=3 OR F2=3ORF3=3ORF4=3THEN GOSU
B10230:IF R>4THEN GOTO 160
230 IF F1=4 OR F2=4ORF3=4ORF4=4THEN GOSU

```


B10330:IF R> 4THEN GOTO 160
240 IF F1=5 OR F2=5ORF3=5ORF4=5THEN GOSU
B10430:IF R>4THEN GOTO 160
250 IF F1=6 OR F2=6ORF3=6ORF4=6THEN GOSU
B10530:IF R>4THEN GOTO 160
260 IF F1=7 OR F2=7ORF3=7ORF4=7THEN GOSU
B10630:IFR>4THEN GOTO 160
270 IF F1=8 OR F2=8ORF3=8ORF4=8THEN GOSU
B10730:IFR>4 THEN GOTO 160
280 IF F1=9 OR F2=9ORF3=9ORF4=9THEN GOSU
B10830:IFR>4THEN GOTO 160
290 IF F1=10OR F2=10ORF3=10ORF4=10THEN G
OSUB10930:IFR>4THEN GOTO 160
300 IF F1=11OR F2=11ORF3=11ORF4=11THEN G
OSUB11030:IFR>4THEN GOTO 160
310 IF F1=12OR F2=12ORF3=12ORF4=12THEN G
OSUB11130:IFR>4 THEN GOTO 160
320 IF F1=13OR F2=13ORF3=13ORF4=13THEN G
OSUB11230:IFR>R THEN GOTO 160
330 IF F1=14OR F2=14ORF3=14ORF4=14THEN G
OSUB12030:IFR>4 THEN GOTO 160
340 IF F1=15OR F2=15ORF3=15ORF4=15THEN G
OSUB12130:IFR>4 THEN 160
350 IF F1=16OR F2=16ORF3=16ORF4=16THEN G
OSUB12230:IFR>4 THEN 160
360 IF F1=17OR F2=17ORF3=17ORF4=17THEN G
OSUB12330:IF R>R THEN 160
370 IF F1=18OR F2=18ORF3=18ORF4=18THEN G
OSUB12430:IF R>4 THEN 160
380 IF F1=19OR F2=19ORF3=19ORF4=19THEN G
OSUB12630:IFR>4 THEN 160
390 IF F1=20OR F2=20ORF3=20ORF4=20THEN G
OSUB12730:IF R>4 THEN 160
400 IF F1=21OR F2=21ORF3=21ORF4=21THEN G
OSUB12830:IFR>4 THEN 160
410 IF F1=22OR F2=22ORF3=22ORF4=22THEN G
OSUB12930:IFR>4THEN 160
420 IF F1=23OR F2=23ORF3=23ORF4=23THEN G
OSUB13030:IFR>4 THEN 160
430 IF F1=24OR F2=24ORF3=24ORF4=24THEN G
OSUB13130:IFR>4 THEN 160

440 IF F1=25OR F2=25ORF3=25ORF4=25THEN G
OSUB13230:IF R>4 THEN 160
450 IF F1=26OR F2=26ORF3=26ORF4=26THEN G
OSUB13330:IFR>4 THEN 160
460 IF F1=27OR F2=27ORF3=27ORF4=27THEN G
OSUB14020:IFR>4 THEN 160
470 IF F1=28OR F2=28ORF3=28ORF4=28THEN G
OSUB14120:IFR>4 THEN 160
480 IF F1=29OR F2=29ORF3=29ORF4=29THEN G
OSUB14220:IFR>4 THEN 160
490 IF F1=30OR F2=30ORF3=30ORF4=30THEN G
OSUB14320:IFR>4 THEN 160
500 IF F1=31OR F2=31ORF3=31ORF4=31THEN G
OSUB14420:IFR>4THEN 160
510 IF F1=32OR F2=32ORF3=32ORF4=32THEN G
OSUB14620:IFR>4 THEN 160
520 IF F1=33OR F2=33ORF3=33ORF4=33THEN G
OSUB14720:IFR>4 THEN 160
530 IF F1=34OR F2=34ORF3=34ORF4=34THEN G
OSUB14820:IFR>4 THEN 160
540 IF F1=35OR F2=35ORF3=35ORF4=35THEN G
OSUB14920:IF R>4 THEN 160
550 IF F1=36OR F2=36ORF3=36ORF4=36THEN G
OSUB15020:IF R>4 THEN 160
560 IF F1=37OR F2=37ORF3=37ORF4=37THEN G
OSUB15120:IF R>4 THEN 160
570 IF F1=38OR F2=38ORF3=38ORF4=38THEN G
OSUB15220:IF R>4 THEN 160
580 IF F1=39OR F2=39ORF3=39ORF4=39THEN G
OSUB15320:IFR>4 THEN160
590 IF F1=40OR F2=40ORF3=40ORF4=40THEN G
OSUB16020:IFR>4 THEN 160
600 IF F1=41OR F2=41ORF3=41ORF4=41THEN G
OSUB16120:IFR>4 THEN 160
610 IF F1=42OR F2=42ORF3=42ORF4=42THEN G
OSUB16220:IFR>4 THEN 160
620 IF F1=43OR F2=43ORF3=43ORF4=43THEN G
OSUB16320:IF R>4 THEN 160
630 IF F1=44OR F2=44ORF3=44ORF4=44THEN G
OSUB16420:IF R>4 THEN 160
640 IF F1=45OR F2=45ORF3=45ORF4=45THEN G

```

CSUB16620:IF R>4 THEN 160
650 IF F1=46OR F2=46ORF3=46ORF4=46THEN G
OSUB16720:IF R>4 THEN 160
660 IF F1=47OR F2=47ORF3=47ORF4=47THEN G
OSUB16820:IF R>4 THEN160
670 IF F1=48OR F2=48ORF3=48ORF4=48THEN G
OSUB16920:IF R>4 THEN 160
680 IF F1=49OR F2=49ORF3=49ORF4=49THEN G
OSUB17020:IF R>4 THEN 160
690 IF F1=50OR F2=50ORF3=50ORF4=50THEN G
OSUB17120:IF R>4 THEN 160
700 IF F1=51OR F2=51ORF3=51ORF4=51THEN G
OSUB17220:IF R>4 THEN 160
710 IF F1=52OR F2=52ORF3=52ORF4=52THEN G
OSUB17320:IF R>4 THEN 160
1000 REM
1010 GOTO 160
10000 REM CARDS SUBROUTINES
10010 PRINTCHR$(147)
10020 PRINT:PRINT
10030 PRINT" [ ] "
10040 PRINT"| 2 ♠ 2 |"
10050 PRINT"|          | YOU WILL BE "
10060 PRINT"|          | PROTECTED OR"
10070 PRINT"|          | RECEIVE GIFTS"
10080 PRINT"| 2 ♠ 2 |"
10090 PRINT" [ ] "
10100 R=R+1:FORP=1 TO 2000:NEXT P
10110 RETURN
10120 REM
10130 PRINT" [ ] "
10140 PRINT"| 3 ♠ 3 |"
10150 PRINT"|          | I SEE A LOVER'S"
10160 PRINT"|    ♠    | FLIGHT"
10170 PRINT"|          |"
10180 PRINT"| 3 ♠ 3 |"
10190 PRINT" [ ] "
10200 R=R+1:FORP=1 TO 2000:NEXT P
10210 RETURN

```

```

10230 PRINT" [ ] "
10240 PRINT" | 4♠ ♠4 | "
10250 PRINT" | | YOU WILL RECEIVE"
10260 PRINT" | | BAD NEWS"
10270 PRINT" | | "
10280 PRINT" | 4♠ ♠4 | "
10290 PRINT" [ ] "
10300 R=R+1:FORP=1 TO 2000:NEXT P
10310 RETURN
10330 PRINT" [ ] "
10340 PRINT" | 5♠ ♠5 | "
10350 PRINT" | | YOU WILL MEET WITH"
10360 PRINT" | ♠ | MISFORTUNE"
10370 PRINT" | | "
10380 PRINT" | 5♠ ♠5 | "
10390 PRINT" [ ] "
10400 R=R+1:FORP=1 TO 2000:NEXT P
10410 RETURN
10430 PRINT" [ ] "
10440 PRINT" | 6♠ ♠6 | "
10450 PRINT" | | A GOOD VOYAGE WILL"
10460 PRINT" | ♠ ♠ | BE MADE SOON"
10470 PRINT" | | "
10480 PRINT" | 6♠ ♠6 | "
10490 PRINT" [ ] "
10500 R=R+1:FORP=1 TO 2000:NEXT P
10510 RETURN
10530 PRINT" [ ] "
10540 PRINT" | 7♠ ♠7 | "
10550 PRINT" | | GOOD FORTUNE"
10560 PRINT" | ♠ ♠ | "
10570 PRINT" | ♠ | "
10580 PRINT" | 7♠ ♠7 | "
10590 PRINT" [ ] "
10605 R=R+1:FORP=1 TO 2000:NEXT P
10610 RETURN

```

```

10630 PRINT" [ ] "
10640 PRINT"| 8♠ ♠8 | "
10650 PRINT"|   ♠ | THERE WILL BE"
10660 PRINT"|   ♠ ♠ | A SCANDAL FOR A"
10670 PRINT"|   ♠ | WOMAN"
10680 PRINT"| 8♠ ♠8 | "
10690 PRINT" [ ] "
10705 R=R+1:FORP=1 TO 2000:NEXT P
10710 RETURN
10730 PRINT" [ ] "
10740 PRINT"| 9♠ ♠9 | "
10750 PRINT"|   ♠ ♠ | THIS CARD IS A"
10760 PRINT"|   ♠ | BAD OMEN"
10770 PRINT"|   ♠ ♠ | "
10780 PRINT"| 9♠ ♠9 | "
10790 PRINT" [ ] "
10805 R=R+1:FORP=1 TO 2000:NEXT P
10810 RETURN
10830 PRINT" [ ] "
10840 PRINT"| 10♠ ♠10 | "
10850 PRINT"|   ♠ | A FRIEND WILL BE"
10860 PRINT"|   ♠ ♠ | A TRAITOR"
10870 PRINT"|   ♠ ♠ | "
10880 PRINT"| 10♠♠♠10 | "
10890 PRINT" [ ] "
10905 R=R+1:FORP=1 TO 2000:NEXT P
10910 RETURN
10930 PRINT" [ ] "
10940 PRINT"|  A   A | "
10950 PRINT"|   | EITHER GREAT"
10960 PRINT"|   ♠ | RICHES OR GREAT"
10970 PRINT"|   | MISERY"
10980 PRINT"|  A   A | "
10990 PRINT" [ ] "
11005 R=R+1:FORP=1 TO 2000:NEXT P
11010 RETURN

```

```

11030 PRINT" [ ] "
11040 PRINT"| J♣ J | "
11050 PRINT"| *** | SOMEONE WILL MAKE"
11060 PRINT"| .. | AN INDISCREET"
11070 PRINT"|  ) | INQUIRY"
11080 PRINT"| J ♣ J | "
11090 PRINT"| [ ] "
11105 R=R+1:FORP=1 TO 2000:NEXT P
11110 RETURN
11130 PRINT" [ ] "
11140 PRINT"| Q♣ Q | "
11150 PRINT"| ##### | I SEE A WIDOW"
11160 PRINT"| .. \ | "
11170 PRINT"|  ) \ | "
11180 PRINT"| Q ♣ Q | "
11190 PRINT"| [ ] "
11205 R=R+1:FORP=1 TO 2000:NEXT P
11210 RETURN
11230 PRINT" [ ] "
11240 PRINT"| K♣ K | "
11250 PRINT"| ***** | I SEE A"
11260 PRINT"|  ) \ | PROFESSIONAL MAN"
11270 PRINT"|  ) \ | "
11280 PRINT"| K ♣ K | "
11290 PRINT"| [ ] "
12005 R=R+1:FORP=1 TO 2000:NEXT P
12010 RETURN
12030 PRINT" [ ] "
12040 PRINT"| A A | "
12050 PRINT"|  ) | CALAMITIES ABOUND"
12060 PRINT"|  ) | "
12070 PRINT"|  ) | "
12080 PRINT"| A A | "
12090 PRINT"| [ ] "
12105 R=R+1:FORP=1 TO 2000:NEXT P
12110 RETURN

```

```

12130 PRINT" [ ] "
12140 PRINT"| 2 ♣ 2 | "
12150 PRINT"|          | MANY SMALL"
12160 PRINT"|          | DISAPPOINTMENTS"
12170 PRINT"|          | "
12180 PRINT"| 2 ♣ 2 | "
12190 PRINT'| [ ] '
12205 R=R+1:FORP=1 TO 2000:NEXT P
12210 RETURN
12230 PRINT" [ ] "
12240 PRINT"| 3 ♣ 3 | "
12250 PRINT"|          | YOU MAY WORK WITH"
12260 PRINT"|  ♣  | ANOTHER. BEWARE!"
12270 PRINT"|          | "
12280 PRINT"| 3 ♣ 3 | "
12290 PRINT'| [ ] '
12305 R=R+1:FORP=1 TO 2000:NEXT P
12310 RETURN
12330 PRINT" [ ] "
12340 PRINT"| 4♣ ♣4 | "
12350 PRINT"|          | UNEXPECTED GOOD"
12360 PRINT"|          | FORTUNE SOON"
12370 PRINT"|          | "
12380 PRINT"| 4♣ ♣4 | "
12390 PRINT'| [ ] '
12405 R=R+1:FORP=1 TO 2000:NEXT P
12410 RETURN
12430 PRINT" [ ] "
12440 PRINT"| 5♣ ♣5 | "
12450 PRINT"|          | EXPECT BRIEF"
12460 PRINT"|  ♣  | FINANCIAL SUCCESS"
12470 PRINT"|          | "
12480 PRINT"| 5♣ ♣5 | "
12490 PRINT'| [ ] '
12605 R=R+1:FORP=1 TO 2000:NEXT P
12610 RETURN

```

```

12630 PRINT" [ ] "
12640 PRINT"| 6# #6 | "
12650 PRINT"| [ ] | A FRIEND WILL"
12660 PRINT"| # # | BETRAY YOU"
12670 PRINT"| [ ] | "
12680 PRINT"| 6# #6 | "
12690 PRINT" [ ] "
12705 R=R+1:FORP=1 TO 2000:NEXT P
12710 RETURN
12730 PRINT" [ ] "
12740 PRINT"| 7# #7 | "
12750 PRINT"| [ ] | I SEE A DARK CHILD"
12760 PRINT"| # # | IN YOUR LIFE"
12770 PRINT"| # | "
12780 PRINT"| 7# #7 | "
12790 PRINT" [ ] "
12805 R=R+1:FORP=1 TO 2000:NEXT P
12810 RETURN
12830 PRINT" [ ] "
12840 PRINT"| 8# #8 | "
12850 PRINT"| # | THERE WILL BE"
12860 PRINT"| # # | DOMESTIC DISPUTES"
12870 PRINT"| # | "
12880 PRINT"| 8# #8 | "
12890 PRINT" [ ] "
12905 R=R+1:FORP=1 TO 2000:NEXT P
12910 RETURN
12930 PRINT" [ ] "
12940 PRINT"| 9# #9 | "
12950 PRINT"| # # | BAD NEWS WILL COME"
12960 PRINT"| # | FOR YOU SOON"
12970 PRINT"| # # | "
12980 PRINT"| 9# #9 | "
12990 PRINT" [ ] "
13005 R=R+1:FORP=1 TO 2000:NEXT P
13010 RETURN

```



```

13030 PRINT" [ ] "
13040 PRINT" 10# #10 "
13050 PRINT" | # | DIFFICULTIES"
13060 PRINT" | # # | AHEAD"
13070 PRINT" | # # | "
13080 PRINT" 10#####10 "
13090 PRINT" [ ] "
13105 R=R+1:FORP=1 TO 2000:NEXT P
13110 RETURN
13130 PRINT" [ ] "
13140 PRINT" | J# J | "
13150 PRINT" | *** | YOU WILL SEARCH"
13160 PRINT" | .. | FOR A MATE"
13170 PRINT" | _ \ | "
13180 PRINT" | J #J | "
13190 PRINT" [ ] "
13205 R=R+1:FORP=1 TO 2000:NEXT P
13210 RETURN
13230 PRINT" [ ] "
13240 PRINT" | Q# Q | "
13250 PRINT" | * \ | YOU WILL HAVE A"
13260 PRINT" | .. \ | BOUNTIFUL HARVEST"
13270 PRINT" | ●● \ | "
13280 PRINT" | Q #Q | "
13290 PRINT" [ ] "
13305 R=R+1:FORP=1 TO 2000:NEXT P
13310 RETURN
13330 PRINT" [ ] "
13340 PRINT" | K# K | "
13350 PRINT" | **** | FAVORABLE TIDINGS"
13360 PRINT" | .. \ | "
13370 PRINT" | -- \ | "
13380 PRINT" | K #K | "
13390 PRINT" [ ] "
14005 R=R+1:FORP=1 TO 2000:NEXT P
14010 RETURN
14020 PRINT"

```

```

14030 PRINT" [ ] "
14040 PRINT" | A   A | "
14050 PRINT" |           | THIS CARD WILL"
14060 PRINT" |   ♦   | BRING YOU ALL YOU"
14070 PRINT" |           | WISH FOR"
14080 PRINT" | A   A | "
14090 PRINT" [ ] "
14100 PRINT"
14105 R=R+1:FORP=1 TO 2000:NEXT P
14110 RETURN
14120 PRINT" [ ] "
14130 PRINT" | 2 ♦ 2 | "
14140 PRINT" |           | "
14150 PRINT" |           | YOU IMAGINE MANY"
14160 PRINT" |           | OF YOUR TROUBLES"
14170 PRINT" | 2 ♦ 2 | "
14180 PRINT" [ ] "
14190 PRINT" "
14200 PRINT"
14205 R=R+1:FORP=1 TO 2000:NEXT P
14210 RETURN
14220 PRINT"
14230 PRINT" [ ] "
14240 PRINT" | 3 ♦ 3 | "
14250 PRINT" |           | THE COLLABORATION"
14260 PRINT" |   ♦   | WILL BE USEFUL"
14270 PRINT" |           | "
14280 PRINT" | 3 ♦ 3 | "
14290 PRINT" [ ] "
14300 PRINT"
14305 R=R+1:FORP=1 TO 2000:NEXT P
14310 RETURN
14320 PRINT"
14330 PRINT" [ ] "
14340 PRINT" | 4♦ ♦4 | "
14350 PRINT" |           | PLEASANT NEWS FROM"
14360 PRINT" |           | A LADY"
14370 PRINT" |           | "
14380 PRINT" | 4♦ ♦4 | "
14390 PRINT" [ ] "
14400 PRINT"

```

```

14405 R=R+1:FORP=1 TO 2000:NEXT P
14410 RETURN
14420 PRINT"
14430 PRINT" [ 5♦ ♦5 ] "
14440 PRINT" | 5♦ ♦5 | "
14450 PRINT" |           | REASON CONQUERS"
14460 PRINT" |     ♦     | FORTUNE"
14470 PRINT" |           | "
14480 PRINT" | 5♦ ♦5 | "
14490 PRINT" [           ] "
14500 PRINT"
14605 R=R+1:FORP=1 TO 2000:NEXT P
14610 RETURN
14620 PRINT"
14630 PRINT" [ 6♦ ♦6 ] "
14640 PRINT" | 6♦ ♦6 | "
14650 PRINT" |           | BE CAUTIOUS AT"
14660 PRINT" |     ♦ ♦     | THE PRESENT"
14670 PRINT" |           | "
14680 PRINT" | 6♦ ♦6 | "
14690 PRINT" [           ] "
14700 PRINT"
14705 R=R+1:FORP=1 TO 2000:NEXT P
14710 RETURN
14720 PRINT"
14730 PRINT" [ 7♦ ♦7 ] "
14740 PRINT" | 7♦ ♦7 | "
14750 PRINT" |           | YOUR POSITION WILL"
14760 PRINT" |     ♦ ♦     | IMPROVE NOW"
14770 PRINT" |           | "
14780 PRINT" | 7♦ ♦7 | "
14790 PRINT" [           ] "
14800 PRINT"
14805 R=R+1:FORP=1 TO 2000:NEXT P
14810 RETURN

```

```

14820 PRINT"
14830 PRINT" [ ] "
14840 PRINT" | 8♦ ♦8 | "
14850 PRINT" | ♦ | I SEE A YOUNG MAN"
14860 PRINT" | ♦ ♦ | AND A DARK WOMAN"
14870 PRINT" | ♦ | "
14880 PRINT" | 8♦ ♦8 | "
14890 PRINT" [ ] "
14900 PRINT"
14905 R=R+1:FORP=1 TO 2000:NEXT P
14910 RETURN
14920 PRINT"
14930 PRINT" [ ] "
14940 PRINT" | 9♦ ♦9 | "
14950 PRINT" | ♦ ♦ | THIS REINFORCES"
14960 PRINT" | ♦ | THE OTHER CARDS"
14970 PRINT" | ♦ ♦ | IN THIS HAND"
14980 PRINT" | 9♦ ♦9 | "
14990 PRINT" [ ] "
15000 PRINT"
15005 R=R+1:FORP=1 TO 2000:NEXT P
15010 RETURN
15020 PRINT"
15030 PRINT" [ ] "
15040 PRINT" | 10♦ ♦10 | "
15050 PRINT" | ♦ | I SEE A HOUSE"
15060 PRINT" | ♦ ♦ | "
15070 PRINT" | ♦ ♦ | "
15080 PRINT" | 10♦♦♦10 | "
15090 PRINT" [ ] "
15100 PRINT"
15105 R=R+1:FORP=1 TO 2000:NEXT P
15110 RETURN
15120 PRINT"
15130 PRINT" [ ] "
15140 PRINT" | J♦ J | "
15150 PRINT" | *** | I SEE A YOUNG"
15160 PRINT" | .. | PERSON"
15170 PRINT" | _ \ | "
15180 PRINT" | J ♦J | "
15190 PRINT" [ ] "

```

```

15200 PRINT"
15205 R=R+1:FORP=1 TO 2000:NEXT P
15210 RETURN
15220 PRINT"
15230 PRINT" 

|    |   |
|----|---|
|    |   |
| Q♦ | Q |
|    |   |
|    |   |
|    |   |
|    |   |

 "
15240 PRINT" | Q♦ Q | "
15250 PRINT" | 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 | GREAT RICHES"
15260 PRINT" | .. \ | "
15270 PRINT" | 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 \ | "
15280 PRINT" | Q ♦Q | "
15290 PRINT" 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 "
15300 PRINT"
15305 R=R+1:FORP=1 TO 2000:NEXT P
15310 RETURN
15320 PRINT"
15330 PRINT" 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 "
15340 PRINT" | K♦ K | "
15350 PRINT" | **** | I SEE A DARK MAN"
15360 PRINT" | /OO\ | "
15370 PRINT" | <O> \ | "
15380 PRINT" | K ♦K | "
15390 PRINT" 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 "
15400 PRINT"
16005 R=R+1:FORP=1 TO 2000:NEXT P
16010 RETURN
16020 PRINT"
16030 PRINT" 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 "
16040 PRINT" | A A | "
16050 PRINT" | | A STRONG WILL;"
16060 PRINT" | ♦ | UNCHANGEABLE LAW"
16070 PRINT" | | "
16080 PRINT" | A A | "
16090 PRINT" 

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

 "
16100 PRINT"
16105 R=R+1:FORP=1 TO 2000:NEXT P
16110 RETURN

```

```

16120 PRINT"
16130 PRINT"      "
16140 PRINT" [ ] "
16150 PRINT" | 2 ♣ 2 | FAVORABLE BUSINESS"
16160 PRINT" |      | AND PLEASURE"
16170 PRINT" |      | "
16180 PRINT" |      | "
16190 PRINT" | 2 ♣ 2 | "
16200 PRINT" [ ] "
16205 R=R+1:FORP=1 TO 2000:NEXT P
16210 RETURN
16220 PRINT"
16230 PRINT" [ ] "
16240 PRINT" | 3 ♣ 3 | "
16250 PRINT" |      | UNEXPECTEDLY YOU"
16260 PRINT" |  ♣  | WILL ADVANCE"
16270 PRINT" |      | "
16280 PRINT" | 3 ♣ 3 | "
16290 PRINT" [ ] "
16300 PRINT"
16305 R=R+1:FORP=1 TO 2000:NEXT P
16310 RETURN
16320 PRINT"
16330 PRINT" [ ] "
16340 PRINT" | 4 ♣ ♣4 | "
16350 PRINT" |      | EVENTS CONTRARY TO"
16360 PRINT" |      | YOUR DESIRES"
16370 PRINT" |      | "
16380 PRINT" | 4 ♣ ♣4 | "
16390 PRINT" [ ] "
16400 PRINT"
16405 R=R+1:FORP=1 TO 2000:NEXT P
16410 RETURN
16420 PRINT"
16430 PRINT" [ ] "
16440 PRINT" | 5 ♣ ♣5 | "
16450 PRINT" |      | HAPPINESS, SUCCESS"
16460 PRINT" |  ♣  | & GIFTS ARE YOURS"
16470 PRINT" |      | "
16480 PRINT" | 5 ♣ ♣5 | "
16490 PRINT" [ ] "

```

```

16500 PRINT"
16505 R=R+1:FORP=1 TO 2000:NEXT P
16510 RETURN
16620 PRINT"
16630 PRINT" [ ] "
16640 PRINT" | 6# #6 | "
16650 PRINT" |      | PLEASANT MEMORIES"
16660 PRINT" |  #  # | ARE FORETOLD"
16670 PRINT" |      | "
16680 PRINT" | 6# #6 | "
16690 PRINT" [ ] "
16700 PRINT"
16705 R=R+1:FORP=1 TO 2000:NEXT P
16710 RETURN
16720 PRINT"
16730 PRINT" [ ] "
16740 PRINT" | 7# #7 | "
16750 PRINT" |      | I SEE A FAIR CHILD"
16760 PRINT" |  #  # | & A RESOLUTION."
16770 PRINT" |      | "
16780 PRINT" | 7# #7 | "
16790 PRINT" [ ] "
16800 PRINT"
16805 R=R+1:FORP=1 TO 2000:NEXT P
16810 RETURN
16820 PRINT"
16830 PRINT" [ ] "
16840 PRINT" | 8# #8 | "
16850 PRINT" |      | I SEE MARRIAGE"
16860 PRINT" |  #  # | FOR A FAIR WOMAN"
16870 PRINT" |      | "
16880 PRINT" | 8# #8 | "
16890 PRINT" [ ] "
16900 PRINT"
16905 R=R+1:FORP=1 TO 2000:NEXT P
16910 RETURN

```



```

16920 PRINT"
16930 PRINT" [ ] "
16940 PRINT"| 9# #9 | "
16950 PRINT"| # # | A PROPITIOUS"
16960 PRINT"| # | FATE FOR MILITARY"
16970 PRINT"| # # | PERSONS"
16980 PRINT"| 9# #9 | "
16990 PRINT" [ ] "
17000 PRINT"
17005 R=R+1:FORP=1 TO 2000:NEXT P
17010 RETURN
17020 PRINT"
17030 PRINT" [ ] "
17040 PRINT"| 10# #10 | "
17050 PRINT"| # | I SEE A GOOD"
17060 PRINT"| # # | & LONG MARRIAGE"
17070 PRINT"| # # | "
17080 PRINT"| 10####10 | "
17090 PRINT" [ ] "
17100 PRINT"
17105 R=R+1:FORP=1 TO 2000:NEXT P
17110 RETURN
17120 PRINT"
17130 PRINT" [ ] "
17140 PRINT"| J# J | "
17150 PRINT"| *** | BENEFICENCE FOR"
17160 PRINT"| . . | YOU AND YOURS"
17170 PRINT"| _ \ | "
17180 PRINT"| J #J | "
17190 PRINT" [ ] "
17200 PRINT"
17205 R=R+1:FORP=1 TO 2000:NEXT P
17210 RETURN
17220 PRINT"
17230 PRINT" [ ] "
17240 PRINT"| Q# Q | "
17250 PRINT"| XXXX | I SEE A WOMAN"
17260 PRINT"| /.. \ | OF UNCERTAIN"
17270 PRINT"| OO \ | CHARACTER"
17280 PRINT"| Q #Q | "
17290 PRINT" [ ] "

```



```

17300 PRINT"
17305 R=R+1:FORP=1 TO 2000:NEXT P
17310 RETURN
17320 PRINT"
17330 PRINT"  "
17340 PRINT"| K  K |"
17350 PRINT"| **** | BEWARE OF ILL WILL"
17360 PRINT"|  \ | FROM THOSE AROUND"
17370 PRINT"|  \ | YOU AND YOURS"
17380 PRINT"| K  K |"
17390 PRINT"  "
17400 PRINT"
17410 R=R+1:FORP=1 TO 2000:NEXT P
17420 RETURN
20000 REM END
20010 PRINT:PRINT"LOTS OF LUCK!"
20020 END

```

The following lines show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustrations in *Magic Cards*.

```

10000 REM CARDS SUBROUTINES
10010 PRINTCHR$(147)
10020 PRINT:PRINT
10030 PRINT"{#207}{#183 5}{#208} "
10040 PRINT"{#180}2 {#193} 2{#170} "
10050 PRINT"{#180}          {#170} "
10060 PRINT"{#180}          {#170} YOU WILL B
E PROTECTED"
10070 PRINT"{#180}          {#170} OR RECEIVE
GIFTS"
10080 PRINT"{#180}2 {#193} 2{#170} "
10090 PRINT"{#204}{#175 5}{#186} "
10100 FOR P=1 TO 2000:NEXT P
10110 R=R+1:RETURN
10120 REM
10130 PRINT"{#207}{#183 5}{#208} "
10140 PRINT"{#180}3 {#193} 3{#170} "

```

```

10150 PRINT"{#180}          {#170} "
10160 PRINT"{#180}  {#193}  {#170} I SEE
A LOVER'S"
10170 PRINT"{#180}          {#170} FLIGHT"
10180 PRINT"{#180}3 {#193} 3{#170} "
10190 PRINT"{#204}{#175 5}{#186} "
10200 R=R+1:FOR P=1 TO 2000:NEXT P
10210 RETURN
10230 PRINT"{#207}{#183 5}{#208} "
10240 PRINT"{#180}4{#193}  {#193}4{#170}
"
10250 PRINT"{#180}          {#170} "
10260 PRINT"{#180}          {#170} YOU WILL R
ECEIVE"
10270 PRINT"{#180}          {#170} BAD NEWS"
10280 PRINT"{#180}4{#193}  {#193}4{#170}
"
10290 PRINT"{#204}{#175 5}{#186} "
10300 R=R+1:FOR P=1 TO 2000:NEXT P
10310 RETURN
10330 PRINT"{#207}{#183 5}{#208} "
10340 PRINT"{#180}5{#193}  {#193}5{#170}
"
10350 PRINT"{#180}          {#170} "
10360 PRINT"{#180}  {#193}  {#170} YOU W
ILL MEET WITH"
10370 PRINT"{#180}          {#170} MISFORTUNE
"
10380 PRINT"{#180}5{#193}  {#193}5{#170}
"
10390 PRINT"{#204}{#175 5}{#186} "
10400 R=R+1:FOR P=1 TO 2000:NEXT P
10410 RETURN
10430 PRINT"{#207}{#183 5}{#208} "
10440 PRINT"{#180}6{#193}  {#193}6{#170}
"
10450 PRINT"{#180}          {#170} "
10460 PRINT"{#180}  {#193}  {#193}  {#170}
A GOOD VOYAGE WILL"
10470 PRINT"{#180}          {#170} BE MADE SO
ON"

```

```

10480 PRINT"#{180}6#{193} {#193}6#{170}
"
10490 PRINT"#{204}{#175 5}{#186} "
10500 R=R+1:FORP=1 TO 2000:NEXT P
10510 RETURN
10530 PRINT"#{207}{#183 5}{#208} "
10540 PRINT"#{180}7#{193} {#193}7#{170}
"
10550 PRINT"#{180}      {#170} "
10560 PRINT"#{180} {#193} {#193} {#170}
GOOD FORTUNE "
10570 PRINT"#{180} {#193} {#170} "
10580 PRINT"#{180}7#{193} {#193}7#{170}
"
10590 PRINT"#{204}{#175 5}{#186} "
10605 R=R+1:FORP=1 TO 2000:NEXT P
10610 RETURN
10630 PRINT"#{207}{#183 5}{#208} "
10640 PRINT"#{180}8#{193} {#193}8#{170}
"
10650 PRINT"#{180} {#193} {#170} "
10660 PRINT"#{180} {#193} {#193} {#170}
THERE WILL BE A"
10670 PRINT"#{180} {#193} {#170} SCAND
AL FOR A WOMAN"
10680 PRINT"#{180}8#{193} {#193}8#{170}
"
10690 PRINT"#{204}{#175 5}{#186} "
10705 R=R+1:FORP=1 TO 2000:NEXT P
10710 RETURN
10730 PRINT"#{207}{#183 5}{#208} "
10740 PRINT"#{180}9#{193} {#193}9#{170}
"
10750 PRINT"#{180} {#193} {#193} {#170}
"
10760 PRINT"#{180} {#193} {#170} THIS
CARD IS A"
10770 PRINT"#{180} {#193} {#193} {#170}
BAD OMEN"
10780 PRINT"#{180}9#{193} {#193}9#{170}
"

```

```

10790 PRINT"{#204}{#175 5}{#186} "
10805 R=R+1:FORP=1 TO 2000:NEXT P
10810 RETURN
10830 PRINT"{#207}{#183 5}{#208} "
10840 PRINT"10{#193} {#193}10"
10850 PRINT"{#180} {#193} {#170} A FRI
END WILL BE "
10860 PRINT"{#180} {#193} {#193} {#170}
A TRAITOR "
10870 PRINT"{#180} {#193} {#193} {#170}"

10880 PRINT"10{#193 3}10"
10890 PRINT"{#204}{#175 5}{#186}"
10905 R=R+1:FORP=1 TO 2000:NEXT P
10910 RETURN
10930 PRINT"{#207}{#183 5}{#208} "
10940 PRINT"{#180}A A{#170}"
10950 PRINT"{#180} {#170} EITHER GRE
AT "
10960 PRINT"{#180} {#193} {#170} RICHE
S OR GREAT "
10970 PRINT"{#180} {#170} MISERY "
10980 PRINT"{#180}A A{#170}"
10990 PRINT"{#204}{#175 5}{#186}"
11005 R=R+1:FORP=1 TO 2000:NEXT P
11010 RETURN
11030 PRINT"{#207}{#183 5}{#208} "
11040 PRINT"{#180}J{#193} J{#170}"
11050 PRINT"{#180} *** {#170}"
11060 PRINT"{#180} ..{#194} {#170} SOMED
NE WILL MAKE AN"
11070 PRINT"{#180}{#213}{#192 2}{#201} {
#170} INDISCREET INQUIRY"
11080 PRINT"{#180}J {#193}J{#170}"
11090 PRINT"{#204}{#175 5}{#186}"
11105 R=R+1:FORP=1 TO 2000:NEXT P
11110 RETURN
11130 PRINT"{#207}{#183 5}{#208} "
11140 PRINT"{#180}Q{#193} Q{#170}"
11150 PRINT"{#180}#### {#170} I SEE A W
IDOW"
11160 PRINT"{#180} ..{#205} {#170}"

```

```

11170 PRINT"{#180} {#209 2}{#202} {#170}
"
11180 PRINT"{#180}Q {#193}Q{#170}"
11190 PRINT"{#204}{#175 5}{#186}"
11205 R=R+1:FORP=1 TO 2000:NEXT P
11210 RETURN
11230 PRINT"{#207}{#183 5}{#208} "
11240 PRINT"{#180}K{#193} K{#170}"
11250 PRINT"{#180}*****{#170}"
11260 PRINT"{#180} {#190 2}{#205} {#170}
I SEE A PROFESS-"
11270 PRINT"{#180}{#202}{#192 2}{#203}{#
202}{#170} IDONAL MAN"
11280 PRINT"{#180}K {#193}K{#170}"
11290 PRINT"{#204}{#175 5}{#186}"
12005 R=R+1:FORP=1 TO 2000:NEXT P
12010 RETURN
12030 PRINT"{#207}{#183 5}{#208} "
12040 PRINT"{#180}A A{#170}"
12050 PRINT"{#180} {#170}"
12060 PRINT"{#180} {#216} {#170} CALA
MITIES ABOUND"
12070 PRINT"{#180} {#170}"
12080 PRINT"{#180}A A{#170}"
12090 PRINT"{#204}{#175 5}{#186}"
12105 R=R+1:FORP=1 TO 2000:NEXT P
12110 RETURN
12130 PRINT"{#207}{#183 5}{#208} "
12140 PRINT"{#180}2 {#216} 2{#170} "
12150 PRINT"{#180} {#170} MANY SMAL
L "
12160 PRINT"{#180} {#170} DISAPPOIN
TMENTS "
12170 PRINT"{#180} {#170} "
12180 PRINT"{#180}2 {#216} 2{#170} "
12190 PRINT"{#204}{#175 5}{#186}"
12205 R=R+1:FORP=1 TO 2000:NEXT P
12210 RETURN
12230 PRINT"{#207}{#183 5}{#208} "
12240 PRINT"{#180}3 {#216} 3{#170} "
12250 PRINT"{#180} {#170} YOU MAY W

```

```

ORK WITH"
12260 PRINT"{#180}  {#216}  {#170}  ANOT
HER. BEWARE!"
12270 PRINT"{#180}      {#170}  "
12280 PRINT"{#180}3 {#216} 3{#170}  "
12290 PRINT"{#204}{#175 5}{#186}  "
12305 R=R+1:FORP=1 TO 2000:NEXT P
12310 RETURN
12330 PRINT"{#207}{#183 5}{#208}  "
12340 PRINT"{#180}4{#216}  {#216}4{#170}
"
12350 PRINT"{#180}      {#170}  UNEXPECTE
D GOOD"
12360 PRINT"{#180}      {#170}  FORTUNE S
OON  "
12370 PRINT"{#180}      {#170}  "
12380 PRINT"{#180}4{#216}  {#216}4{#170}
"
12390 PRINT"{#204}{#175 5}{#186}  "
12405 R=R+1:FORP=1 TO 2000:NEXT P
12410 RETURN
12430 PRINT"{#207}{#183 5}{#208}  "
12440 PRINT"{#180}5{#216}  {#216}5{#170}
"
12450 PRINT"{#180}      {#170}  YOU MAY E
XPECT BRIEF"
12460 PRINT"{#180}  {#216}  {#170}  FINAN
CIAL SUCCESS"
12470 PRINT"{#180}      {#170}  "
12480 PRINT"{#180}5{#216}  {#216}5{#170}
"
12490 PRINT"{#204}{#175 5}{#186}  "
12605 R=R+1:FORP=1 TO 2000:NEXT P
12610 RETURN
12630 PRINT"{#207}{#183 5}{#208}  "
12640 PRINT"{#180}6{#216}  {#216}6{#170}
"
12650 PRINT"{#180}      {#170}  YOU WILL
BE BETRAYED"
12660 PRINT"{#180}  {#216}  {#216}  {#170}
  BY A FRIEND"

```

```

12670 PRINT"{#180}      {#170} "
12680 PRINT"{#180}6{#216} {#216}6{#170}
"
12690 PRINT"{#204}{#175 5}{#186} "
12705 R=R+1:FORP=1 TO 2000:NEXT P
12710 RETURN
12730 PRINT"{#207}{#183 5}{#208} "
12740 PRINT"{#180}7{#216} {#216}7{#170}
"
12750 PRINT"{#180}      {#170} I SEE A DA
RK CHILD"
12760 PRINT"{#180} {#216} {#216} {#170}
IN YOUR LIFE"
12770 PRINT"{#180} {#216} {#170} "
12780 PRINT"{#180}7{#216} {#216}7{#170}
"
12790 PRINT"{#204}{#175 5}{#186} "
12805 R=R+1:FORP=1 TO 2000:NEXT P
12810 RETURN
12830 PRINT"{#207}{#183 5}{#208} "
12840 PRINT"{#180}8{#216} {#216}8{#170}
"
12850 PRINT"{#180} {#216} {#170} THERE
WILL BE"
12860 PRINT"{#180} {#216} {#216} {#170}
DOMESTIC DISPUTES"
12870 PRINT"{#180} {#216} {#170} "
12880 PRINT"{#180}8{#216} {#216}8{#170}
"
12890 PRINT"{#204}{#175 5}{#186} "
12905 R=R+1:FORP=1 TO 2000:NEXT P
12910 RETURN
12930 PRINT"{#207}{#183 5}{#208} "
12940 PRINT"{#180}9{#216} {#216}9{#170}
"
12950 PRINT"{#180} {#216} {#216} {#170}
BAD NEWS WILL COME"
12960 PRINT"{#180} {#216} {#170} FOR Y
OU SOON"
12970 PRINT"{#180} {#216} {#216} {#170}
"

```

```

12980 PRINT"{#180}9{#216} {#216}9{#170}
"
12990 PRINT"{#204}{#175 5}{#186} "
13005 R=R+1:FORP=1 TO 2000:NEXT P
13010 RETURN
13030 PRINT"{#207}{#183 5}{#208} "
13040 PRINT"10{#216} {#216}10"
13050 PRINT"{#180} {#216} {#170} DIFFI
CULTIES "
13060 PRINT"{#180} {#216} {#216} {#170}
AHEAD"
13070 PRINT"{#180} {#216} {#216} {#170}"

13080 PRINT"10{#216 3}10"
13090 PRINT"{#204}{#175 5}{#186}"
13105 R=R+1:FORP=1 TO 2000:NEXT P
13110 RETURN
13130 PRINT"{#207}{#183 5}{#208} "
13140 PRINT"{#180}J{#216} J{#170}"
13150 PRINT"{#180} *** {#170} YOU WILL S
EARCH"
13160 PRINT"{#180} ..{#194} {#170} FOR A
MATE"
13170 PRINT"{#180} {#175 2}{#202} {#170}
"
13180 PRINT"{#180}J {#216}J{#170}"
13190 PRINT"{#204}{#175 5}{#186}"
13205 R=R+1:FORP=1 TO 2000:NEXT P
13210 RETURN
13230 PRINT"{#207}{#183 5}{#208} "
13240 PRINT"{#180}Q{#216} Q{#170}"
13250 PRINT"{#180} *{#205} {#170} YOU W
ILL HAVE A"
13260 PRINT"{#180} ..{#205} {#170} BOUNT
IFUL HARVEST"
13270 PRINT"{#180} {#209 2}{#202} {#170}
"
13280 PRINT"{#180}Q {#216}Q{#170}"
13290 PRINT"{#204}{#175 5}{#186}"
13305 R=R+1:FORP=1 TO 2000:NEXT P
13310 RETURN

```



```

13330 PRINT"{#207}{#183 5}{#208} "
13340 PRINT"{#180}K{#216} K{#170}"
13350 PRINT"{#180} ****{#170} FAVORABLE
TIDINGS"
13360 PRINT"{#180} ..{#205} {#170}"
13370 PRINT"{#180} --{#202} {#170}"
13380 PRINT"{#180}K {#216}K{#170}"
13390 PRINT"{#204}{#175 5}{#186}"
14005 R=R+1:FORP=1 TO 2000:NEXT P
14010 RETURN
14020 PRINT"{RED}"
14030 PRINT"{#207}{#183 5}{#208} "
14040 PRINT"{#180}A A{#170}"
14050 PRINT"{#180} {#170} THIS CARD
WILL"
14060 PRINT"{#180} {#218} {#170} BRING
YOU ALL YOU"
14070 PRINT"{#180} {#170} WISH FOR"
14080 PRINT"{#180}A A{#170}"
14090 PRINT"{#204}{#175 5}{#186}"
14100 PRINT"{BLK}"
14105 R=R+1:FORP=1 TO 2000:NEXT P
14110 RETURN
14120 PRINT"{RED}"
14130 PRINT"{#207}{#183 5}{#208} "
14140 PRINT"{#180}2 {#218} 2{#170} "
14150 PRINT"{#180} {#170} YOU IMAGIN
E MANY"
14160 PRINT"{#180} {#170} OF YOUR TR
OUBLES"
14170 PRINT"{#180} {#170} "
14180 PRINT"{#180}2 {#218} 2{#170} "
14190 PRINT"{#204}{#175 5}{#186} "
14200 PRINT"{BLK}"
14205 R=R+1:FORP=1 TO 2000:NEXT P
14210 RETURN
14220 PRINT"{RED}"
14230 PRINT"{#207}{#183 5}{#208} "
14240 PRINT"{#180}3 {#218} 3{#170} "
14250 PRINT"{#180} {#170} THE COLLAB
ORATION"

```

```

14260 PRINT" {#180} {#218} {#170} WILL
BE USEFUL"
14270 PRINT" {#180} {#170} "
14280 PRINT" {#180}3 {#218} 3{#170} "
14290 PRINT" {#204} {#175 5} {#186} "
14300 PRINT" {BLK} "
14305 R=R+1:FORP=1 TO 2000:NEXT P
14310 RETURN
14320 PRINT" {RED} "
14330 PRINT" {#207} {#183 5} {#208} "
14340 PRINT" {#180}4 {#218} {#218}4 {#170}
"
14350 PRINT" {#180} {#170} PLEASANT N
EWS FROM"
14360 PRINT" {#180} {#170} A LADY"
14370 PRINT" {#180} {#170} "
14380 PRINT" {#180}4 {#218} {#218}4 {#170}
"
14390 PRINT" {#204} {#175 5} {#186} "
14400 PRINT" {BLK} "
14405 R=R+1:FORP=1 TO 2000:NEXT P
14410 RETURN
14420 PRINT" {RED} "
14430 PRINT" {#207} {#183 5} {#208} "
14440 PRINT" {#180}5 {#218} {#218}5 {#170}
"
14450 PRINT" {#180} {#170} REASON CON
QUERS"
14460 PRINT" {#180} {#218} {#170} FORTU
NE"
14470 PRINT" {#180} {#170} "
14480 PRINT" {#180}5 {#218} {#218}5 {#170}
"
14490 PRINT" {#204} {#175 5} {#186} "
14500 PRINT" {BLK} "
14605 R=R+1:FORP=1 TO 2000:NEXT P
14610 RETURN
14620 PRINT" {RED} "
14630 PRINT" {#207} {#183 5} {#208} "
14640 PRINT" {#180}6 {#218} {#218}6 {#170}
"

```

```

14650 PRINT"{#180}          {#170} BE CAUTIOUS AT"
14660 PRINT"{#180} {#218} {#218} {#170} THE PRESENT"
14670 PRINT"{#180}          {#170} "
14680 PRINT"{#180}6{#218} {#218}6{#170} "
14690 PRINT"{#204}{#175 5}{#186} "
14700 PRINT"{BLK}"
14705 R=R+1:FORP=1 TO 2000:NEXT P
14710 RETURN
14720 PRINT"{RED}"
14730 PRINT"{#207}{#183 5}{#208} "
14740 PRINT"{#180}7{#218} {#218}7{#170} "
14750 PRINT"{#180}          {#170} YOUR POSITION WILL"
14760 PRINT"{#180} {#218} {#218} {#170} IMPROVE NOW"
14770 PRINT"{#180} {#218} {#170} "
14780 PRINT"{#180}7{#218} {#218}7{#170} "
14790 PRINT"{#204}{#175 5}{#186} "
14800 PRINT"{BLK}"
14805 R=R+1:FORP=1 TO 2000:NEXT P
14810 RETURN
14820 PRINT"{RED}"
14830 PRINT"{#207}{#183 5}{#208} "
14840 PRINT"{#180}8{#218} {#218}8{#170} "
14850 PRINT"{#180} {#218} {#170} I SEE A YOUNG MAN"
14860 PRINT"{#180} {#218} {#218} {#170} AND A DARK WOMAN"
14870 PRINT"{#180} {#218} {#170} "
14880 PRINT"{#180}8{#218} {#218}8{#170} "
14890 PRINT"{#204}{#175 5}{#186} "
14900 PRINT"{BLK}"
14905 R=R+1:FORP=1 TO 2000:NEXT P
14910 RETURN

```

```

14920 PRINT"{RED}"
14930 PRINT"{#207}{#183 5}{#208} "
14940 PRINT"{#180}9{#218} {#218}9{#170}
"
14950 PRINT"{#180} {#218} {#218} {#170}
THIS REINFORCES"
14960 PRINT"{#180} {#218} {#170} THE O
THER CARDS"
14970 PRINT"{#180} {#218} {#218} {#170}
IN THIS HAND"
14980 PRINT"{#180}9{#218} {#218}9{#170}
"
14990 PRINT"{#204}{#175 5}{#186} "
15000 PRINT"{BLK}"
15005 R=R+1:FORP=1 TO 2000:NEXT P
15010 RETURN
15020 PRINT"{RED}"
15030 PRINT"{#207}{#183 5}{#208} "
15040 PRINT"10{#218} {#218}10"
15050 PRINT"{#180} {#218} {#170} I SEE
A HOUSE "
15060 PRINT"{#180} {#218} {#218} {#170}"
15070 PRINT"{#180} {#218} {#218} {#170}"
15080 PRINT"10{#218 3}10"
15090 PRINT"{#204}{#175 5}{#186}"
15100 PRINT"{BLK}"
15105 R=R+1:FORP=1 TO 2000:NEXT P
15110 RETURN
15120 PRINT"{RED}"
15130 PRINT"{#207}{#183 5}{#208} "
15140 PRINT"{#180}J{#218} J{#170}"
15150 PRINT"{#180} *** {#170} I SEE A YO
UNG"
15160 PRINT"{#180} ..{#194} {#170} PERSO
N"
15170 PRINT"{#180} {#175 2}{#202} {#170}
"
15180 PRINT"{#180}J {#218}J{#170}"
15190 PRINT"{#204}{#175 5}{#186}"
15200 PRINT"{BLK}"
15205 R=R+1:FORP=1 TO 2000:NEXT P

```

```

15210 RETURN
15220 PRINT "{RED}"
15230 PRINT "{#207}{#183 5}{#208} "
15240 PRINT "{#180}Q{#218} Q{#170}"
15250 PRINT "{#180} {#166 2}{#223} {#170}
GREAT RICHES"
15260 PRINT "{#180} ..{#205} {#170}"
15270 PRINT "{#180} {#209 2}{#202} {#170}
"
15280 PRINT "{#180}Q {#218}Q{#170}"
15290 PRINT "{#204}{#175 5}{#186}"
15300 PRINT "{BLK}"
15305 R=R+1:FORP=1 TO 2000:NEXT P
15310 RETURN
15320 PRINT "{RED}"
15330 PRINT "{#207}{#183 5}{#208} "
15340 PRINT "{#180}K{#218} K{#170}"
15350 PRINT "{#180}**** {#170} I SEE A DA
RK MAN"
15360 PRINT "{#180}{#206}{#215 2}{#205} {
#170}"
15370 PRINT "{#180}{#203}<>{#202} {#170}"
15380 PRINT "{#180}K {#218}K{#170}"
15390 PRINT "{#204}{#175 5}{#186}"
15400 PRINT "{BLK}"
16005 R=R+1:FORP=1 TO 2000:NEXT P
16010 RETURN
16020 PRINT "{RED}"
16030 PRINT "{#207}{#183 5}{#208} "
16040 PRINT "{#180}A A{#170}"
16050 PRINT "{#180} {#170} A STRONG W
ILL;"
16060 PRINT "{#180} {#211} {#170} UNCHA
NGEABLE LAW"
16070 PRINT "{#180} {#170}"
16080 PRINT "{#180}A A{#170}"
16090 PRINT "{#204}{#175 5}{#186}"
16100 PRINT "{BLK}"
16105 R=R+1:FORP=1 TO 2000:NEXT P
16110 RETURN
16120 PRINT "{RED}"

```

```

16130 PRINT" {#207} {#183 5} {#208} "
16140 PRINT" {#180}2 {#211} 2{#170} "
16150 PRINT" {#180}          {#170} FAVORABLE
BUSINESS"
16160 PRINT" {#180}          {#170} AND PLEASU
RE"
16170 PRINT" {#180}          {#170} "
16180 PRINT" {#180}2 {#211} 2{#170} "
16190 PRINT" {#204} {#175 5} {#186} "
16200 PRINT" {BLK} "
16205 R=R+1:FORP=1 TO 2000:NEXT P
16210 RETURN
16220 PRINT" {RED} "
16230 PRINT" {#207} {#183 5} {#208} "
16240 PRINT" {#180}3 {#211} 3{#170} "
16250 PRINT" {#180}          {#170} UNEXPECTED
LY YOU"
16260 PRINT" {#180}  {#211}  {#170} WILL
ADVANCE"
16270 PRINT" {#180}          {#170} "
16280 PRINT" {#180}3 {#211} 3{#170} "
16290 PRINT" {#204} {#175 5} {#186} "
16300 PRINT" {BLK} "
16305 R=R+1:FORP=1 TO 2000:NEXT P
16310 RETURN
16320 PRINT" {RED} "
16330 PRINT" {#207} {#183 5} {#208} "
16340 PRINT" {#180}4{#211}  {#211}4{#170}
"
16350 PRINT" {#180}          {#170} EVENTS CON
TRARY TO"
16360 PRINT" {#180}          {#170} YOUR DESIR
ES"
16370 PRINT" {#180}          {#170} "
16380 PRINT" {#180}4{#211}  {#211}4{#170}
"
16390 PRINT" {#204} {#175 5} {#186} "
16400 PRINT" {BLK} "
16405 R=R+1:FORP=1 TO 2000:NEXT P
16410 RETURN
16420 PRINT" {RED} "

```

```

16430 PRINT"{#207}{#183 5}{#208} "
16440 PRINT"{#180}5{#211} {#211}5{#170}
"
16450 PRINT"{#180}      {#170} HAPPINESS,
SUCCESS"
16460 PRINT"{#180} {#211} {#170} & GIF
TS ARE YOURS"
16470 PRINT"{#180}      {#170} "
16480 PRINT"{#180}5{#211} {#211}5{#170}
"
16490 PRINT"{#204}{#175 5}{#186} "
16500 PRINT"{BLK}"
16505 R=R+1:FDRP=1 TO 2000:NEXT P
16510 RETURN
16520 PRINT"{RED}"
16620 PRINT"{RED}"
16630 PRINT"{#207}{#183 5}{#208} "
16640 PRINT"{#180}6{#211} {#211}6{#170}
"
16650 PRINT"{#180}      {#170} PLEASANT M
EMORIES"
16660 PRINT"{#180} {#211} {#211} {#170}
ARE FORETOLD"
16670 PRINT"{#180}      {#170} "
16680 PRINT"{#180}6{#211} {#211}6{#170}
"
16690 PRINT"{#204}{#175 5}{#186} "
16700 PRINT"{BLK}"
16705 R=R+1:FDRP=1 TO 2000:NEXT P
16710 RETURN
16720 PRINT"{RED}"
16730 PRINT"{#207}{#183 5}{#208} "
16740 PRINT"{#180}7{#211} {#211}7{#170}
"
16750 PRINT"{#180}      {#170} I SEE A FA
IR CHILD"
16760 PRINT"{#180} {#211} {#211} {#170}
& A RESOLUTION."
16770 PRINT"{#180} {#211} {#170} "
16780 PRINT"{#180}7{#211} {#211}7{#170}
"

```

```

16790 PRINT"{#204}{#175 5}{#186} "
16800 PRINT"{BLK}"
16805 R=R+1:FORP=1 TO 2000:NEXT P
16810 RETURN
16820 PRINT"{RED}"
16830 PRINT"{#207}{#183 5}{#208} "
16840 PRINT"{#180}8{#211} {#211}8{#170}
"
16850 PRINT"{#180} {#211} {#170} I SEE
MARRIAGE "
16860 PRINT"{#180} {#211} {#211} {#170}
FOR A FAIR WOMAN"
16870 PRINT"{#180} {#211} {#170} "
16880 PRINT"{#180}8{#211} {#211}8{#170}
"
16890 PRINT"{#204}{#175 5}{#186} "
16900 PRINT"{BLK}"
16905 R=R+1:FORP=1 TO 2000:NEXT P
16910 RETURN
16920 PRINT"{RED}"
16930 PRINT"{#207}{#183 5}{#208} "
16940 PRINT"{#180}9{#211} {#211}9{#170}
"
16950 PRINT"{#180} {#211} {#211} {#170}
A PROPITIOUS "
16960 PRINT"{#180} {#211} {#170} FATE
FOR MILITARY"
16970 PRINT"{#180} {#211} {#211} {#170}
PERSONS"
16980 PRINT,"{#180}9{#211} {#211}9{#170}
"
16990 PRINT"{#204}{#175 5}{#186} "
17000 PRINT"{BLK}"
17005 R=R+1:FORP=1 TO 2000:NEXT P
17010 RETURN
17020 PRINT"{RED}"
17030 PRINT"{#207}{#183 5}{#208} "
17040 PRINT"10{#211} {#211}10"
17050 PRINT"{#180} {#211} {#170} I SEE
A GOOD"
17060 PRINT"{#180} {#211} {#211} {#170}

```


& LONG MARRIAGE"

17070 PRINT"{#180} {#211} {#211} {#170}"

17080 PRINT"10{#211 3}10"

17090 PRINT"{#204}{#175 5}{#186}"

17100 PRINT"{BLK}"

17105 R=R+1:FORP=1 TO 2000:NEXT P

17110 RETURN

17120 PRINT"{RED}"

17130 PRINT"{#207}{#183 5}{#208} "

17140 PRINT"{#180}J{#211} J{#170}"

17150 PRINT"{#180} *** {#170} BENEFICEN
CE FOR"

17160 PRINT"{#180} ..{#194} {#170} YOU A
ND YOURS"

17170 PRINT"{#180} {#175 2}{#202} {#170}
"

17180 PRINT"{#180}J {#211}J{#170}"

17190 PRINT"{#204}{#175 5}{#186}"

17200 PRINT"{BLK}"

17205 R=R+1:FORP=1 TO 2000:NEXT P

17210 RETURN

17220 PRINT"{RED}"

17230 PRINT"{#207}{#183 5}{#208} "

17240 PRINT"{#180}Q{#211} Q{#170}"

17250 PRINT"{#180}{#214 4} {#170} I SEE
A WOMAN OF"

17260 PRINT"{#180}{#206}..{#205} {#170}
UNCERTAIN CHARAC-

17270 PRINT"{#180} {#215 2}{#202} {#170}
TER"

17280 PRINT"{#180}Q {#211}Q{#170}"

17290 PRINT"{#204}{#175 5}{#186}"

17300 PRINT"{BLK}"

17305 R=R+1:FORP=1 TO 2000:NEXT P

17310 RETURN

17320 PRINT"{RED}"

17330 PRINT"{#207}{#183 5}{#208} "

17340 PRINT"{#180}K{#211} K{#170}"

17350 PRINT"{#180}**** {#170} BEWARE OF
ILL WILL"

17360 PRINT"{#180} {#209 2}{#205} {#170}

```
FROM THOSE AROUND"  
17370 PRINT"{#180}{#202}{#162 2}{#203}{#  
202}{#170} YOU AND YOURS"  
17380 PRINT"{#180}K {#211}K{#170}"  
17390 PRINT"{#204}{#175 5}{#186}"  
17400 PRINT"{BLK}"  
17410 R=R+1:FOR P=1 TO 2000:NEXT P  
17420 RETURN  
20000 REM END  
20010 PRINT:PRINT"LOTS OF LUCK!"  
20020 END
```

Part 4

ADVANCED GAME TOOLS AND TECHNIQUES

Chapter 16

ENHANCING COMMODORE 64 GAMES

To create computer games, you must know two things: First, you need to know the general techniques of programming in BASIC (or some other program language); second, you must know the capabilities of the computer you own. In this part of the book we will explore both areas, the heavier emphasis falling on the Commodore's strong graphics and sound functions.

Of great importance to serious programmers is the way in which the Commodore designers organized the 64's internal memory. For many applications it is necessary to have a fairly good understanding of this, otherwise some applications would be difficult or impossible to achieve. Our sample programs should help you get started.

Animation is one very widely used technique which we examine in some detail. Sprites are an outstanding feature of the C-64. Sprites are so important and work so well with animation that we have dedicated two chapters to the subject. You will find countless uses for sprites in your games.

Alternate input devices—joysticks and game paddles, specifically—are of great benefit in fast-action games. We touch on both keyboard and joystick control of moving objects.

Perhaps the most powerful and least well-developed feature of the Commodore 64 is its marvelous sound capabilities. We have included examples of both music and sound effects which should give you some good ideas for your games.

Finally, we have provided sample programs which show you how to create data files that store both sprite and screen data. These can be very handy when you want to use the same body of data in several programs.

While we emphasize the Commodore 64's internal workings, we will not neglect the sort of general information that may be useful to you. Remember, in the final analysis hardware and programming technique must work together.

Chapter 17

MANAGING MEMORY

The last thing most new computer owners want to get involved in is talk about memory maps, pointers, microchips, etc. This is understandable. However, there are instances when a little knowledge about these topics can go a long way toward solving big problems. This frequently occurs in the area of game programs, especially when graphics are involved.

In order to write simple, short games (or any such program), all you need is a computer and a rudimentary knowledge of BASIC. As soon as you start writing larger programs which use lots of graphics and sound, however, you'll need to understand how the computer's memory is organized and controlled.

This chapter is intended to give you a fundamental understanding of the Commodore 64's memory organization and to help you modify the Commodore's normal memory organization whenever it fails to accommodate the needs of your programs. We will keep the complexity to a minimum. Our goal will be to provide you with some skeleton (setup) programs which can be inserted in your games.

Here goes!

Commodore 64 Memory Organization

All data is stored in the computer's memory in the form of bytes. Each byte is stored in a unique location called a memory address. In the Commodore 64, there are 65536 such addresses assigned to RAM. The C-64's microprocessor is the 6510. It can access (look at) any one of those 65536 locations at any given moment.

The 65536 addresses which make up the Commodore's RAM are arranged in a very specific way. Some parts of RAM are used by the computer to keep track of program instructions, where variables are stored, and so forth; other portions are dedicated to the task of defining what is displayed on the video monitor; and still other parts are free to be used by your program and the variables it creates.

Here is a very simplified graphic description (map) of the 64 kilobytes (64K) of RAM in the Commodore. The boundaries are expressed in decimal form:

57344-65535	(CBM kernal-operating system)
56320-57343	(Input/Output registers)
55296-56296	(Color RAM)
53248-53294	(VIC-II registers)
49152-53237	(special RAM area)
40960-49151	(BASIC language interpreter)
2048-40959	(free RAM area)
2040-2047	(sprite pointers)
1024-2039	(screen memory)
0-1023	(operating system memory)

Let's take a closer look at the areas of this map which we will need to manipulate as we write longer, more complicated games. Some areas of memory contain what are called registers. Registers are locations which act essentially as switches. They hold data which indicate the status of various graphics and sound capabilities. Other memory locations are called pointers. The data contained in pointer locations tells the Commodore 64 where to go to get different types of information. For instance, one pointer indicates the starting location of the current screen memory. Then there is the large area which holds your program instructions. The memory range 2048-40959 is where BASIC programs are stored. That totals nearly 39,000 bytes.

Reconfiguring Memory

That's a lot of space. Keep in mind, however, that the actual program instructions are not the only things which need room in

memory. As a program is run, extra space is needed to store variables, strings, and so on. Even so, if your program uses no custom graphics characters or sprites, you should be able to avoid reconfiguring the Commodore's memory. However, many games will need custom characters and sprites. In such cases, some of that memory area will have to be reserved for graphics data so the BASIC program will not wipe them out.

The screen and color memory areas will be vital in graphics and sound programming, as well as the VIC-II registers and sprite pointers. The questions that this chapter addresses are: "Which parts of the map must be reconfigured?" and "How is the reconfiguring accomplished?"

For example, say you want to write a game which uses two or three sprites. That, in itself, would not be a problem. On the other hand, what if your program used custom characters as well? If your program is fairly short, say 8 or 9 kilobytes, you can use the memory as is, with one exception: You will need to put some of that RAM out of bounds to your BASIC program as mentioned earlier. This program line would do the trick:

```
100 POKE 52,48:POKE 56,48:CLR
```

This line will cut your available RAM from 38,909 bytes to just 10,237. That's quite a reduction, isn't it? But the lost RAM isn't really lost. It's just reserved for special duty: to store data that defines sprites or custom characters. In addition to lowering the upper limit of the BASIC program area (accomplished by line 100 above), you must also instruct your Commodore 64 where to find the new sprite and character data (e.g., above memory location 12288). We will examine this closely in later chapters.

As long as your program is short enough to fit into 10 kilobytes of RAM, using sprites or custom characters presents no problem. But if your game is 12K or 15K long, as are some of the games in this book, you must use a different approach, an approach which requires some knowledge of how the Commodore 64's micro-processors function.

The VIC-II is the microprocessor (6567) which controls all Commodore 64 graphics. Whereas the 6510 can access 64K (65536 bytes), the VIC-II can access, or see, only 16K at a time. All data which relates to graphics must be within the range accessed by the VIC-II. This includes the screen memory (1024 bytes), character data (2048 bytes for a complete 256 character set), and any sprite data.

So how do you reduce BASIC program memory to reserve memory for sprite data? Actually, the solution is simple. Why not reserve less area for sprite data? We must keep in mind the limitations of the VIC-II chip. By moving the sprite data area higher in memory, the VIC-II can no longer see it. There is an answer (there usually is), and it has to do with the way the VIC-II chip sees different portions of RAM.

The VIC-II accesses 16K of RAM at a time. Divide 64K by 16K and you get 4. There are, in effect, four 16K sections (banks) of memory. The VIC-II can see any of these banks, but only one at a time!

Here are the four banks complete with starting memory locations:

Bank 3 (49152)
Bank 2 (32768)
Bank 1 (16384)
Bank 0 (0)

When you power up your C-64, the VIC-II chip is looking at bank zero. As long as the VIC-II looks at bank zero, any special sprite or custom character data area must be contained within that bank. Unfortunately, this leaves us relatively little area for BASIC programs, since screen memory and operating system memory take their own chunks. And if we want to use custom characters, character-definition data requires its share of space as well.

Moving Screen Memory

One solution is to move the screen memory, as well as sprite and character data, to a different bank. We are going to present a setup

program which utilizes bank two. Bank two starts at location 32768. When this program is run, you will have plenty of RAM (from location 2048 to 32767), over 30 kilobytes, as well as plenty of room to store sprite and custom character data.

But before presenting the setup program, let's review the main functions of the 6510 and VIC-II microprocessors. The 6510 is the chip which performs the essential mathematical calculations in your Commodore 64. The 6510 can access all portions of the computer's memory. Relocating the starting or ending location of program RAM temporarily fools the 6510 into thinking it can see less memory. The VIC-II chip handles all video output functions. The screen memory, character data, and sprite data (if any) must all reside in whichever of the four 16K banks the VIC-II chip is focused on at the moment. The BASIC program can reside in an area of memory not currently accessed by the VIC-II, since the program itself is the province of the 6510 chip, not the VIC-II.

Now the program:

```
10 REM *****
20 REM * A PROGRAM TO MOVE *
30 REM * SCREEN AND CHARACTER *
40 REM * MEMORY ABOVE 32768. *
50 REM * BASIC PROGRAM AREA *
55 REM * IS UNDER 32768. *
60 REM *****
70 :
2000 POKE52,128:POKE56,128:CLR:REM-
      RESERVES MEMORY FOR SPRITE DATA,ETC
2010 POKE56576,(PEEK(56576)AND 252)OR1:
      REM-SWITCHES VIC CHIP TO BANK2(32768)
2020 POKE53272,32:REM -CHANGES SCRNM &
      CHAR MEMORY LOCATIONS-
2030 POKE648,136:REM -POINTS BASIC TO
      NEW SCREEN LOCATION-
2040 POKE56334,PEEK(56334)AND 254:REM-
      KEYBOARD TURNED OFF-
2050 POKE1,PEEK(1) AND 251:REM -SWITCHES
      IN CHARACTER ROM-
```

```
2060 FORI=0TO2047:REM-TRANSFERS DATA FOR
    256 CHARACTERS TO RAM.
2065 POKE32768+I,PEEK(53248+I):NEXT
2070 POKE1,PEEK(1) OR 4:REM-SWITCH
    I/O IN-
2080 POKE56334,PEEK(56334)OR1:REM -TURNS
    KEYBOARD BACK ON-
```

The setup program does several things. In addition to moving screen memory, character memory, and sprite data, the program copies all data which define character set one into RAM and tells the Commodore where to find the new screen memory area.

After you run this program, you can enter and run BASIC programs, do calculations in the immediate mode, and more. Only when you press CTRL-C= will you see a difference (you will see only garbage). This is because you did not transfer the data to define character set two, which is what is normally displayed when those two keys are pressed.

The setup program provides ample memory in which to store sprite data (locations 35840 to 40960). All 256 screen characters are subject to redefinition, and you still have 30,720 bytes for program lines and variables. You can use this setup program in any game. We suggest that you place this routine at the end of your program. Your first program line can call this routine with a GOSUB.

Once the program is working, you can make a version without REMARKS which will take up less memory. In the coming chapter on sprites, this setup program will be especially useful.

Raising the Start of BASIC Programs

Another way to reserve memory area for graphics data is to raise the beginning location of your BASIC program. With this method it is not necessary to relocate screen and character memory, so you may prefer it over the setup program. However, there is one aspect of this method which you may not like: The starting location of BASIC programs must be raised before your program is

loaded into memory. This can be done in the immediate mode by typing a few commands. Then your game can be LOADED and RUN.

But having to type even a few lines before running your program can be inconvenient. A better way to accomplish this is to write a short boot program which modifies memory before it loads the main program. Remember, it must be a separate program because loading the main program will wipe out the first one. The change in the starting location of the BASIC program, however, will still be in effect.

In this method, you end up with a little less available memory (about 24K instead of 30K), but in the long run you may prefer it. And with this method, the BASIC program and all variables are stored at locations 16383 and above. Conversely, screen memory and all character data are located below 16384! How is that possible? Remember, the Commodore 64 has more than one microprocessor. While the 6510 microprocessor is executing your main program (stored in one area of memory), the VIC-II microprocessor is simultaneously handling all the graphic functions, whose data is stored in a different area of memory. (And we've barely mentioned a third microprocessor, the SID chip, which takes care of all sound and music generation!)

Here is the boot program which sets the starting location of memory at 16384:

```
1 REM *****
2 REM * THIS PROGRAM CAN BE USED *
3 REM * TO RAISE THE LOW END OF *
4 REM * THE BASIC PROGRAM AREA OF *
5 REM * MEMORY TO 16384. THE PRO- *
6 REM * GRAM THEN WIPES ITSELF *
7 REM * OUT AND RUNS ANOTHER PRO- *
8 REM * GRAM, ONE WHICH WILL USE *
9 REM * SOME OF THE AREA BELOW *
10 REM * 16384 FOR SPRITE/CHAR *
11 REM * DATA. SCREEN MEMORY *
12 REM * WILL NOT NEED TO BE MOVED *
13 REM *****
```

```
15 PRINT "{CLR}LOAD"CHR$(34)"[NAME OF PRO  
GRAMCHR$(34),8"  
20 POKE198,6  
30 DATA 19,13,82,85,78,13  
40 FORI=1TO6  
50 READ A:POKEI+630,A:NEXT  
70 POKE44,64:POKE16384,0:CLR:NEW
```

(This program can be stored on disk or tape; only the file name needs to be altered to load different games.)

The boot program takes advantage of the Commodore 64's powerful keyboard editing features. First, the words LOAD "PROGRAM NAME", 8 are printed on the screen (line 10). The CHR\$(34) is used to produce quotation marks. In line 20, the number of characters held in the keyboard buffer is set to six. The six keystrokes held in the keyboard buffer are set (line 30-50). Line 70 raises the starting location of the program area of RAM. The program is wiped out by the NEW in line 70. However, the six characters held in the keyboard buffer are still there. Those characters are HOME (which places the cursor at the top line); a carriage return (which copies the PRINTed line); the letters R, U and N; and finally, a trailing carriage return.

This has the effect of LOADING and then RUNNING the desired program, even though the boot program has been erased. This technique, of directly POKING characters into the keyboard buffer which ends with the carriage return, fools the Commodore 64 into believing that someone is actually typing those words.

Summary

There are many other ways to reconfigure the Commodore 64 memory, but since this is an introductory book, we are content to allow interested readers to explore those more advanced techniques on their own.

To summarize, the Commodore 64 game programmer has several memory-organization options:

1. The default configuration is fine for programs which use no graphics or at most two or three sprites.
2. With relatively short programs needing many sprites, moving the end of BASIC down to location 12288 provides ample room to store sprite data which will be protected from the main program.
3. If your program needs custom characters, a modification of option two is possible. In option two, the area protected for sprite data can also hold custom keyboard character data.
4. When your program occupies a lot of memory and uses a good deal of sprite/custom character data, there are two solutions: (a) The starting location of BASIC can be moved upward in memory, thus reserving much of the area below for sprite/character data; or (b) The end of BASIC can be lowered with screen memory as well as sprite/character data placed above that location.

Whichever solution you choose, make sure that you have all the pointers set correctly. For instance, if you move screen memory, don't forget to tell the VIC-II chip about it! Use our example programs as starting points.

Chapter 18

ANIMATION

Of all the graphic features of microcomputers, undoubtedly the most popular is animation. Video-arcade games, which first appeared in the 1970s, have been phenomenally successful because of their color animation effects. The Commodore 64 offers the game designer a variety of animation possibilities.

Animation gives the viewer the sensation of seeing objects moving on the video screen. We are going to cover three categories of animation in this chapter: stationary animation, moving animation, and a combination of both.

Stationary animation may seem to be a contradiction in terms, but it is not. Stationary animation occurs whenever an object changes shape without moving to a new location. Here's an example to try on your Commodore 64:

```
10 PRINT "CLEAR C=M"  
20 FOR I=1 TO 50: NEXT  
30 PRINT "CLEAR SHIFT-N"  
40 FOR I=1 TO 50: NEXT  
50 GOTO 10
```

This program produces a vertical line that swings back and forth. Another example of stationary animation might be a face which alternates between a smile and a frown while remaining in one spot. *Greg's Guitar* is a little program which demonstrates stationary animation using PRINT statements.

```

1 REM *****
2 REM * GREG'S GUITAR *
3 REM * BY GREGORY RUPP *
4 REM *****
10 PRINT "{CLR}"
15 PRINT"
20 PRINT"
30 PRINT"
40 PRINT"
50 PRINT"
60 PRINT"
70 PRINT"
80 PRINT"
90 PRINT"
100 PRINT"
105 PRINT"
110 PRINT"
120 PRINT"
130 PRINT"
140 PRINT"
145 FOR I=1 TO 1000: NEXT
150 PRINT "{CLR}"
1000 PRINT"
2000 PRINT"
3000 PRINT"
4000 PRINT"
5000 PRINT"
6000 PRINT"
7000 PRINT"
8000 PRINT"
9000 PRINT"
10000 PRINT"
11000 PRINT"
12000 PRINT"
13000 PRINT"
13500 PRINT"
14000 FOR I=1 TO 1000: NEXT
15000 GOTO 10

```

The second type of animation involves objects which move from one part of the screen to another. Spaceships, monsters, asteroids,

people, you name it, all these and more have been seen zipping across microcomputer video monitors. This is generally what comes to mind when we think of animation.

The third, and most complicated type of animation involves objects which both change shape and move from one point on the screen to another. A good example of this would be a bird that flies across the screen, wings flapping all the while.

Draw-Erase-Redraw

The process of animating a shape, regardless of which type of animation, is based on a simple principle. That principle is *draw-erase-redraw*. In moving an object around the screen, you are really displaying your object in a succession of locations. If the object, say a bird, is drawn successively in a series of adjacent locations, you will end up with a line of birds across the screen. What makes that process animation is the erasing of the bird image from one location before displaying the image at the next, then erasing the image at spot two before displaying it at spot three, and so on.

When we say draw, we mean display. There are several ways to display objects on the screen. We will detail those shortly. We have also used the term erase. By erase, we actually mean display a blank space. If the background color is blue, you would display a blue space to erase an image at a given screen location. That is the essence of draw-erase-redraw. Display your shape or image at a screen location, erase it by displaying a background-colored space at that same location, then display the image again in an adjacent location.

PRINT Animation

There are three ways to display objects on the Commodore 64's video screen. One way was shown in the animation example given earlier in this chapter. That technique uses `PRINT` statements. You can `PRINT` graphics characters inside quotes, or you can `PRINT` string variables which have been defined previously.

But achieving animation via PRINT statements is probably the least desirable method, because it is difficult to position your object precisely using PRINT. Using the Plot subroutine, as in the game *A Day at the Races* (Chapter 30), lessens this problem greatly. Also, it is difficult to animate larger objects using PRINT. Drawing, erasing, and redrawing objects needing more than one horizontal line is very slow and awkward. You may, however, find the PRINT statement valuable in animating objects of two or three adjacent character spaces on the same line.

The essential features of the PRINT animation method are these:

1. Clear the screen with PRINT "Clear-Screen".
2. Use cursor moves within a PRINT statement to find the desired screen location.
3. Use a PRINT statement to draw your image, preferably in string variable form.
4. PRINT HOME (not CLEAR SCREEN), then more cursor moves to find the original location, where you PRINT a blank space to erase the existing image.
5. Again use PRINT HOME plus cursor moves to locate the next position, where the object is redrawn with PRINT statements.

If you desire in-place animation, then only one routine is needed to find the correct screen position. What will change will be the characters PRINTed. Again, *Greg's Guitar* is an example.

PRINT statements can even be used to achieve the third type of animation, a moving object which changes its shape. *Birdfight* is an inventive example of this technique.

```
1 REM *****
2 REM *   BIRDFIGHT- AN EXAMPLE *
3 REM *   OF KEYBOARD GRAPHICS *
4 REM *     BY GREGORY RUPP     *
5 REM *****
```

```

100 A1$=""
110 A$ ="  "
120 B$ ="  "
130 C$ ="  "
131 D$ ="  "
132 E$ ="  "
133 F$ ="  "
134 F1$=""
140 G1$=""
141 G$ ="  "
142 H$ ="  "
143 I$ ="  "
144 J$ ="  "
150 K$ ="  "
151 L$ ="  "
152 L1$=""
160 IF S=34 THEN S=0
170 PRINT "{CLR}"SPC(S)A1$
180 PRINTSPC(S)A$
190 PRINTSPC(S)B$
200 PRINTSPC(S)C$
210 PRINTSPC(S)D$
220 PRINTSPC(S)E$
230 PRINTSPC(S)F$
240 PRINTSPC(S)F1$
250 FORI=1TO100:NEXT
260 PRINT "{CLR}"
270 PRINTSPC(S)G1$
280 PRINTSPC(S)G$
290 PRINTSPC(S)H$
300 PRINTSPC(S)I$
310 PRINTSPC(S)J$
320 PRINTSPC(S)K$
330 PRINTSPC(S)L$
340 PRINTSPC(S)L1$
350 FORT=1TO150:NEXT
360 IFS=17THENGOSUB740
370 S=S+1
380 IFS=28THEN400
390 GOTO 160
400 PRINT"

```

ENGARDE!!"

```

410 FORI=1TO 450:NEXT
420 A2$="          {#205}{#213}{#209}{#201}
":A3$="    {#213}{#201} {#187} {#205}{#18
7}"
430 A4$=" {#207} {#202}{#203} {#204}{#20
6}{#188}{#205}"
440 A5$="{#188} . {#206}{#221}{#205}{#18
7}{#201}{#206}"
450 A6$="          {#188} {#207}{#221}{#163}"

460 A7$="          {#183 2}"
470 A8$="          {#202}{#209}{#203}"
480 A9$=""
490 B1$="          {#172}{#206}"
500 B2$=" .          {#206}{#190}. "
510 B3$="{#187}{#213}{#201}{#206}{#187}{
#206} {#172}{#205} "
520 B4$="{#204}{#202}{#203}{#205}{#213}{
#185}{#195}{#203}{#210}{#186} {#187}"
530 PRINT"{CLR}"
540 PRINTSPC(S)A2$
550 PRINTSPC(S)A3$
560 PRINTSPC(S)A4$
570 PRINTSPC(S)A5$
580 PRINTSPC(S)A6$
590 PRINTSPC(S)A7$
600 FORI=1TO167:NEXT
610 PRINT"{CLR}"
620 PRINTSPC(S)A8$
630 PRINTSPC(S)A9$
640 PRINTSPC(S)B1$
650 PRINTSPC(S)B2$
660 PRINTSPC(S)B3$
670 PRINTSPC(S)B4$
680 A=28
690 PRINT""
700 PRINT"                                OH
,WELL!?!"
710 FORI=1TO3000:NEXT
720 PRINT"{CLR}"
730 END

```

```
740 PRINT"  
YET!"  
750 FORI=1TO1500:NEXT  
760 RETURN
```

I'LL GET HIM

Birdfight and *Greg's Guitar* were both written by the thirteen-year-old son of one of the authors. These programs use the standard keyboard graphics characters. Custom characters can be used in this fashion as well.

In summary, PRINT-statement animation has severe limitations but, nevertheless, can be used to advantage in given game situations.

POKE Animation

The two more advanced and useful methods for creating animation effects on the Commodore 64 are (1) POKE animation, and (2) sprite animation. Sprite animation will be covered in a later chapter. Here we will detail the use of POKE.

In the last chapter we studied how the C-64's memory is organized. You will recall that an important area of memory is called screen memory. In the screen memory area, or map, every character space on the screen has a separate memory location. The screen is divided into 25 horizontal lines, each with 40 columns, or character spaces. 40 times 25 equals 1000, the number of screen locations. In its default mode, the Commodore 64's screen memory area is located in the range 1024-2023 (decimal).

Instead of PRINTing a character on the screen, it is possible, and often desirable, to POKE that character directly into the appropriate location. One big advantage of POKEing is that you never need to use a complicated routine of cursor moves to find the right spot on the screen. Each screen memory location corresponds to a specific location on the video screen.

There is one complication, however. POKEing a character into screen memory does nothing unless you also POKE a color code

into the corresponding location on the screen color map. With PRINT statements, of course, the character color is set once and need never be changed unless a new color is desired. With POKES, each screen location must have its color defined, even if you cover the whole screen with a single character of a single color (e.g., a red diamond).

In this chapter we will refer to the screen memory map at locations 1024-2023. Remember, however, that we have included a number of games in which the screen map has been moved to a different area of RAM. When working with screen memory that has been moved, remember to POKE characters to locations within the limits of the redefined map. The addresses change, not the concept.

It is important to note that the screen color map does not change its location (55296-56295). Regardless of where the screen memory map resides, color codes will have to be POKED into this area. The color codes are:

0=BLACK	8=ORANGE
1=WHITE	9=BROWN
2=RED	10=Light RED
3=CYAN	11=GREY 1
4=PURPLE	12=GREY 2
5=GREEN	13=Light GREEN
6=BLUE	14=Light BLUE
7=YELLOW	15=GREY 3

The Screen Display Codes for the Commodore 64 are listed in Appendix F. There are 256 codes, numbered 0-255. Codes 128 to 255 are the reverse images of 0 to 127. (A reminder: We are dealing exclusively with character set one, which contains uppercase letters and graphics characters.)

To display a green spade in the lower right-hand corner of the screen, try the following line:

```
100 POKE 2023,65:POKE 56295,5
```


The first half of this program line POKES the code for a spade into screen location 2023, which is the lower right-hand corner. The second half of the line POKES the code for the color green into color map location 56295, which is the equivalent of the previous location. Try the first half of the line by itself and you will see nothing. Add the second POKE and you will get your green spade!

The process of displaying a character on the video screen using POKES is thus twofold: First, a character code is POKED into the desired screen memory location, and second, the desired color code is POKED into the matching color memory map location. This process can be simplified with the following line:

```
100 POKE (Screen Memory Location),  
Screen Code: POKE (Screen Memory  
Location + 54274), Color Code
```

POKEing Character and Color Codes

Whatever the screen memory location, the corresponding color memory location will equal the screen memory location plus 54274. (Unless you have moved the screen memory map. In this case, simply subtract the starting address of the new screen memory from 55296. The remainder will be the new offset to add to the screen memory location when you want to POKE the color code.) We've included a program here that draws a border around the screen by the means of POKES. Notice that FOR/NEXT loops are used. The draw and redraw aspects of animation are included. Since the erase step is omitted, we have rows of dots rather than a single, moving dot.

```
1 REM *****  
2 REM * DRAW A SCREEN BORDER *  
3 REM * USING FOR/NEXT LOOPS *  
4 REM * AND POKES *  
5 REM *****  
100 PRINT "{CLR}"  
110 FOR I=1024 TO 1063
```

```

120 POKEI,81:NEXT
130 FORI=55296 TO 55335
140 POKEI,6:NEXT
150 FOR I=1063 TO 2023 STEP 40
160 POKE I,81
170 NEXT
180 FOR I=55335 TO 56295 STEP 40
190 POKE I,6
200 NEXT
210 FORI=2023 TO 1984 STEP -1
220 POKE I,81:NEXT
230 FORI=56295 TO 56256 STEP-1:POKEI,6:N
EXT
240 FORI=1984 TO 1024 STEP-40
250 POKE I,81:NEXT
260 FORI=56256 TO 55296 STEP-40
270 POKE I,6:NEXT
280 FORI=1TO 500 :NEXT
300 PRINT"{HOME}{DOWN 6}{RIGHT 5}  THAT'
S A NICE BORDER."

```

We recommend using the POKE statement to display characters for animation. PRINT is handier for general purpose screen displays, however. For example, POKEing letters onto the screen for your game instructions would be decidedly more awkward than using PRINT statements.

POKE Animation in Action

We now offer a sample animation program using POKES. In this program, you will be moving a single character, a diamond, around the screen. The direction of movement is controlled with the I, J, K, and M keys. I is for upward movement, J for leftward, M for downward, and K for rightward.

The starting position is arbitrary; it could be in a corner instead of the center of the screen. Since POKES put values into memory locations, we must be careful not to POKE something where it will foul up the program! We have included a routine which POKES

a border of red circles around the screen. By using PEEKs, we ensure that the user cannot move his diamond beyond the border.

The routine to move a character around the screen has the following elements:

1. A variable represents the current screen position of the animated character (in the range 1024-2023).
2. A second variable represents the position to which the character is intended to move, determined by which key (I, J, K, or M) is pressed.
3. The new position is PEEKed to see whether that location is legal (on the screen).
4. If the intended new location is legal, the character is first erased from its current position by POKING a space where it was.
5. The current-position variable (CP) then takes the value of the new-position variable (NP).
6. The character is POKED into the new position (now contained in CP), and the color code is POKED into the matching color map location.
7. Finally, the keyboard is again scanned to determine which, if any, key has been pressed.

In case the intended new position is illegal, the character remains at its current location, and the keyboard is again scanned for a new move. *Monster Maze*, Chapter 26, uses this type of routine.

The terms legal move and illegal move bear some explaining. If you do not want a player to go into a particular area (through a wall, for example), a move there is illegal. If the illegal locations are all of a single character code, checking for an illegal move is simple:

```
IF PEEK (NEWPOSITION) = 32 THEN MOVE OK
```

Character code 32 is a blank space, so any other value will be filtered out by this line. Your game may be more complex. Maybe your character is allowed to go through one kind of wall, in the form of a certain character code, but not through another type of wall, represented by another character code. Your move routine will have to include a line which tests for any and all possible illegal character codes. Let's assume that your game character is allowed to move to empty spaces (code 32) and also to checkerboard spaces (code 102). The following line would filter out other characters:

```
100 IF PEEK (NEWPOSITION) = 32 OR PEEK  
(NEWPOSITION) = 102 THEN MOVE OK
```

If the value of the new position (the intended move position) is some other code, then the character is not allowed to go there. Perhaps the character blows up or meets some other horrible fate that you have devised.

Here is the basic character move routine using POKES and PEEKs. Note that by POKING location 650 with 130, any key is automatically repeated when continually pressed. This is helpful in computer action games.

```
1 REM *****  
2 REM * ROUTINE TO MOVE A CHARACTER *  
3 REM * AROUND THE SCREEN USING *  
4 REM * POKES INSTEAD OF PRINT *  
5 REM *****  
8 POKE53280,15:POKE53281,1:PRINT" {BLK}":  
REM -SET CHAR, SCREEN, BORDER COLORS-  
10 PRINT" {CLR} {DOWN 7} {RIGHT 4} USE TH  
E I,J,K, AND M KEYS"  
20 PRINT" {DOWN} TO MOVE THE HEART AROU  
ND THE SCREEN"  
30 PRINT" {DOWN 3} {RIGHT 6} {RVS} PRESS AN  
Y KEY TO CONTINUE "  
40 GET X$:IFX$=""THEN40
```

```

1000 PRINT" {CLR} "
1010 A=1200
1020 C=54272
1030 POKEA,83:POKEA+C,0
1035 POKE650,128:REM -KEYS REPEAT-
1040 GET A$:IF A$=""THEN1040
1050 IF A$="I"THEN AA=A-40:GOTO1100
1060 IF A$="J"THEN AA=A-1:GOTO1100
1070 IF A$="M"THEN AA=A+40:GOTO1100
1080 IF A$="K"THEN AA=A+1:GOTO1100
1100 IF PEEK(AA)<>32THEN GOTO1040
1120 POKEA,32:A=AA:GOTO1030

```

The following line ranges show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustrations in *Greg's Guitar*.

```

15 PRINT"      {#185 2}
20 PRINT"      {#204}{#186}
30 PRINT"      {#204}{#186}
40 PRINT"      {#204}{#186}
50 PRINT"      {#204}{#186}
60 PRINT"      {#204}{#186}
70 PRINT"      {#204}{#186}
80 PRINT"      {#206}{#180}{#204}{#186}{#170}
} {#205}
90 PRINT"      {#180}{#183}{#204}{#186}{#183}
} {#170}
100 PRINT"      {#205}      {#206}
105 PRINT"      {#170}      {#180}
110 PRINT"      {#206}      {#205}
120 PRINT"      {#180} {#185 2}{#172}{#170}
130 PRINT"      {#180} {#185 2}{#172}{#170}
140 PRINT"      {#205}{#175 4}{#206}

1000 PRINT"      {#206}{#205}
"
2000 PRINT"      {#206}{#205}{#20
6}      "

```

```

3000 PRINT"                {#206}{#205}{#206
}      "
4000 PRINT"                {#206}{#205}{#206}
      "
5000 PRINT"                {#175}  {#206}{#205}{#
206}      "
6000 PRINT"                {#206 2} {#206}{#205}{#
206}      "
7000 PRINT"                {#170} {#205}{#206}{#205
} {#206}      "
8000 PRINT"                {#175}{#206} {#206}{#205}
{#206}
9000 PRINT"                {#206}    {#205}{#206}{#205
}{#206}{#180}      "
10000 PRINT"                {#206}      {#175}{#206}
      "
11000 PRINT"                {#180}  {#205}  {#206}
12000 PRINT"                {#205} {#205} {#172}{#170}
      "
13000 PRINT"                {#205} {#172} {#206}
      "
13500 PRINT"                {#205}{#175}{#206}
      "

```

The following lines show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustrations in *Birdfight*.

```

100 A1$=      "
110 A$=" {#172}{#213}{#201}  {#206}  "
120 B$=" {#167}{#202}{#203}{#172}{#206}
{#202}{#209}{#203}"
130 C$=" {#208}{#205}{#206}{#190}":D$="
{#167} ":E$=" {#206}{#205}":F$=" {#20
6} {#165} ":F1$="{#183} {#183}"
140 G1$="      {#213}{#209}{#201} ":G$="
" {#213}{#201} ":H$=" {#202}{#203} "
:I$=" {#167}{#208}{#205}{#175}{#213}{#16
4 3}":J$=" {#188}{#167} {#202}"
150 K$=" {#170}{#205}":L$=" {#170}{#20
6}":L1$=" {#183 2}"

```

Chapter 19

USING ANIMATION IN GAMES

Draw-erase-redraw is the fundamental principle of animation, as we showed in the previous chapter. Knowing how to move an object across the screen is one thing; knowing how and when to incorporate animation effectively in your games, however, is another. As always, a clearly defined game concept is a prerequisite to the effective use of any computer technique, animation included.

We divide the uses of animation in Commodore 64 games into two broad categories. They are objects (planes, aliens, etc.) which move across the screen, and stationary characters which change in appearance. The first of these can be further subdivided.

There are three variations of cross-screen animation. These variations are horizontal movement, vertical movement, and diagonal movement. Screen objects can move in one direction, then another, but only in one direction at a time. Sprite animation is the most efficient way of accomplishing this, especially when the moving object is large and great speed is required.

Diagonal Animation

On the other hand, the `PRINT` and `POKE` methods of animation are not without their uses. The following program demonstrates how diagonal animation can be accomplished using both techniques. In the first part of this program (lines 1-150), the diamond (as string variable `A$`) is positioned using the handy `Plot` routine (lines 20 and 25). The `Plot` subroutine is called (`GOSUB 20`) immediately before a character is drawn or erased. The diagonal direction (in this case from the upper-left corner toward the lower-right) is

achieved by changing the values of the V (vertical) and H (horizontal) variables. The character is drawn, then erased (with blank space) using the same coordinates (V,H). Then V and H are incremented by 1 before the draw-erase process is repeated.

In the second half of the program, the POKE method is used. Since all 1000 screen positions have a unique memory location in the screen map, only one variable is needed to indicate the object's position (another advantage of the POKE method). A FOR/NEXT loop increments variable V by 41 each time the diamond (code 90) is displayed in a new location.

Why 41? Look at the screen map (Appendix A) and study the numbering system. If a character is displayed at 1024, a move down one space puts that character at location 1064. Location 1064 equals 1024 plus 40. Move one to the right of 1064 and you are at 1065. Downward diagonal movement to the right from location 1024 puts the character at 1065, right? Well, that's where 41 comes from.

As with the PRINT method, the character must first be erased at the old location before it can be redrawn elsewhere. Since the POKE method is a little faster, we put a pause loop on line 222. The POKE technique is smoother than the PRINT method, but with both there is a bit of a flicker as the character moves. This problem is largely solved by using sprite animation. Another nice feature of sprites is that the Commodore 64's internal machine language routines take care of the erasing step for you!

```
1 REM *****
2 REM *   DIAGONAL ANIMATION   *
3 REM * USING PLOT SUBROUTINE *
4 REM *****
10 GOTO100
15 REM *****
16 REM *   PLOT SUBROUTINE TO   *
17 REM * POSITION OBJECT ON SCR *
18 REM *****
20 POKE781,V:POKE782,H:POKE783,0
25 SYS 65520:RETURN
100 PRINT "{CLR}"
```



```

110 A$="#{#218}"
120 FOR I=0 TO 22
130 H=I:V=I
140 GOSUB 20:PRINT A$
145 GOSUB 20:PRINT " "
150 NEXT
190 GET X$:IF X$="" THEN 190
200 PRINT "{CLR}"
205 V=1024
210 FOR I=0 TO 22
220 POKEV,90:POKE V +54272,0
222 FOR J=1 TO 25:NEXT J
225 POKEV,32
230 V=V+41
240 NEXT

```

Horizontal Animation

Single-direction movement is easier. First, let's consider horizontal movement with `PRINT` statements. Maybe your game calls for a bird to fly across the screen. The `V` variable will stay the same, since your bird will be flying at a constant altitude. Change line 120 to `FOR I=0 TO 39`. Next, change line 130 to `H=I:V=5`. Now run the program.

If all went well, your diamond flew at a steady altitude across the screen. By changing the value of `V` in line 130, you can make the character fly at any altitude. As long as variable `V` is made equal to a real number rather than a variable, the altitude will remain constant.

Vertical Animation

To make the character move vertically, you must give the horizontal variable `H` a constant value while making `V` equal to a variable which changes value. Make sure that you keep within the screen boundaries. `V` must be in the range 0 to 24 while `H` can range from 0 to 255. If you want to avoid "wraparound" (seeing your character starting over again on the next line), keep variable `H` under 40.

Horizontal movement using POKES is accomplished quite easily. Change line 230 to $V=V+1$ and run the program. To create left-to-right movement, subtract from V. To create horizontal movement in the other direction (right to left), variable V is given the value of 39. Thirty-nine is the last column of each row. Line 210 should now read: `210 FOR I=39 TO 0 STEP -1.`

If you are not familiar with the STEP statement, a word of explanation may be useful. Most FOR/NEXT loops do not have the STEP statement. Without it the STEP, or increment, is assumed to be positive 1. STEP 2 would indicate an increment of positive 2. In the case of horizontal animation from right to left, an increment of negative 1 is needed; therefore the STEP -1. Try this program with STEP -2 and see what happens.

Vertical movement is accomplished by changing line 230 to $V=V+40$ for downward movement and $V=V-40$ for upward movement. We mentioned earlier the need for caution when POKING numbers into screen memory. To make sure that you do not run right off the screen and into another portion of your Commodore's memory, add this line:

```
235 IF V<1024 OR V>2023 THEN END
```

If you are working with a screen memory which has been moved, change 1024 and 2023 to correspond to the starting and ending memory location of your new screen memory.

Animation in Games

Now that you understand the concept of using variables to keep track of changing screen positions, let's again consider the uses of animation in games. One possibility is to present the player with a flying target, which is the case in our game *Interceptor*, Chapter 27. On the other hand, you may wish to display flying objects as a visual enhancement to a role-playing game. Characters could move through doors, behind trees, etc. Clouds could come across the sky to accompany the text message, "Suddenly you see the

storm approaching.” The animation does not have to be fancy to add a nice touch to a game which is otherwise mostly verbal.

If you are creating a maze game, the player will need an object to guide through the maze. Whether a mouse, a hero, a car, or whatever, the animation techniques we have discussed (as well as the sprite techniques we’ll get to later) will be useful.

We have discussed animation in which an object is drawn, erased, then redrawn in an adjacent space. You may want to devise a game in which an object moves (maybe reappears is a better word) far away from its previous position. Such an application could be in a role-playing game with a ghost shifting unpredictably from one part of the screen to another. Or maybe you want to construct a shooting gallery game. There could be a number of locations where targets suddenly pop up. In each case, animation principles are used.

By the way, animation need not be rapid. As long as the player has the ability to move a game piece from one screen location to another, such an action is subject to the rules and restraints of animation. You can adjust the speed of the moving character by increasing or decreasing the increments of the location variables.

In the next chapter we will continue our discussion of animation, showing how joysticks and game paddles can be used to control screen movement.

Chapter 20

JOYSTICKS AND GAME PADDLES

Anyone who has ever visited a video-game arcade is familiar with joysticks. The joystick is an input device consisting of a vertical stick or handle mounted on a rectangular base. A small button is mounted next to the stick itself. The information input to the Commodore 64 from a joystick corresponds to the direction in which the stick is pushed. In addition, the status of the button (pushed or not pushed) is also input.

There are two game control ports on the right side of the computer. One or two joysticks may be used at a time. Game paddle controllers may also be plugged into the control ports.

Joysticks

Joysticks consist of four switches which surround a vertical bar or handle. With the handle in the straight-up position, all the switches are open. As soon as the user pushes the joystick handle in any direction, one or more of the switches closes. A fifth switch is connected to the firing button; it closes when the button is pressed. Memory locations 56320 and 56321 contain the data from which we can determine the direction in which the joystick is being pushed and whether the button has been pressed.

Joysticks are most commonly used to control the movements of an object on the video screen. To demonstrate this, we have modified the POKE animation demonstration program.

```

1 REM *****
2 REM * ANIMATION DEMO USING JOY- *
3 REM * STICK CONTROL OF MOVEMENT.*
4 REM * F1 KEY OR FIREBUTTON      *
5 REM * DRAWS SIMPLE MAZE WALLS.  *
6 REM * THIS PROGRAM IS MODIFIED  *
7 REM * FROM KEYBOARD MOVEMENT    *
8 REM * CONTROL DEMO PROGRAM       *
9 REM * --USE JOYSTICK PORT 2--    *
10 REM *****
12 :
50 GOSUB 5000
60 GOTO340
100 REM *****
101 REM * JOYSTICK READ ROUTINE *
102 REM *****
110 JS=PEEK(56320)AND31:REM =READ JOYSTI
CK
115 FB=JS AND 16:REM =FIRE BUTTON?
120 JS=15-(JS AND 15):REM =DIRECTION
140 RETURN
340 POKEA,83:POKEA+C,0
350 GOSUB 100
355 IF FB=0 GOTO 430
360 IF JS=1 THEN AA=A-40:GOTO 410
370 IF JS=2 THEN AA=A+40:GOTO 410
380 IF JS=4 THEN AA=A- 1:GOTO 410
385 IF JS=5 THEN AA=A-41:GOTO 410
388 IF JS=6 THEN AA=A+39:GOTO 410
390 IF JS=8 THEN AA=A+ 1:GOTO 410
392 IF JS=9 THEN AA=A-39:GOTO 410
394 IF JS=10 THEN AA=A+41:GOTO410
395 GET A$:IF A$="{F1}" THEN 430
398 GOTO 350
410 IF PEEK(AA)= 32 THEN GOTO420
415 GOTO 350
420 POKEA,32:A=AA:GOTO 340
430 REM =BEGIN MAZE WALLS =
431 FOR I=1799 TO 1805:POKEI,64:POKEI+CL
,3:NEXT:
440 FOR I=1150 TO 1590 STEP 40

```

```

450 POKEI,93:POKEI+CL,7:NEXT
460 FOR I=1311 TO 1320:POKEI,64 :POKEI+CL,12:NEXT
470 FOR I=1090TO1460STEP40:POKEI,93 :POKEI+CL,4:NEXT
475 FOR I=1600 TO 1613:POKEI,67:POKEI+CL,8:NEXT
480 FORI=1613 TO 1813 STEP 40:POKEI,66:POKEI+CL,0:NEXT
500 GOTO340
5000 PRINT"{CLR}"
5010 PRINT"{DOWN 4}{RIGHT 4}USE THE JOYSTICK IN "
5020 PRINT"{DOWN 2}{RIGHT 4}IN PORT 2 TO CONTROL"
5030 PRINT"{DOWN 2}{RIGHT 4}MOVEMENT. TO CREATE A MAZE"
5050 PRINT"{DOWN 2}{RIGHT 4}PRESS THE F1 KEY OR FIRE "
5055 PRINT"{DOWN 2}{RIGHT 4}BUTTON ON THE JOYSTICK."
5060 FORI=1TO5000:NEXT
6000 CH=102:CL=54272
6001 PRINT"{CLR}":COL=10
6002 POKE650,130
6003 FORI=1024 TO 1063
6004 POKEI,CH:NEXT
6005 FORI=55296 TO 55335
6006 POKEI,COL:NEXT
6007 FOR I=1063 TO 2023 STEP 40
6008 POKE I,CH
6009 NEXT
6010 FOR I=55335 TO 56295 STEP 40
6011 POKE I,COL
6012 NEXT
6013 FORI=2023 TO 1984 STEP -1
6014 POKE I,CH:NEXT
6015 FORI=56295 TO 56256 STEP-1:POKEI,COL:NEXT
6016 FORI=1984 TO 1024 STEP-40
6017 POKE I,CH:NEXT

```

```
6018 FORI=56256 TO 55296 STEP-40
6019 POKE I,COL:NEXT
6020 REM GOTO2000
6021 A=1200
6022 C=54272
6050 RETURN
```

Lines 100-140 contain the subroutine which reads the joystick. Line 110 reads just those bits of location 56320 which give the status of the joystick switches. Line 115 determines whether the fire button has been pressed. Line 120 then evaluates the specific direction in which the joystick is being pushed. Variable JS can stand for the numbers 0-10. Those numbers are interpreted thus:

Value of JS Direction of joystick

- 0 none
- 1 up
- 2 down
- 3 (no function)
- 4 left
- 5 up and left
- 6 down and left
- 7 (no function)
- 8 right
- 9 up and right
- 10 down and right

Lines 355-394 change the value of JS into game actions. For instance, if JS is 1, variable AA is made equal to 40 less than A (the screen character's current position on the screen memory map). AA is therefore exactly one space above the current position of the character.

If JS equals 2, then AA is made equal to A plus 40—a move one space down. Notice that with the joystick, diagonal movements are provided for. If JS is 9, for example, AA becomes A minus 39, which makes the new character position up one and to the right of the old position.

FB is the variable which monitors the fire button. If the value of FB is 0, the button has been pushed. Any non-zero value means that the button is not being pushed. You can use this information to activate many actions. One obvious use is to fire a missile. Pressing the fire button can also send a spaceship into hyperspace, or it can signal the restart of a game that has been completed. Think of the fire button as an ON/OFF switch which can control any action.

Game Paddles

Game paddles are yet another input device for your Commodore 64. In a sense, a joystick is a pair of paddles combined into one unit. The paddles also plug into the control ports, one pair per port. Each paddle has a large round knob and a fire button. As the knob is turned, a numerical value from 0-255 is generated in locations 54297 (paddle A) and 54298 (paddle B). The fire button status (for both paddles) is registered at location 56321.

How you use the data input by the paddles is up to you. In our paddle demonstration program, we use the paddle values to control right/left movement.

```
0 REM *****
1 REM ** PADDLE DEMO PROGRAM **
2 REM *****
3 POKE53280,1 :PRINT"{BLK}"
4 PRINT"{CLR}":POKE1064,102:POKE1064+542
72,0:POKE1103,102:POKE1103+54272,0
8 POKE49152,120:POKE49153,96:REM KEYBOAR
D TURN-OFF ROUTINE
14 REM *****
15 REM * PORT 1 PADDLES *
16 REM *****
17 CL=54272
20 AA=1084:POKEAA,42:POKE AA +CL,0: GOTO
100
30 SYS 49152: POKE56320,64:A=PEEK(542
97):B=PEEK(54298):REM READ PADDLES
```

```

35 SYS 49152: B1=(PEEK(56321)AND 4)
37 SYS 49152: B2=(PEEK(56321)AND 8):REM
SECOND PADDLE
40 RETURN
100 GOSUB 30
102 IF B1=0 THEN PRINT"{HOME}{DOWN 9}FIR
E!";:FORI=1TO40:NEXT:PRINT"{LEFT 5}
"
110 IF A>76 AND A<176 THEN 100
111 IF A<76 THEN F=1
112 IF A>175 THEN F=2
115 IFF=1THENB=AA+1: IFPEEK(B)=32THENPOKE
AA,32:AA=B:POKEAA,42:POKEAA+CL,0:GOTO100
117 IFF=2THENB=AA-1: IFPEEK(B)=32THENPOKE
AA,32:AA=B:POKEAA,42:POKEAA+CL,0:GOTO100
120 GOTO100

```

This program is actually a very simplified version of the POKE animation routine we used to demonstrate the use of joysticks. Line 4 displays “blocks” at either end of the second row. The character, an asterisk, is guided back and forth along that row by paddle A.

In line 8, a 2-byte routine is stored in an out-of-the-way area of RAM. CL in line 17 is the offset for finding the correct location in color memory. CL is always added to a character’s location in screen memory to find the right location to POKE the color code. Line 20 displays the asterisk in the middle of the row and then jumps over the paddle-reading subroutine to the main body of the program.

Lines 30-40 read the paddles. The SYS 49152 temporarily disables the keyboard, allowing the status of the paddles and the buttons to be read. Location 49152 is where the short routine was stored in line 8. Variable A contains the value for paddle A. Variables B, B1, and B2 (which correspond to paddle B and the fire buttons) are not used in this demonstration program. The GOSUB 30, IF statements evaluate the paddle data and take appropriate action. The appropriate action here is the positioning of the asterisk.

One problem with game paddles is that their readings tend to be inaccurate in BASIC. This problem was easily eliminated here because we were only interested in whether the paddle was turned right, turned left, or in the middle. By defining left to be 0-75 and right to be 175-255, a wide middle range remains which causes the character to stay put. In other words, the user has to turn the paddle far to the right or far to the left to make the character move.

In each of these demonstration programs, we have used only one device—joystick or paddles. Try them both if you have joysticks as well as paddles. You'll probably prefer using joysticks because they can control movement in two directions rather than just one, as is the case with paddles.

Chapter 21

SPRITE GRAPHICS

One of the most powerful and attractive features of the Commodore 64 is its sprite graphics capabilities. But first of all, what is a sprite? Not a fairy, elf, or goblin, as the dictionary says, a sprite (also called a Moveable Object Block or MOB) is a user-defined shape that can be displayed and moved around the screen independent of text background.

Sprites are quite versatile. Sprites can be expanded in size horizontally, vertically, or in both directions. Sprites can be any color the Commodore can produce. Sprites can have more than one color (in the multicolor mode). Sprites can pass in front of or behind background objects. Sprites can even pass in front of one another according to a defined order of priority.

This all sounds great, you say, but do sprites have any negative features? Perhaps “negative” is too harsh a term. Let’s just say that sprites do have limitations. For one thing, unless you’re writing a very sophisticated machine language program, only eight sprites can be displayed on the screen at one time. For another, sprites can only be marginally enlarged. More significantly, the Commodore 64 does not support sprite graphics (nor high-resolution mode) with specific BASIC statements. Instead, PEEKs and POKEs must be used to manipulate sprites. This process is much less handy, but the results are well worth the effort.

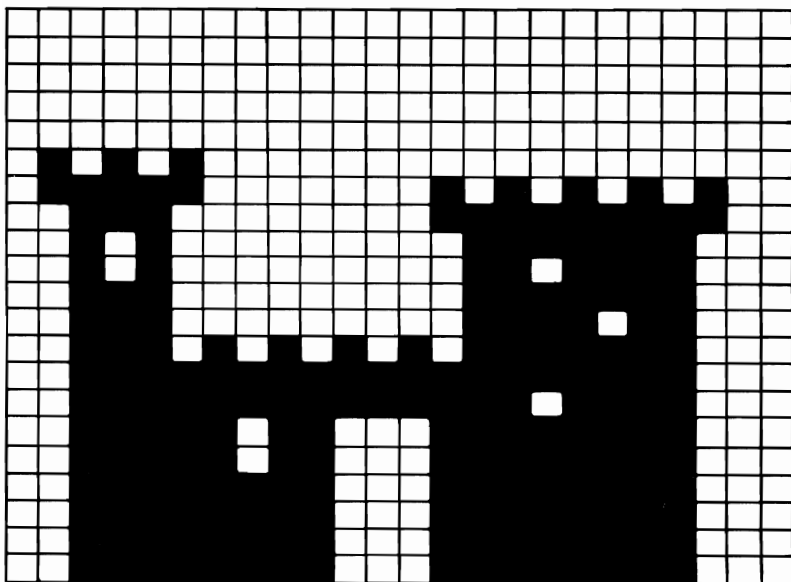
Sprites are very useful in games. Most action or arcade-style games have various objects flying around the screen: spaceships, missiles, animals, people, etc. Moving these kinds of objects is what sprites do best. Sprites can also be used as stationary graphic enhancements. For instance, in *Goal to Go!* (Chapter 31) sprites mark the

position of the ball on the playing field. It is also possible to display two or more sprites next to one another to make a single, larger image.

In effect, sprites are small areas of high-resolution graphics, yet they are much easier to use than the regular hi-res graphics mode.

Designing Sprites

Designing sprites is not unlike designing custom text characters. A sprite is designed on a 24-by-21 grid (See Appendix E). Design your sprite by filling in squares you want ON and leaving blank those you want OFF. Here is an example of a sprite designed on the grid:

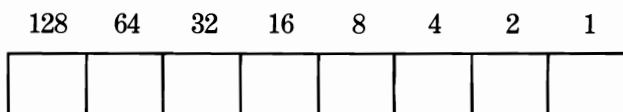


After designing your sprite, you must change the pattern of dots into numerical code. With custom characters, all you had to do was code 8 bytes. With sprites, however, each of the 21 rows is comprised of 24 spaces, which translates into 3 bytes per row. Multiply 3 (the number of bytes per row) by 21 (the number of rows)

and you have a total of 63 bytes. That's a lot more to code than with a custom character, but you get more when you're finished.

For each sprite you use in a game, you will have to include 64 bytes in DATA statements. That includes the 63 bytes mentioned in the previous paragraph, plus a sixty-fourth byte which is always 0.

There are many sprite-generating programs on the market today, and we have included a sprite-editing program at the end of this chapter. You may prefer to use one of them for designing sprites, or you may prefer to use paper and pencil (don't laugh, one of the authors of this book does it that way). If you code sprites by hand, you must calculate the values of 63 bytes. Each byte is calculated by adding the values of those spaces which are ON. The values are the same as with custom characters:



If all spaces are blank, the byte's value is 0. If all bytes are filled (ON), the value is 255 (derived by adding each of the individual values shown). If the left-most and right-most spaces were filled, that byte would have a value of 129 (128 or the left-most space added to 1 for the right-most). And so on.

When you are done, you will have 63 numbers, each representing a byte. Each number will be from 0 to 255. It is important to keep the numbers in order. The first number corresponds to the byte in the top row left position, the second number is top row middle, the third is top row right, the fourth second row left, the fifth second row middle, and so on through the sixty-third number, which is bottom row right. Don't forget, add a sixty-fourth number (always a zero) to every list of sprite data.

Storing Sprite Data

The next step is to store the data in your Commodore's memory. There are a number of areas, or pages, in each of the 16K memory

banks which can hold sprite data. As an example, let's assume you are using the default 16K memory bank, bank zero. In bank zero there are a number of good places to store sprite data. One place which is frequently used is memory area 832-1023. This area contains sprite pages 13, 14, and 15, beginning respectively at memory locations 832, 896, and 960. (There is one thing to remember about pages 13-15: They are found in the cassette buffer. If you use a cassette recorder for your data storage, you may want to avoid problems by storing sprite data in other areas of memory.) Perhaps a better area is the memory range 2048 to 4095. This area contains sprite data pages 32-63. It's unlikely that you will want to store data for more sprites than can be contained in this range.

Remember that every 64 bytes begins a new sprite data page. Page 32 begins at location 2048, page 33 begins at 2112, page 34 begins at 2176, etc. The trick is to load your 64 bytes exactly into one of the sprite pages. If your numbers are off by even a single location, your sprite will resemble a garbage heap. So the first sprite data byte must be in the first address of the sprite page. Here is an example using a FOR/NEXT loop. The sprite definitions are contained in DATA statements, which are first READ, then POKED into the appropriate address:

```
2100 FOR I=0 TO 63:READ A:POKE I+2048,A:
NEXT I
2110 DATA 0,0,63 .....
```

In line 2100, the 64 bytes (0 is byte 1) are read from the DATA statements and then POKED consecutively into memory locations, one after the other. Note that the loop begins with 0. That way we can use the starting address of the desired sprite data page as part of the formula for determining where to POKE the next byte. The first time through the loop, the first value of A is POKED into location 4098+0, which is 4098. The next time through the loop, the location is 4098+1, or address 4099. The second DATA value is POKED into 4099. The next time through the loop, the location is 4098+2, into which the third DATA value is POKED. And so it goes, until all 64 bytes defining the sprite are stored in the consecutive locations 2048-2111.

Filling Sprites with Data

At this point you have your sprite stored in a safe part of memory. Now comes the fun part! Using `POKEs` and `PEEKs`, you will tell the computer where to find data for a particular sprite, as well as when and where to display that sprite.

It might be a good idea to emphasize the difference between a sprite and the sprite data which define the sprite. A sprite is rather like a shoe box. The box has one specific number on it, but many different pairs of shoes can be placed inside. So it is with sprites. Sprite one is like the box, empty until you tell the computer where to go to get the data to put in the box. You can store many sets of `DATA` statements in memory, each defining a separate object. One or more sprites can use the same sprite data. As a matter of fact, all eight sprites can use the same data, or each sprite can use different data. But remember, no matter how many sets of sprite definition data in memory, only eight sprites can be displayed at once. Also, a particular sprite can only be displayed at one place at a time, so to display two missiles at once, for example, you must use two sets of sprite data.

Of course, you must tell your Commodore 64 where in memory to go to fetch the data to define a particular sprite. Each sprite has a 1-byte pointer in memory which holds this information. In the default memory configuration, these pointers are found at memory locations 2040-2047. That places them just a few bytes higher than screen memory (1024-2023). In fact, the sprite definition pointers are always the last 8 bytes of the 2K byte block which contains screen memory, even when the screen memory has been moved.

To tell your Commodore where to get data for sprite zero, you must `POKE` the correct value into location 2040. (If the screen memory has been moved, this location would change; remember, the pointer for sprite zero is always 7 bytes higher than the last screen memory location.) Assuming for now that you want sprite zero to use the data stored in page 13 (which begins at address 832 in the cassette buffer), you would use this line:

2110 POKE 2040,13:REM...13 POINTS TO LOCATION 832 (64*13)

If you wanted sprite zero to use data from sprite page 14, you would substitute 14 for 13 in line 2110. The trick is to keep track of where a particular set of sprite data begins. This can be done with REMark statements.

Turning Sprites ON and OFF

Turning sprites ON and OFF is not as simple as telling the computer where to find the correct definition data. Whereas each sprite has its own byte in memory to store the value which points to the correct data, all eight sprites must share a single byte controlling the ON/OFF status. The 8 bits of the byte at location 53269 are switches. Each sprite has its own bit, or switch. If the value of that bit is 1, then the sprite is displayed. If the value of the bit is 0, then the sprite is turned off.

It is also possible to simply POKE a number into 53269. Let's say you want to turn off sprite three. POKEing 53269,0 would certainly turn off sprite three, since sprite three's bit would then be a 0. The trouble is, all bits at location 53269 would now be 0, so all sprites would be turned off simultaneously!

To control single bits in a byte, we must use the logical operations OR and AND. We could go into a very lengthy discussion of how OR and AND function, but perhaps it would be better to simply provide you with the correct format. As long as you are careful to use the correct form, you'll get the job done.

To turn on sprite zero, use this line:

```
2115 POKE 53269,PEEK(53269) OR 1
```

The OR format alters just the bit which turns on the sprite you want to turn on. The following table shows the correct format for turning on each of the eight sprites:

Sprite	To turn on
0	POKE 53269,PEEK[53269] OR 1
1	POKE 53269,PEEK[53269] OR 2
2	POKE 53269,PEEK[53269] OR 4
3	POKE 53269,PEEK[53269] OR 8
4	POKE 53269,PEEK[53269] OR 16
5	POKE 53269,PEEK[53269] OR 32
6	POKE 53269,PEEK[53269] OR 64
7	POKE 53269,PEEK[53269] OR 128

Turning off a sprite also uses location 53269. Instead of POKEing a 1 into the appropriate bit, you will be POKEing a 0. Since we must use decimal numbers as counterparts to the computer's native binary numbers, this time AND (a logical operation) must be used. Don't worry if the format seems strange.

Sprite	To turn off
0	POKE 53269,PEEK[53269] AND 254
1	POKE 53269,PEEK[53269] AND 253
2	POKE 53269,PEEK[53269] AND 251
3	POKE 53269,PEEK[53269] AND 247
4	POKE 53269,PEEK[53269] AND 239
5	POKE 53269,PEEK[53269] AND 223
6	POKE 53269,PEEK[53269] AND 191
7	POKE 53269,PEEK[53269] AND 127

Selecting the color of a sprite is easier, since each sprite has a whole byte to store color information. Locations 53287 to 53294 hold the color information for sprites zero to eight, respectively. To make sprite zero black, you would include this line:

```
2120 POKE 53287,COL (WHERE COL STANDS
FOR ANY VALID COMMODORE 64 COLOR
CODE, 0-15)
```

To color sprite zero red, you would put a 2 in place of the symbol COL. Until you POKE another color value into location 53287, sprite zero will remain red, even if you change the shape of that sprite.

In the next chapter we will discuss how you position and move sprites on your screen.

Sprite Editing Program

```
1 REM *****
2 REM * SPRITE EDITOR PROGRAM *
3 REM *****
4 POKE53280,1:POKE53281,1:PRINT "{BLK}":P
RINT "{CLR}";:
5 PRINT "{DOWN 6}          {RVS} SPRITE ED
ITOR {ROFF}"
6 PRINT "{DOWN 2}  HELPS YOU CREATE SPRIT
ES FOR"
8 PRINT "{DOWN}  THE COMMODORE 64 COMPUTE
R."
9 PRINT "{DOWN 3}  (USE STANDARD CURSOR M
OVES.)"
10 FORI=1TO6000:NEXT
16 PRINT "{CLR}";:FORI=1TO21:FORJ=1TO24
18 PRINT ".";:NEXT J:PRINT:NEXT I
20 PRINT "{HOME}{DOWN}{RIGHT 26}{RVS} COM
MANDS: {ROFF}"
21 PRINT "{DOWN}{RIGHT 26} PRESS... "
22 PRINT "{DOWN}{RIGHT 26} {RVS}F1{ROFF}
TO PLOT"
23 PRINT "{DOWN}{RIGHT 26} {RVS}F3{ROFF}
NO PLOT"
24 PRINT "{DOWN}{RIGHT 26} {RVS}F5{ROFF}
ERASE"
25 PRINT "{DOWN}{RIGHT 26} {RVS}F7{ROFF}
DISPLAY "
26 FORI=0TO62:READ A:POKE 832+I,A:NEXT:
28 POKE 895,0
```



```

601 IF PX>23 THEN PX=23
602 IF PY<0 THEN PY=0
603 IF PY>20 THEN PY=20
605 POKE781,PY:POKE782,PX:POKE783,0
610 SYS65520:PRINTP$: RETURN
1000 REM *****
1001 REM * CALCULATE DATA STATEMENTS *
1002 REM *   "." IS CHR$(46)           *
1003 REM *****
1005 PRINT"{HOME}{DOWN 17}{RIGHT 25} {RV
S} PLEASE WAIT {ROFF}"
1012 DIM SD(505) :X=1
1020 FORI=1024 TO 1824 STEP 40
1030 FORJ=I TO I+23
1040 SD(X)= PEEK(J):X=X+1:
1050 NEXT J:NEXT I
2000 DIM BY(64):DIM DE(8)
2005 DE(1)=128:DE(2)=64:DE(3)=32:DE(4)=1
6:DE(5)=8:DE(6)=4:DE(7)=2:DE(8)=1
2100 X=1
2110 FOR I=1 TO 63
2120 FOR J=1 TO 8
2130 IF SD(X)=81 THEN SD(X)=DE(J):GOTO21
60
2140 IF SD(X)<>81 THEN SD(X)=0
2160 BY(I)=BY(I)+SD(X)
2170 X=X+1
2180 NEXT J,I
3000 FORI=0 TO 62:POKEI+832,BY(I+1):NEXT

3002 PRINT"{CLR} "
3005 PRINT"{HOME}{DOWN 12} YOUR SPRITE
LOOKS"
3008 PRINT" LIKE THIS ----->"
3010 POKE 53248,220:POKE53249,150
4000 :
4010 PRINT"{HOME}DO YOU WANT TO SEE THE
DATA"
4020 PRINT"STATEMENTS ON THE SCREEN (S)"
4030 PRINT"OR SENT TO PRINTER (P)?"

```

```

4040 PRINT"ENTER 'S' OR 'P'." :
4042 GET A$: IFA$="" THEN 4042
4050 IFA$="S" THEN PRINT"{CLR}": FOR I=1 TO 6
2: PRINT BY (I) ", "; : NEXT: PRINT BY (63): GOTO 5
000
4100 REM ***** PRINT DATA *****
4105 PRINT"{HOME}{DOWN 21} TO RUN PROGRAM
AGAIN, ENTER 'Y' ."
4110 OPEN 1,4
4120 CMD1
4130 FOR I=1 TO 62: PRINT BY (I) ", "; : NEXT: PRIN
T BY (63)
4140 CLOSE 1,4
4150 PRINT"{HOME}"
5000 PRINT"{HOME}{DOWN 21} TO RUN PROGRAM
AGAIN, ENTER 'Y' ."
5010 GET A$: IF A$="" THEN 5010
5020 IF A$="Y" THEN CLR: POKE 53269,0: GOT
O 16
5030 PRINT"THAT'S ALL": END

```


Chapter 22

ANIMATING SPRITES

Positioning Sprites

In the previous chapter we learned how to design sprites, how to store them in RAM in the form of `DATA` statements, how to choose their color, and how to turn them on and off. The only thing left to do is to figure out how to get them onto the screen!

Let's take this in two steps: First, we'll examine how to place the sprite at a given point on the screen. Second, we'll learn how to move a sprite around on the screen. To do this, it is important to understand the concept of Cartesian coordinates as found in algebraic graphs. A good example of this is as close as the glove compartment of your automobile.

Take a look at a street map. Along the sides there are letters, and along the top and bottom there are numbers. Look up any city in the map's index and find the coordinates. Let's say the coordinates are B and 5. To find that city, you place your finger on the letter B on one side of the map, then run your finger horizontally until it is directly under the number 5. There is the city you were looking for! The B and the 5 are coordinates. When used together they pinpoint a particular spot on the map.

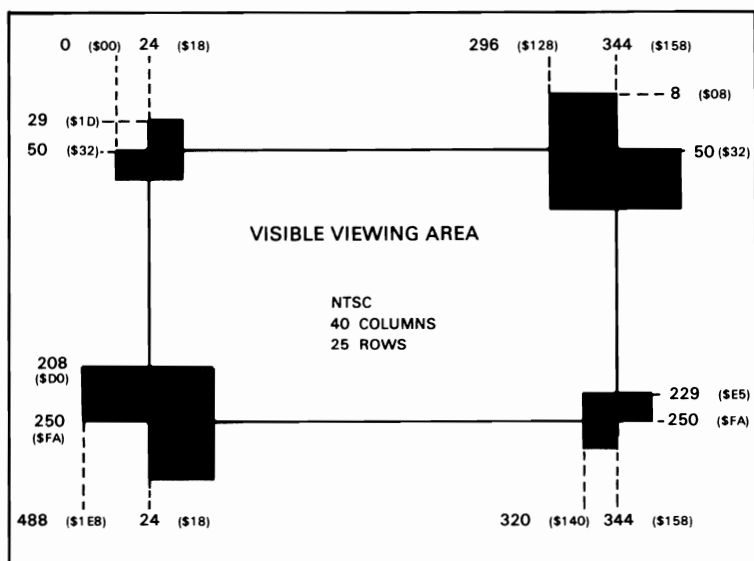
In the same way, the screen may be thought of as a grid, the main difference being that both horizontal and vertical coordinates are expressed in numbers. The Commodore 64's screen is a coordinate grid of 512 horizontal positions by 256 vertical positions. Position 0,0 is the upper left-hand corner. Position 511,0 is the upper right-hand corner. Position 511,255 is the lower right-hand corner. Position 0,255 is the lower left-hand corner.

The Most Significant Bit

Sounds simple, right? Well, there's a catch. Although there are 512 horizontal positions and 256 vertical positions, they are not all visible! The border around the screen acts as a frame or mask. The positions which are on the visible portion of the screen are:

In the horizontal (X), positions 24-343.

In the vertical (Y), positions 50-249.



This border effect is actually advantageous because sprites can enter the visible area bit by bit, rather than all at once. The sprite comes into view from off stage, so to speak.

There are 17 sprite position registers in the VIC-II chip. Each sprite has two registers, one for the X coordinate and one for the Y coordinate. (The seventeenth and last register will be discussed later.) Here are the X and Y register locations:

53248 Sprite 0- X coordinate

53249 Sprite 0- Y coordinate

53250 Sprite 1- X coordinate
53251 Sprite 1- Y coordinate
53252 Sprite 2- X coordinate
53253 Sprite 2- Y coordinate
53254 Sprite 3- X coordinate
53255 Sprite 3- Y coordinate
53256 Sprite 4- X coordinate
53257 Sprite 4- Y coordinate
53258 Sprite 5- X coordinate
53259 Sprite 5- Y coordinate
53260 Sprite 6- X coordinate
53261 Sprite 6- Y coordinate
53262 Sprite 7- X coordinate
53263 Sprite 7- Y coordinate

To display sprite zero at position 75,75, you would use this program line:

100 POKE 53248,75:POKE 53249,75

To display sprite one at position 100,63, this line would be used:

110 POKE 53249,100:POKE 53250,63

Keep in mind that, when positioning a sprite on the screen, the position indicated by the coordinates refers to the upper left-hand corner of the sprite's 24-by-21 grid. Even if the pixels in the upper left-hand corner are OFF, that is still the point of orientation. It's a good idea to experiment with sprites of various shapes to get a feel for moving a sprite on and off the visible area of the screen.

Before we go on to animation, there is one hurdle that must be jumped. It has to do with that seventeenth sprite position register mentioned earlier. More specifically, it has to do with horizontal positioning of sprites.

Each memory location is comprised of one byte. Each byte is, in turn, made up of eight bits. A bit can be a 0 or a 1; that is the basis of the binary numbering system which the Commodore 64 and other computers use internally. Early on we discussed the decimal equivalent of a byte in which all eight bit positions had the value of one: 11111111, which equals 255.

This works out just fine for the Y coordinate. There are 256 Y (vertical) coordinates, numbered 0-255. A single byte can hold any of these values. But there are 512 X (horizontal) positions. A single byte cannot hold a number higher than 255, so how are we going to display a sprite at any position greater than 255?

Enter memory location 53264, called the Most Significant Bit (MSB) register for the X coordinate. Up to and including X position 255, the X register for each sprite does just fine. Starting with position 256, the MSB register (location 53264) must also be used. Each bit in this location serves as the MSB register for a different sprite.

Consider this example. Suppose we have sprite zero which we want to display at coordinates 255,80. By **POKE**ing 255 into location 53248 and 80 into location 53249, the sprite will appear as planned.

Move the sprite one pixel to the right, however, and you have a problem. One plus 255 (the old X position) equals 256. A single memory location cannot hold a number greater than 255, so location 53248 needs help. The solution is to **POKE** a 1 into the correct bit of location 53264, the MSB register. At the same time, a 0 must be **POKE**d into location 53248. The Commodore adds the values stored in 53248 and the equivalent bit of 53264 to obtain the X coordinate. If the correct bit in 53264 has a 1, and 53248 has a 0, then the sprite is displayed at position 256. A 1 in any bit of 53264 equals 256. In fact, all the bits in 53264 can have only two values, 0 and 256 (represented by 1).

Now let's move the sprite one more pixel to the right. That makes position 257. The correct bit (bit 0) in 53264 still contains a 1, which stands for 256 in decimal numbers. Now **POKE** a 1 into location 53248. The 256 from the bit in location 53264 is added to the 1 in location 53248. One plus 256 equals 257, which is where the sprite is supposed to appear. To make the sprite appear at position 258,

a 2 (00000010 or 10 in binary) would be POKEd into 53248, and the contents of 53264 would remain unchanged. The 2 of 53248 and the 256 still in 53264 add up to 258, the new X coordinate of the sprite.

The maximum X (horizontal) value which can be stored for each sprite is the 256 from the MSB (53264) and the 255 from the X register (53248 for sprite zero). That adds up to 511; there are 512 horizontal positions, 0-511. Now you see why the sprite can be partially or totally off the screen. It is a function of the values stored in the X register and MSB register (0-511) compared to the number of pixels across each line (320).

Sprites in Action

A sample sprite-animation program, which sets up two sprites which use the same DATA statements, works as an example. (This routine is the basis for the animation portion of *Interceptor*, Chapter 27.)

```
100 REM *****
101 REM *   INTRODUCTORY SPRITE   *
102 REM *     DEMO PROGRAM #5     *
103 REM *****
105 PRINT "{CLR}":POKE53281,1:POKE53280,8
110 FORI=0 TO 62:READ A:POKE832+I,A:NEXT
115 POKE895,0
117 RESTORE:FORI=0TO62:READ A: POKE896+
I,A:NEXT
120 POKE2040,13:REM SPRITE 0 GETS DATA
    FROM PAGE 13
121 POKE2041,14
130 POKE 53269,PEEK(53269)OR 1:REM TURN
    ON SPRITE 0
131 POKE53269,PEEK(53269)OR2
140 POKE53287,0:REM SET SPRITE 0 COLOR
141 POKE53288,0
150 POKE53249, 70:REM -SPRITE 0 Y POSITI
ON
151 POKE53250,200 :Y1=200:POKE53251,Y
```

```

152 GOTO160
153 REM =====RND YO POSITION =====
155 YO=INT(RND(1)*50)+50:SD=INT(RND(1)*6
+2) :RETURN:REM-SET ALTITUDE AND SPEED
160 GOSUB 153
165 GETA$:IF A$="{F1}" THEN F=1:REM -IF
F1 KEY HAS BEEN PRESSED, SET FLAG 'F'
167 POKE53249, YO:REM -SPRITE 0 Y POSITI
ON
170 IF F THEN Y1=Y1-5:IF Y1<0 THEN Y1=20
0:F=0:REM MOVE PLANE VERTICALLY IF F=11
175 X=X+SD:REM -SET INCREMENT FOR BOMBER
CROSSING THE SCREEN
180 IF X>330 THEN SP=1:GOSUB153:REM -SEN
D BOMBER BACK TO LEFT IF PAST 330
182 REM --RESET BOMBER X,Y
185 IF SPTHEN POKE53287,1: X=0:POKE53264
,PEEK(53264)AND254:POKE53248,0:POKE53287
,0
186 IF X=0THENSP=0:REM -RESET SP FLAG TO
0 SO PLANE CAN FLY ACROSS AGAIN
190 IF X<256 THEN POKE53248,X:POKE53264,
PEEK(53264) AND 254:REM BOMBER'S X,Y
200 IF X>255 THEN POKE53264,PEEK(53264)O
R1:POKE53248,(X-256):REM IF PAST 255
205 POKE53251,Y1
210 GOTO165
215 REM POKE 53269,PEEK(53269)AND 255
220 REM X=0:POKE53248,0:POKE53264,PEEK(5
3264) AND 255:POKE53269,PEEK(53269)OR1
230 GOTO160
300 DATA 0,0,0,0,0,0,0,0,63,0,0,28,0,0,30,
0,192,15,0,224,7,128,240,3,192
310 DATA 255,255,254,255,255,255,255,255
,254,0,3,192,0
320 DATA 7,128,0,15,0,0,30,0,0,28,0,0,63
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

```

In this program, there is animation in both horizontal and vertical directions. A random number between 49 and 99 is selected in line

155. This value is used as the Y coordinate for the bomber sprite which flies left-to-right across the screen. Each time a plane crosses, its altitude is different due to this random number feature. Another random number (SD) is also selected in line 155. SD sets the number of spaces (pixels) the bomber will jump each time it is redrawn. The more spaces in each jump, the faster the plane will appear to fly.

In line 167, the Y coordinate of the bomber (Y0) is POKEd. In line 170, if the flag F has been set, the other plane is moved vertically. Flag F is set to 1 when the F(function)1 key has been pressed. The animation is done by decrementing (i.e., decreasing) the vertical coordinate of the second bomber (Y1). Notice that there is a check of Y1 which sets its value back to 200 when Y1 has reached 0. This has the effect of sending the plane back to the bottom. The last part of line 170 sets the flag (F) back to 0. This is done so the plane will wait at the bottom of the screen until the F1 key is pushed again.

Line 175 adds the speed factor (SD) to the X coordinate. This tells the computer where next to draw the first bomber, horizontally (the vertical, or Y, coordinate stays the same until a new bomber appears). The right-most position allowed in this program is 330. In line 180, flag SP is set to 1 if that position has been reached. Flag SP causes the first plane to return to the left of the screen (in line 185). The color of the plane is changed to white (POKE 53287,1) to eliminate a flicker as the plane is reset. The color is turned back to black at the end of the line.

The actual POKING of X coordinate values is done in lines 190 and 200. In 190, if the plane has not passed the 255 mark, its X position is POKEd into 53248, while a 0 is POKEd into the correct bit of 53264.

When the plane passes the 255 mark, the correct bit in 53264 is turned on, and a 0 is POKEd into 53248. The quantity X minus 256 is the difference between the 256 value of the bit in 53264 and the current X position. If the plane is at position 300, for instance, 1 is POKEd into 53264 (decimal 256). X minus 256 in this case is 300 minus 256, or 44, which is POKEd into 53248.

Line 205 sets the new vertical position (Y1) of the second sprite. The horizontal position of this sprite never changes, so it need not be reset each pass through the main program routine (lines 165-210). This is the opposite of sprite zero, which maintains the same vertical position all the way across the screen, but continually updates the horizontal position (X).

Turning the MSB ON and OFF

Here are the statements to use when turning the MSB ON or OFF. Remember, the remainder of a sprite's X (horizontal) position must be simultaneously POKED into the proper location (53248 for sprite zero, 53250 for sprite one, etc.).

Enabling and disabling MSB register.

Sprite number To enable (set to 1)

0	POKE 53264,PEEK[53264] OR 1
1	POKE 53264,PEEK[53264] OR 2
2	POKE 53264,PEEK[53264] OR 4
3	POKE 53264,PEEK[53264] OR 8
4	POKE 53264,PEEK[53264] OR 16
5	POKE 53264,PEEK[53264] OR 32
6	POKE 53264,PEEK[53264] OR 64
7	POKE 53264,PEEK[53264] OR 128

Sprite number To disable (set to 0)

0	POKE 53264,PEEK[53264] AND 254
1	POKE 53264,PEEK[53264] AND 253
2	POKE 53264,PEEK[53264] AND 251
3	POKE 53264,PEEK[53264] AND 247
4	POKE 53264,PEEK[53264] AND 239
5	POKE 53264,PEEK[53264] AND 223
6	POKE 53264,PEEK[53264] AND 191
7	POKE 53264,PEEK[53264] AND 127

Diagonal Sprite Movement

We have presented a sprite animation example in which one sprite moves horizontally and the other vertically. What about diagonal movement? In diagonal sprite animation, you must change both the X and Y coordinates every time the sprites are redrawn. A routine similar to the following would be needed:

```
100 FOR I=1 TO 200
110 X=X+1
120 Y=Y+1
130 GOSUB (PLOT X,Y POSITIONS)
140 NEXT I
```

This is a loop which changes the value of both the X and Y coordinates. In this case, the object would be moving from the upper left-hand corner toward the lower right-hand corner. To move right to left, or bottom to top, X and Y would have to start big and decrease ($X=X-1;Y=Y-1$). You need not have uniform increments. X could move more or less than Y. For instance:

```
100 FOR I=1 TO 100
110 X=I
120 Y=Y+2
130 GOSUB (PLOT X,Y POSITIONS)
140 NEXT I
```

Here we see the value of X equal to the value of the loop index variable (I). Y, on the other hand, is incremented by twos. The X,Y positions would be:

<u>X</u>	<u>Y</u>
1	2
2	4
3	6
4	8
5	10
etc.	

This pattern would produce a curving downward motion. There are many combinations of X and Y increments (or decrements—X could start at 250 and work downward), each producing a different movement. Once you have the basic concept, you will enjoy experimenting.

It is not too hard to move sprites with joysticks or paddles. Convert the directional information delivered by a joystick into X and Y values. For instance:

```
100 IF JS=1 THEN Y=Y-1
110 IF JS=2 THEN Y=Y+1
120 IF JS=4 THEN X=X-1
130 IF JS=8 THEN X=X+1
140 GOSUB (PLOT X,Y POSITIONS)
```

The value of JS (the variable which reflects which way the joystick is being pushed) determines how the X,Y positions will be updated before the sprite is redrawn.

Sprite Expansion

One more Commodore 64 sprite feature worth mentioning involves the expand and unexpand locations—53271 for vertical, 53277 for horizontal. To enable and disable expansion, use the same technique as with enabling and disabling the MSB register (53264). For instance, to enable vertical expansion for sprite zero, use:

```
100 POKE 53271,PEEK (53271) OR 1
```

To unexpand or disable sprite zero, use:

```
110 POKE 53271,PEEK (53271) AND 254
```

The POKEs and PEEKs are the same; the address (53271 or 53277 instead of 53264) is all that changes.

Sprites are a lot of fun. They make animation easier to use, especially when you want to animate more than one object at a time. They can be designed pixel by pixel, and can change size and color. In short, sprite graphics are a prime plus of the Commodore 64 computer and a powerful tool for your game construction kit.

Chapter 23

SAVING SPRITE DATA IN SEQUENTIAL FILES

A nice feature of your Commodore 64 is that it can help you do repetitive tasks. Few microcomputing tasks are more repetitive and boring than typing long lists of `DATA` statements. Why not let your C-64 do some of that work for you.

The way to save that typing time is to use sequential text files to store your `DATA` statements. Data that is stored in text files can be accessed easily by different programs without having to retype all those numbers! A good application of text files is with sprites. Let's say you design a beautiful sprite, or maybe more than one, which could be used in many different programs. Save the data for those sprites in a sequential text file, and you will never have to type them again. Future programs will include a routine which will access the file, input the data, and put them to use.

There are three types of data files in the Commodore world. They are sequential, random, and relative. The second and third are more complex and aren't particularly suited to this function, so we'll concentrate on sequential files.

Creating Sequential Files

Sequential files can be created on both tape and disk. Since disk files are so much handier, we'll use them here. The following sample program can be easily modified to work with tape, but be warned, the run time will be a lot longer.

```

10 REM *****
20 REM * SEQ FILE MAKER TEST *
21 REM * THIS PROGRAM CREATES A *
22 REM * SEQUENTIAL TEXT FILE *
23 REM * CALLED SPRITEDATA WITH *
24 REM * DATA FOR A SAMPLE SPRITE.*
25 REM * USE THIS PROGRAM AS A *
26 REM * SUBROUTINE IN YOUR PRO- *
27 REM * GRAM USING SPRITES. *
30 REM *****
40 PRINT "{CLR}"
50 PRINT "{DOWN 4}{RIGHT 7} {RVS} SQUENTI
AL FILE MAKER {ROFF}"
55 DIM SD(63)
58 FOR I=0 TO 62:READ A:SD(I)=A:NEXT
60 OPEN2,8,4,"O:SPRITED,SEQ,W"
70 FOR I=0 TO 62
80 PRINT#2,SD(I) :NEXT
90 CLOSE2,8,4
100 DATA 0,0,0,0,96,0,0,240,0,1,248,0
101 DATA 0,240,0,0,96,0,0,240,0,1,248,0
102 DATA 7,216,0,15,216,56,31,31,248,63
103 DATA 31,208,59,0,0,27,128,0,25,192
104 DATA 0,12,96,0,12,48,0,24,96,0,48
105 DATA 192,0,123,128,0,123,224,0

```

The File-Maker program asks you to type in the 63 bytes of data required to define a sprite (the sixty-fourth byte, 0, is supplied by the program). You will be asked to give the file a name.

Reading Sequential Files

The File-Reader program, which follows, accesses a saved file, storing the data in a memory location of your choosing. This program will be a module in any game which you want to access sprite data from a sequential file. Remember, by using this method, you do not have to type all those numbers every time you want to use a favorite sprite in a program.

```

10 REM *****
20 REM * SEQ FILE READER NUMBER 2 *
31 REM * LINES 60-80 READ DATA SPRITE *
    * DATA FROM SEQUENTIAL FILE *
32 REM * LINES 101-110 LOAD SPRITE *
    * DATA INTO MEMORY *
35 REM *****
40 PRINT "{CLR}"
50 PRINT "{DOWN 4} {RIGHT 7} {RVS} SEQUENT
IAL FILE READER {ROFF}"
55 DIMA(63)
60 OPEN2,8,4,"0:SPRITED,SEQ,R"
70 FOR I=0 TO 62
80 INPUT#2,A:POKE832+I,A:NEXT
90 CLOSE2,8,4
100 PRINT "{CLR}"
101 REM FOR I=0 TO 62
102 REM POKEI+832,A(I):NEXT
105 POKE895,0:REM-64TH BYTE MUST BE 0
109 REM *****
110 REM * LINE 120 DUMPS CONTENTS OF *
111 REM * 832-895 TO SHOW THAT DATA *
112 REM * HAVE BEEN LOADED THERE *
113 REM *****
120 FOR I=832 TO 895:PRINTPEEK(I):NEXT

```

If you are using tape files, make the following changes to these programs:

File Maker:

```

60 OPEN1,1,1,"(FILE NAME)"
80 PRINT#1,SD(I):NEXT
90 CLOSE 1

```

File Reader:

```

60 OPEN1,1,1,"(FILE NAME)"
80 INPUT#1,A:POKE 832+I,A:NEXT
90 CLOSE 1

```

Saving Screen Data

Another way to use sequential files in game construction is to save an entire background scene. The data for an entire scene can be saved in a text file just as sprite data can be saved. It does take a little longer to save and load, since there are 1000 bytes of data in a whole screen compared to 63 bytes in a sprite. The following program creates a screen pattern then saves the data for that screen to disk. The second part of the program retrieves the data and "paints" the picture on the screen. This saves having to type long PRINT statements again if you want to use the same screen background in another game.

```
10 REM *****
11 REM * SCREEN SAVE PROG LINES 30-130 *
12 REM * CREATE A SAMPLE SCREEN.      *
13 REM * LINES 200- SHOULD BE RUN ONLY *
14 REM * TO SAVE AND RECALL A SCREEN. *
15 REM *****
20 PRINT "{CLR}"
30 DIM A(20),B(20)
40 FOR I=1TO20
50 READ A(I),B(I):NEXT
60 DATA 18,9,13,12,20,10,11,16,22,7,17,1
5
70 DATA 20,7,11,18,16,22,19,8,13,15
80 DATA 13,20,18,10,15,7,19,9
90 DATA 23,8,15,17,9,20,11,18,15,6
100 FOR I=1TO20
110 FOR J=1TOA(I):PRINT " ";:NEXTJ
120 FOR J=1TOB(I):PRINT "{RVS} {ROFF}";:N
EXT J
125 PRINT
130 NEXT I
200 REM *****
201 REM * CREATE SEQ FILE      *
202 REM * WHICH STORES SCRNS *
203 REM * & COLOR CODES      *
204 REM *****
210 OPEN1,8,4,"0:TEXT SCREEN,SEQ,W"
```



```

220 FORI=1024 TO 2023:
230 PRINT#1,PEEK(I):PRINT#1,PEEK(I+54272
)
235 NEXT
240 CLOSE 1,8,4
250 PRINT"{CLR}":FORI=1TO2000:NEXT
300 REM *****
301 REM * READ SCRNM DATA FROM *
302 REM * FROM SEQ DISK FILE *
303 REM *****
310 PRINT"{CLR}"
320 OPEN1,8,4,"O:TEXT SCREEN,SEQ,R"
325 DIM X(1000),Y(1000)
330 FORI=1TO 1000
335 INPUT#1,X(I),Y(I):NEXT
340 CLOSE1,8,4
400 REM *****
401 REM * POKE SCREEN DATA *
402 REM *****
410 FORI=1TO1000
420 POKEI+1023,X(I):POKEI+1023+54272,Y(I
):NEXT

```

This technique works much too slowly with tape files to be practical, but if you would like to try it, make the appropriate line changes, as indicated in the earlier sprite data program.

Chapter 24

COMMODORE 64 SOUND

There has been a fair amount of publicity about Commodore graphics, especially concerning sprites, but less has been said about the Sound Interface Device (SID). But the Commodore's sound capabilities are perhaps even more impressive than its graphics, because the SID chip allows you to control the voice, frequency, waveform, note, attack, decay, sustain and release of musical tones.

For those with some background in music, the flexibility of the SID should be more than adequate to produce a variety of musical effects. And for those of us who don't know much about music, Commodore sound is, nonetheless, impressive and exciting. In fact, learning to create sound on the Commodore is a good way to learn about music. There are memory locations (the sound chip) which address specific functions: volume, frequency, etc. You POKE numbers into these memory locations to produce sounds. The SID chip is a full-blown sound/music synthesizer. The SID memory locations run from 54272 to 54296. These are the locations into which you POKE the numbers (0-255) to control the sound characteristics enumerated above. Appendices O and E in the *Commodore 64 Programmer's Reference Guide* contain the complete specifications of the SID.

SID Features

SID features include three voices, four waveforms (types of sound), three envelope generators (attack/decay/sustain/release), multiple voices, and ring modulation. The voices are really tone oscillators or tone types. The waveforms help replicate the sounds of different musical instruments. For example, the sawtooth waveform sounds

like a harpsichord, but by decreasing the numbers for attack and decay, the tone becomes more like that of a banjo. The waveforms available on SID are (1) triangular (flute-like), (2) sawtooth (harpsichord), (3) pulse (piano), and (4) noise (blast-off, static, etc.).

The attack/decay/sustain/release settings can be used to vary sounds from long-tone buildup and long finish to short (staccato) sounds. Altering these settings can create contrasts which are similar to the differences between a harpsichord and a banjo or a prolonged rocket blast-off and a sudden gunshot.

The SID in Action

To get going, type in these few lines and run the program:

```
10 V=54296:W=54276:H=54273:L=54272
20 POKE V,15:POKE W,33:POKE A,15
30 FOR X=200 TO 5 STEP-2:POKE H,10:
POKE L,X:NEXT
40 FOR X=150 TO 5 STEP-2:POKE
H,20:POKE L,X:NEXT
50 POKE W,0
```

This short program should produce a weird monster-like sound. Now change lines 20-40 as shown below:

```
20 POKE V,15:POKE W,129:POKE A,15
30 FOR X=200 TO 5 STEP-2:POKE
H,20:POKE L,X:NEXT
40 FOR X=150 TO 5 STEP-2:POKE
H,10:POKE L,X:NEXT
```

Run the program again. Now we hear a blast-off. The new sound results from the change in the waveform (W,33 to W,129) from sawtooth to noise, and also from the change in frequency (POKE H).

Frequency settings are used in conjunction with POKEing notes into the registers. The SID chip is capable of sounding eight octaves. Both the *Commodore 64 User's Guide* and the *Commodore 64 Programmer's Reference Guide* show the codes for each note. However, one octave is enough for a good start. If you wanted to play a song from sheet music, you could look on one of these charts and input the note values via DATA statements accompanied by the required steps (clearing the sound chip, setting the starting location, etc.), and your song would be ready to play. To play a single note, you must POKE a value into both the high frequency and low frequency locations. Obviously, the process of playing a song can get complicated, but remember, you only have to write a program once, and then it is there to use over and over.

The following program plays a song with a harpsichord tone:

```
5 S=54272:REM CLEAR SOUND CHIP
10 FOR L=S TO S+24:POKE L,0:NEXT
20 POKE S+5,9:POKE S+6,0
30 POKE S+24,15:REM SET VOL TO MAX
40 READ HF,LF,DR
50 IF HF<0 THEN END
60 POKE S+1, HF:POKE S,LF
70 POKE S+4,33
80 FOR T=1 TO DR:NEXT
90 POKE S+4,32:FOR T=1 TO 50:NEXT
100 GOTO 40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125
140 DATA 32,94,750,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1
```

The program may seem mysterious, but it's really quite simple. The only differences between this program and the blast-off and monster sound programs are (1) the variables are redefined, and

(2) the DATA statements contain the information for notes and high/low frequencies for the song.

Instead of defining a separate variable for each memory location, the variable S is defined as the starting location of the sound chip (54272), and then the additional functions are added to that location. For example, since 54296 controls volume, S+24 (54272+24=54296) has the same effect as would another variable, V, set to 54296. In order to write a song or copy it from sheet music, you will need a chart giving you the high and low frequency values for each note in the octave you wish to play the song. Here the appropriate frequencies have been referenced and incorporated into the DATA statements.

```
70 POKE S+4,17
90 POKE S+4,16:FOR T=1 TO 50:NEXT
```

Change lines 70 and 90 in your song program and run it. Now the sound has changed from a harpsichord to a piano. The difference results from the waveform change (33/32 to 17/16).

These simple examples should give you a taste of what is possible, but it is up to you to further explore the sound functions on your own to fully experience the synthesizer functions of the Commodore 64. In Chapter 35, where *Scary Hall* is presented, we'll show you how we incorporated some of the SID's capabilities.

Part 5

CREATING ACTION GAMES

Chapter 25

COMPUTER SIMULATIONS

In this and the next two sections we will study the construction of two types of advanced computer games. We are calling these action games, and strategy and fantasy/adventure games. There is an area of overlap in these terms. An action game can certainly involve a degree of strategy, and strategy/adventure games need not be totally lacking in action.

For our purposes, an action game is one in which the player must respond to rapidly moving objects displayed on the video screen. There are two main types of computer action games: One is the shoot-em-up, while the other could be called the run-for-your-life game. In the first, the object of the game is to shoot at and hit targets. In the second, the player tries to avoid being hit or gobbled up by some object or creature.

Many of the popular arcade-style computer games are really combinations of the two types. *Galaxian*, *Pac-Man*, *Asteroids*, and *Defender* come to mind. In each of these, the player must avoid being hit (or eaten) as well as shoot (or eat) other on-screen objects. Our *Monster Maze* (Chapter 26) is, in a limited way, a hybrid of the two types. *Interceptor* (Chapter 27) is a pure example of the shoot-em-up. (Remember, that a game of one kind or the other can be made a hybrid—hence, more fun to play—by adding routines to supply the missing element.)

Computer games are often thought of as simulations. The field of computer simulations is fascinating, because computer simulations give the appearance of reality. In some cases, games simulate real-world events; in others, they simulate imaginary events. Let's examine two types of simulations: continuous and discrete.

Simulations: Continuous and Discrete

Continuous simulations use mathematical formulas to represent physical models. In a continuous simulation, all physical aspects of the event being simulated have parallel mathematical formulas. The mathematical formulas are continuously calculated, just as the real world is constantly undergoing physical influences (e.g., gravity, wind, inertia, friction). A good example of this is NASA's computer simulation of a lunar lander module. Every relevant physical element of the spacecraft's descent to the moon's surface is formulated mathematically. This type of simulation is very complex and requires extreme sophistication in the fields of mathematics and physics.

Unlike NASA, we do not use continuous simulations in this book. Instead we use discrete simulations. In a discrete simulation, real-world events are represented in a more arbitrary manner than in a continuous simulation. Random numbers are frequently used to determine the outcome of events. In a continuous simulation, mathematical formulas closely matching physical processes determine those outcomes. Some specific examples may be helpful.

“Throwing” a Ball: An Example

Assume that you want to simulate a baseball pitch. A continuous simulation would entail the writing of various mathematical formulas modeling the velocity of the ball, the effect of wind and gravity, etc. Formulas representing the physical skills of a particular (or hypothetical) pitcher would also be needed. All in all, such a simulation would present quite a challenge to any programmer.

On the other hand, a much simpler and more practical discrete model could be created. You would first define the scope of the simulation. Are you simulating any pitcher of a given level, or are you simulating Nolan Ryan? In either case, you could do a little research to determine the probability (based on known statistics) of the pitcher throwing a ball or strike. Let's assume that your research informs you that the pitcher has a 53 percent probability of throwing a ball on any given pitch. A random number between 1 and 100 can be calculated ($PBALL = RAND(1)*100+1$). If the number is 53 or less, the pitch is declared a ball.

That is a lot simpler than going to your local physicist-baseball fan for help in designing a mathematical model of every inch of the ball's flight from the pitcher's hand to the catcher's glove. You may say, "Yes, but the discrete simulation method with its random number isn't as realistic as the mathematical, continuous simulation." That may or may not be true. If you have very good statistics, the discrete method may be just as valid. In this case, "very good statistics" might include data covering balls and strikes thrown during day games, during early innings opposed to late innings, during good weather and bad, during April, during September, etc.

The more accurate your data, the more realistic your simulation. We have been talking about the highest level of accuracy here. Most of your games will get along fine with a lower level of validity. After all, you're just simulating a game, not the national budget for the next decade!

Making Decisions with Random Numbers and Variables

The important thing to remember is that you will need to define all the decision points required to make your game convincing and fun to play. Your game road map is a good place to list these points. Then you can design a routine for each decision point. If the information delivered by those routines is sufficiently valid to allow your game to progress as planned, you have succeeded!

Random numbers are not the only way to make decisions in simulation games. Variables will keep track of important elements. For instance, say you are writing a shoot-em-up game. Every time your player fires a missile or throws a tomato, you will want to decrement a variable. The variable, call it THROWS, will decrease by 1 every time the player fires. The variable THROWS will be checked each time through the main program routine. When THROWS equals 0, the program will jump to a subroutine telling the player that the game is over. Here's an example to try on your Commodore:

```
100 S=5: PRINT CHR$(147)
110 PRINT "PRESS -F- TO FIRE"
120 GET A$: IF A$="" GOTO 120
```

```

130 IF A$<>"F" GOTO 120
140 S=S-1
150 GOSUB 170:IF S=0 THEN GOTO 200
160 GOTO 120
170 PRINT "FIRE!"
180 RETURN
200 PRINT "=NO MORE SHOTS=":END

```

Line 100 sets the initial value of S, which is the number of shots allowed each game. The same line also contains the -CLEAR SCREEN- statement. Line 110 prompts the player. In line 120, a keypress is awaited using GET. If the result of the GET is a null (""), the line is repeated. When a key is finally pressed, it is analyzed in line 130; if the key is not F, the program branches back to line 120 to GET another keypress. When F is finally pressed, the SHOTS variable (S) is decremented by one in line 140. In line 150, the FIRE subroutine at 200 is called, then the value of F is compared with zero. If S equals zero, the game ends; otherwise, there is a branch back to 120 to start the cycle again.

There will be many variables other than one keeping track of how many shots the player has left. The essence of this technique is that a given variable's value is periodically checked. When the value is greater than, less than, or equal to a set number, something happens. What happens is up to the game programmer (sorry about that).

Now we will present some concrete examples. *Monster Maze* is primarily a run-for-your-life game. The player has some offensive weapons, but they are limited in number and only serve to help the player successfully negotiate the maze. *Interceptor*, on the other hand, is a pure example of shoot-em-up. As the game stands, the target is incapable of shooting back. Maybe you already have some clever ideas that will make the game more challenging by changing that situation.

Chapter 26

MONSTER MAZE

Monster Maze is a Commodore 64 action game of the run-for-your-life variety. The object of the game is to successfully guide a character, in this case a heart, to the top level of a maze. Here is a road map for *Monster Maze*:

1. The player sees a maze consisting of seven horizontal lines which define levels.
2. The heart appears on the lowest level. The I, J, K, and M keys guide the heart.
3. The heart is moved to higher levels by moving it through gaps, or gateways, which appear randomly.
4. After a time, a monster appears from the left border and tries to devour the heart.
5. The player may fire three missiles per game to stop the monster.
6. Three hearts are given to the player per game. After guiding the heart, the player may play again. The player may choose the level of difficulty on additional games. The player's total points are announced at the game's end.

While not a long game from a programming point of view, a lot happens in *Monster Maze*. While this version of the game is keyboard controlled, a joystick routine can easily be substituted.

Routines and Subroutines

There are several basic characteristics in this game which we suggest you adopt as standard practice when designing Commodore 64 action games. For one, all one-time routines are placed at the end of the program. A one-time routine is simply one which must be performed at the beginning of a game, but not again. This usually means initializing variables, `READING DATA` statements, etc. Another point is that frequently used subroutines are placed near the beginning of the program. Finally, there is a main program routine which calls and coordinates all other portions of the program (except the one-time parts).

Now let's look at *Monster Maze* in detail. Lines 1-25 contain `REMARKS`, the title screen, and playing instructions. Locations 53280 and 53281 are `POKE`d to set the screen and border colors. The initial difficulty level is set by making `W` equal to 60. This sets the time which elapses before the monster appears. Lines 910-917 define the array `MP(8)`. Each element in this array contains the starting screen memory location from which the monster appears. This information is used in the main program routine.

In line 918, the heart's starting screen memory location and level (`LINE`) and the color offset (54272) are defined. The `POKE 650,128` causes all keys to repeat automatically. This makes maneuvering the heart much easier. In each game, the player has three chances to get the heart to the top. The player must keep track of how many hearts have been played.

Lines 921-940 draw a border of dots around the screen. This border prevents screen characters from going out of bounds. The subroutine at 770-870 draws the seven horizontal lines, each with a randomly placed gap, or gate, through which the heart is moved. Line 800 uses `RND` to select where on each line the gap will begin. Line 820 `PRINTS` the gap, and line 830 calculates how many spaces lie between the gap and the right border. The continuation of the horizontal line is drawn there.

The main game routine starts at line 500. Line 540 displays the heart character. Variable `A` contains the current screen memory location of the heart, and `A+C` indicates the corresponding color

memory location. Line 550 looks for the player's input with the GET A\$. A typical way of using GET is: 100 GET A\$: IF A\$="" THEN 100. This sets up a fast loop which continues until a key is pressed. In *Monster Maze*, a GET loop exists, but much has to take place within the loop. In fact, the program flow does not come back to line 550 (unless a key has been pressed) until line 715.

Let's examine closely what happens in the main routine. After the GET A\$ in 550, variable X is incremented. When X reaches 100, the routine to redraw lines and gaps (770-860) is called. Therefore, even if the player does nothing, the lines will be redrawn every few seconds. In line 560, variable M is incremented. When M reaches the value of W (which represents the difficulty level), the monster appears. Lines 570-660 control the monster's movements. The familiar draw-erase-redraw principle is again at work. MS and M2 are the current locations of the monster's two halves.

The Monster Rides

MS is the starting point for the monster. MS needs to be defined only at the beginning of each monster ride. In line 580, array MP(7) determines the starting screen memory positions. The flag MO=1 in line 600 prevents MS from being redefined until a subsequent dash by the monster.

The monster is comprised of two characters, one directly above the other. The draw portion of the animation POKES a zero (the @) into the first two positions. The erase part POKES 32 (a space) into those positions. The redraw part would usually POKe the original character into the next space to the right of the original space, but not in this case.

The monster travels twice as fast as the heart. Therefore, we must look at the next two positions of the monster. But, if only the positions two jumps down the line were checked, it would be possible for the monster to jump right over the heart. MS-39 and MS+1

are the screen positions immediately to the right of the monster, while T1 and T2 are *two* positions to the right. PEEK is used to determine what lies in those four new positions. If PEEK finds a heart (code 83), the player-destroyed subroutine at 440 is called. This subroutine resets variables, reduces the players score by 200 points, and checks to see if the last heart has been lost. If the last heart is a goner, the endgame routine at 875 comes up. If PEEK shows that the right border has been reached (line 630), the monster is erased and reset (M=1:MO=0).

The Heart Moves

The movement of the heart is controlled in lines 670-750. The I, J, K, and M keys are used for up, left, right, and down directions. Variable A is the heart's current position. The intended new position is variable AA, which is calculated by adding to, or subtracting from, A, depending on the key pressed. In lines 720-750 the heart's move is performed.

For those who would prefer to use a joystick to guide the heart, we have included a joystick routine at the end of the chapter, after the keyboard version of the game. Notice that the main difference between these two approaches is that the joystick uses PEEK to check direction of movement instead of the GET found in the keyboard version.

To keep track of the heart's location, two variables are used. These are LINE and LEVEL. On each game level there are two lines on which a character can move. Program lines 730 and 732 check to see if the heart has moved from one of the lines on a level to the other, or whether a move to a whole new level is being made. A vertical move of plus or minus 40 spaces on the screen memory map means a vertical movement of down one row or up one row, respectively. If the new position is 41 (or more) less than the old position, the heart has moved to a new level (through a gap). That situation is handled in line 730. The reverse case is provided for in 732.

Game Levels

There are seven levels, so LEVEL can be any number from one to seven. LINE, on the other hand, is always a screen-map number. LINE starts out equal to 1945, which is the second location on the last line of the lowest level. If a move to the next higher level is made, LINE is decremented by 120. A decrease of 120 is a shift of three rows ($3 * 40$). Since each level has three rows, a solid line and two maneuvering rows, a move of 120 spaces is necessary to jump to the next higher, or lower, level.

If the player presses the space bar, it will be detected in line 715. This causes the missile subroutine at line 30 to fire away. The missile-firing sequence is rather short, so while the missile fires, nothing else moves. The missile sequence uses a loop. Variable A is the heart's current location. The loop starts at A-1 (the space to the left of the heart) and goes to A-15. This is a backward-counting loop, which is why we use STEP -1. In line 40, there is a check for a collision with the monster.

When the top level has been reached, the game-won routine at 1000 announces the total points so far, and gives the player a chance to play again and to choose the level of difficulty. The difficulty level is based on how much time elapses before the heart reaches a level and the moment the monster appears. The greater the value of variable W, the longer the wait. The longer the wait, the better chance the player has to get to the next higher level. You may want to change the scoring to suit your tastes. For instance, *Monster Maze* assesses a 200-point penalty every time the heart gets gobbled up by the monster. Perhaps you'll want to make the penalty consistent with the difficulty level.

The thing to keep in mind about this type of game is that many things occur as the game executes its main routine. Either you are moving something on the screen or variables are being updated. Remember, nothing will happen unless you make it happen, and you make things happen by creating variables that keep an eye on the action!

Monster Maze: Keyboard Version

```
0 REM *****
1 REM *
2 REM * M O N S T E R *
3 REM * M A Z E *
4 REM *
5 REM * COPYRIGHT (C) 1983 *
6 REM * BY WILLIAM RUPP *
7 REM *
8 REM *****
9 CLR:W=60:POKE53281,1:POKE53280,1:PRINT
  "{BLK}"
10 REM *****
11 REM * TITLE AND INSTRUCTIONS *
12 REM *****
13 PRINT"{CLR}{DOWN 3} (RVS} M O
  N S T E R "
14 PRINT"{DOWN 2} (RVS} M A Z
  E "
15 PRINT"{DOWN 3} GUIDE YOUR HEART TO
  THE TOP!"
16 FORI=1TO4000:NEXT
17 PRINT"{CLR}{DOWN 3} USE THE I AND M
  KEYS TO MOVE":PRINT"{DOWN 2} UP AND
  DOWN."
18 PRINT"{DOWN 2} USE THE J AND K KEYS
  TO MOVE":PRINT"{DOWN 2} LEFT AND RI
  GHT."
19 PRINT"{DOWN 2} THE SPACE BAR FIRES M
  ISSILES":PRINT"{DOWN 2} (3 MISSILES
  PER GAME)."
20 FORI=1TO10000:NEXT:POKE53280,7
21 REM *****
22 REM * INITIALIZE VARIABLES *
23 REM *****
24 REM -SEE 918
25 GOTO 910
27 REM *****
28 REM * LASER BLAST *
29 REM *****
```

```

30 REM :
35 LB=LB+1:IF LB>3 THEN RETURN
38 FOR L=A-1 TO A-15 STEP -1
40 IFPEEK(L-1)=81 OR PEEK(L)=81 THEN 4
3
42 POKEL-1,31: POKE L-1+C,0:POKEL,32:NEX
T
43 POKEL,32:POKEMS,32:POKEM2,32:M=1:MO=0
:RETURN
50 POKEMS,32:POKEM2,32:RETURN
435 REM *****
436 REM * HEART DESTROYED *
437 REM *****
440 PRINT"{CLR}{DOWN 5}{RIGHT 5}{RVS}PLA
YER DESTROYED{ROFF}"
450 MO=0:LEVEL=1
460 PLAYER =PLAYER+1:TP=TP-200
470 IF PLAYER>3 THEN 880
480 FORQQ=1TO3500:NEXT
482 A=1970
485 PTS=0
490 GOTO922
500 :
510 LINE=1945
530 REM *****
535 REM * M A I N P R O G R A M *
538 REM * R O U T I N E *
539 REM *****
540 POKE A,83:POKE A+C,2
550 GET A$:X=X+1:IF X= 100 THEN GOSUB 77
0
560 M=M+1:IF M<W THEN 670
570 IF MO=1 THEN 590
580 MS=MP(LEVEL):PV=LEVEL
590 T1=MS-38:T2=MS+2:M2=MS-40
600 MO=1
610 IF PEEK(MS-39)=83 THEN 440
615 IF PEEK(MS+1)=83 THEN 440
620 IF PEEK(T1)=83 THEN 440
625 IF PEEK(T2)=83 THEN440
630 IF PEEK(T1)=81 THEN POKEMS,32:POKEM2

```

```

,32:M=1:MO=0:GOTO670
640 POKEMS,32:POKEM2,32
650 POKET1,0:POKET1+C,5:POKET2,0:POKET2+
C,5
660 MS=T2:M2=T1
670 IF A$="" THEN 550
680 IF A$="I"THEN AA=A-40:GOTO720
690 IF A$="J"THEN AA=A-1:GOTO720
700 IF A$="M"THEN AA=A+40:GOTO720
710 IF A$="K"THEN AA=A+1:GOTO720
715 IF A$=" " THEN GOSUB 30:GOTO550
720 IF PEEK(AA)<>32THEN AA=A:GOTO550
730 IFA< LINE-40 THEN P2=1
732 IFA> LINE+39 THEN P3=1
738 IF P2 THEN LEVEL=LEVEL+1:M=1:MO=0:PO
KEMS,32:POKEM2,32
739 IF P2 THEN LINE=LINE-120:P2=0:IF LEV
EL>7GOTO 1000
745 IF P3 THEN LEVEL=LEVEL-1:LINE=LINE+1
20
748 IF P3 THEN M=1:MO=0:POKEMS,32:POKEM2
,32:P3=0
750 POKEA,32:A=AA
760 GOTO 540
765 REM *****
766 REM *      DRAW HORIZONTAL      *
767 REM *          LINES          *
768 REM *****
770 PRINT"{HOME}{DOWN}";:
780 FOR R=1TO7
790 PRINT"{DOWN 2}";:
800 PRINT"{#209}";:G=INT(RND(1)*34+1)
810 FOR I=1 TO G:PRINT"{#192}";:NEXT
820 PRINT" ";:
830 X=38-(G+4):IF X=0 THEN 850
840 FORI=1TOX:PRINT"{#192}";:NEXT I
850 PRINT"{#209}";:
855 X=0
860 NEXT R
865 IF PEEK(A)=64 THEN A=A+40:POKEA,83:P
OKEA+C,0

```

```

870 RETURN
875 REM *****
878 REM *   PLAYER DESTROYED   *
879 REM *****
880 POKE 650,0:POKE 198,0
890 PRINT"{CLR}{DOWN 3} SORRY, YOU HAVE
BEEN DESTROYED!"
900 PRINT"{DOWN 2}{RIGHT 2} YOU FINISHED
WITH "TP " POINTS."
902 END
905 REM *****
906 REM * START OF LEVEL ARRAY. *
907 REM * EACH NUMBER IS LEFT- *
908 REM * MOST POKE FOR LINE   *
909 REM *****
910 DIM MP(8)
911 MP(1)=1945
912 MP(2)=1825
913 MP(3)=1705
914 MP(4)=1585
915 MP(5)=1465
916 MP(6)=1345
917 MP(7)=1225
918 A=1970:LINE=1945:C=54272:POKE650,130

919 LEVEL=1:A=1970:POKE650,128:REM LOCAT
ION 650 SETS REPEATING KEYS
920 PLAYER=1
921 PRINT"{CLR}";:
922 PRINT"{CLR}";:M=1:MO=0
923 FORI=1024 TO 1063
924 POKEI,81:NEXT
925 FORI=55296 TO 55335
926 POKEI,0:NEXT
927 FOR I=1063 TO 2023 STEP 40
928 POKE I,81
929 NEXT
930 FOR I=55335 TO 56295 STEP 40
931 POKE I,0
932 NEXT
933 FORI=2023 TO 1984 STEP -1

```

```

934 POKE I,81:NEXT
935 FORI=56295 TO 56256 STEP-1:POKEI,0:N
EXT
936 FORI=1984 TO 1024 STEP-40
937 POKE I,81:NEXT
938 FORI=56256 TO 55296 STEP-40
939 POKE I,0:NEXT
940 I=0
941 GOSUB 770
942 GOTO 500
950 END
995 REM *****
996 REM *   GAME HAS BEEN WON   *
997 REM *****
1000 PRINT"{CLR}{DOWN 4}   {RVS} YOU HAV
E MADE IT TO THE TOP! {ROFF}"
1002 PTS=7*(100-W): TP=TP+PTS:LB=0
1005 POKE650,0:POKE198,0: REM - TURN OFF
REPEAT KEY, EMPTY KEYBOARD BUFFER-
1010 PRINT"{DOWN 2}           YOU HAVE "TP "
POINTS.":PTS=0
1020 PRINT"{DOWN 2}IF YOU WANT TO CONTIN
UE, INPUT Y"
1028 GET C$:IFC$= "" THEN 1028
1030 IF C$ ="Y" THEN GOTO 1050
1040 PRINT"{DOWN 2}  THANKS":END
1050 LB=0:PRINT"{DOWN 2} ENTER DIFFICULT
Y FACTOR;1-9"
1052 PRINT"{DOWN 2}           (9 IS THE MOST
DIFFICULT)
1055 GET W$: IF W$="" THEN 1055
1058 W=VAL(W$):IFW=0 THEN PRINT"PLEASE I
NPUT A NUMBER":FORI=1TO4000:GOTO1050
1060 IF W=1 THEN W=90
1065 IF W=2 THEN W=80
1070 IF W=3 THEN W=70
1080 IF W=4 THEN W=60
1090 IF W=5 THEN W=50
1100 IF W=6 THEN W=40
1110 IF W=7 THEN W=30
1120 IF W=8 THEN W=20

```

```
1130 IF W=9 THEN W=10
1150 GOTO 919
```

Monster Maze: Joystick Routine

Add the following lines:

```
100 REM *****
101 REM * JOYSTICK READ ROUTINE *
102 REM *****
110 JS=PEEK(56320)AND31:REM-READ JOYSTIC
K
115 FB=JS AND 16:REM -BUTTON PRESSED?
120 JS=15-(JS AND 15):REM -DIRECTION?
130 RETURN
```

Delete lines 670-715, and add the following lines:

```
670 IF JS=1 THEN AA=A-40:GOTO 720
675 IF FB=0 THEN GOSUB 30
680 IF JS=2 THEN AA=A+40:GOTO 720
685 IF JS=4 THEN AA=A- 1:GOTO 720
690 IF JS=5 THEN AA=A-41:GOTO 720
695 IF JS=6 THEN AA=A+39:GOTO 720
700 IF JS=8 THEN AA=A+ 1:GOTO 720
705 IF JS=9 THEN AA=A-39:GOTO 720
710 IF JS=10THEN AA=A+41:GOTO 720
```


Chapter 27

INTERCEPTOR

By now you're probably asking yourself, "What happened to the classic arcade-style shoot-em-up game?" In response to that question, we present *Interceptor*.

Interceptor contains most of the elements of an arcade game. An airplane flies overhead past buildings, a truck carrying a missile moves along a road, and the player attempts to hit the plane by firing the missile. As the program unfolds, you will see the advantages and limitations of programming action games in BASIC.

Again, as in the other programs, we begin by setting real variable values to 0. Line 12 sets SHots to 50 and HiTs (HT), FLag, and TRies to 0. ML equals VL (vertical height) plus 15. Other real variable values are set in their respective subroutines.

Line 20 clears the screen and sets border, screen, and cursor color. Here we made the background gray (color 15) and the cursor black. The background buildings and the clouds are drawn with keyboard graphics (lines 35-100). Note that the color of the cursor, which also determines the background color, is changed to white in line 35 and to blue in line 44. This allows the clouds to be white and the buildings blue. The clouds are drawn using the curved lines on keys U, I, J and K, as well as with the straight lines on various keys. Again, there is no point in religiously copying the picture in our listing if you can improve on it. In any case, some variation will not hurt.

The buildings are drawn with straight lines on keys H, N, T, Y, and P. Also, the corners on keys O, P, @, and L and the squares on keys A, S, D, F, Z, X, C, and B were used for windows and

building edges. Fences were made with the E, R, and + keys. The flag used the B key. Once the clouds and buildings have been drawn, the cursor color is changed back to black in line 100.

Airplane, Truck, and Missile Sprites

Line 30 first calls subroutine 5000 then subroutine 4000. Let's go to 5000, which creates the three sprites (airplane, truck, and missile). Line 5020 sets the value of V to the beginning of the VIC II chip (53248). This makes all future sprite memory location references simpler to program. For example, instead of coding the sprite activation location 53269, it is coded $V+21$. ($V+21=53248+21$). This technique is a matter of choice and is not mandatory. If desired, you may substitute the actual memory locations for the sprite pointers rather than setting the value of V at the beginning location.

In line 5030, $V+21,7$ POKES the beginning memory location and enables the three sprites. Line 5040 POKES the color red, sets the beginning horizontal position at 150, and sets the vertical position at 229 for the first sprite, which is the truck. Lines 5042-5045 POKE the color dark grey, set the horizontal position at 0, and set the vertical position at 60 for the second sprite, the airplane. Line 5048 POKES the color yellow, sets the horizontal position at 165 ($VL+15$), and sets the vertical position at 205 for the third sprite, the missile.

Lines 5050-5054 set the memory locations for each of the three sprite shapes, which are defined in the DATA statements (lines 5100-5300). It is important to understand that each sprite must have a separate memory location set aside in which to store its shape. As described in Chapter 21, the DATA statement determines the sprite shape, with the first 63 numbers (lines 5100-5160) defining the shape of the first sprite, the second 63 numbers (lines 5170-5230) defining the shape of the second sprite, and the third 63 numbers (lines 5240-5300), defining the shape of the third sprite. Each READ statement (lines 5050-5054) instructs the system to READ the appropriate lines. POKE V1, Q1 . . . instructs the system to POKE these shapes into the specified memory locations. After defining the sprite shapes and storing them in the proper memory locations, the program RETURNS from subroutine 5000.

Subroutine 4000 displays the game instructions on the screen by using `PRINT` statements. It is also possible to place the background drawing in a separate subroutine instead of placing it at the beginning of the program. Generally, subroutines placed at the beginning of a program will execute faster than those at the end of a program. However, in a fairly short program (under 6K) like this one, the placement of subroutines probably won't make a perceptible difference.

Coordinating Sprite Movement

In our road map for *Interceptor*, we must lay out each of the functions of the player-program interaction. At first, this seems quite simple: the airplane flies overhead; the truck moves; and the missile fires, either hitting or missing the plane. However, since two objects cannot move simultaneously, the need for creating the illusion that the objects are moving at the same time makes the actual programming quite challenging

Since we have already defined the shapes and positions of each of the sprites, we can now concentrate on setting out the action of the sprites. First, we must make make player-input via the keyboard possible. Lines 110-130 and lines 210-240 allow this input. Lines 110 and 210 use the `GET` statement to receive keyboard input without `RETURN`ing, making the action much faster than if the player had to press `RETURN` after each play. `GET` acts immediately upon a key press. Here we have written instructions to `GOSUB` upon the pressing of specific keys. The J and K keys move the truck left and right (remember your instructions to the player). Subroutine 800 changes the sprite (truck) location by decrementing VL five points upon each press of J (moving to the left on the screen). Subroutine 850 increments VL five points, causing the opposite effect when K is pressed.

If you have programmed sprite movement, you may have already had the interesting experience of getting the "ILLEGAL QUANTITY" error when your sprite moved near the right edge of the screen. The problem is that the screen is wider than the 255 maximum allowable points for sprite positioning. This means that you either have to reset the memory location (`V+16`) to allow move-

ment from position 256 to the screen edge or forget having the sprite move past 255. Since seeing the plane suddenly stop 2 or 3 inches from the right edge of the screen and then reappear at the left edge would seriously damage the game's simulation effect, you probably agree that the problem must be solved rather than avoided. However, we can deal with the problem of the truck movement by simply adding an error trap to prevent the ILLEGAL QUANTITY error from ending the game. Lines 857-860 do this.

Let's concentrate for a moment on moving the plane across the screen. We want the plane to move continuously throughout the game, whether or not the player fires the missile. Line 160 begins the movement routine. Subroutine 155 selects the random numbers which are responsible for the plane's speed and position, which will vary according to the numbers selected. Line 175 actually moves the plane, its movement determined by the value of SD. The plane begins its journey with a horizontal value of 0 at the left side of the screen. On each pass the altitude is determined by the random number X3. Thus, the plane will move at a different speed and altitude each pass across the screen, providing the player with the challenge of estimating the best moment to fire the missile and the best position for the truck.

Lines 190-200 deal with the problem of moving the plane beyond position 255 at the right of the screen. Line 185 briefly turns the plane sprite to the background color (53240,15) so the player is unaware of the shifting that takes place when the position (beyond 255) is POKED to allow the plane to continue past the right screen edge. If this is not done, the image of the plane flashes at horizontal position 255 as it moves to position 330. SP is a flag for this position which resets the sprite color to background, preventing the flash. Try the program after taking out line 185 to see this effect. (In order to be able to test segments of your program, it is a good idea to type in lines 2000 and 2040, and also line 240. These will allow you to type E to end the game until you get it working correctly.)

In any case, we now have an airplane moving across the screen. We should also have the shape of a truck and a missile on the truck. The next step is to create a way to fire the missile. Line 235 accepts a G from the keyboard and sets the values of both G and

F to 1; then, moving to 166, calls subroutine 5500. Subroutine 5500 produces the sound of the missile firing (gunshot). We will return to the sound effects later. IF G equals 1, THEN FL is set to 1. This (line 170) moves the missile. At the top of the screen, the missile is POKEd back to the vertical position of the truck (Y1=205) and the FLag is reset to 0. After firing, the missile moves upward in increments of five point positions until it reaches the point above the screen top, then it is seen on the truck again, as if a new missile were waiting to be launched. Since the missile cannot be in two places at once, the player cannot fire another missile while one is still in the air.

When Sprites Collide

Now the interesting part of the game: the actual contact of the missile with the plane and the resulting explosion. As you may already have discovered, the Commodore has a sprite collision register, enabling the programmer to tell when a sprite has either hit the background or another sprite. Again, as in the other manipulations of sprites, you must POKE memory locations. Line 171 determines if and when such a collision occurs. The 6=6 is used for a collision between sprites two and three, because there are three sprites and the formula is $2+4=6$, where 2 equals sprite two and 4 equals sprite three. Line 171 says that in the event of a collision, call subroutine 6000, increment the value of HT (hit) by 1, and GOTO 175 (move the plane).

Graphics and Sound Effects

Now, we will take a look at subroutines 5500 and 6000, the sight and sound effects of the game.

Subroutine 5500 creates the missile-firing effect. As in other games with sound, we must set variable values at the beginning memory location of the Commodore 64 SID chip. G1 is the beginning memory location, GW is the waveform (noise), GA is the attack, GH is the high frequency, and GL is the low frequency. Line 5520 sets the volume to maximum and the waveform to noise. Lines 5530-5540 define the attack/decay, setting up a loop for high and

low frequency settings. Line 5550 increments the variable TR for each missile firing. In line 5560, the game ENDS when the player has made 25 TRies.

In subroutine 6000, lines 6010-6025 produce the visual effect of the explosion. Screen and border colors are changed back and forth from yellow to purple to orange and black, with pauses in between to give the effect of repeated explosions. This is a really simple way to produce literally smashing effects. This is also an easy area in the game to change to produce your own unusual visual effects. Line 6026 POKES the horizontal locations for sprites two and three. In line 6029, X2 is set at 0 (plane horizontal position).

Lines 6030-6065 produce the collision sound effects. As in the case of the missile firing, the noise waveform is used, and the volume and attack/decay are also the same. However, the high and low frequency loops are different, producing a prolonged noisy sound. Line 6065 is a pause loop, and it resets the FLag variable to 0. Line 6066 returns the missile to the truck. Line 6070 RETURNS.

Keeping the Action Going

As you can see, there is quite a trick to constructing a series of loops which keep the action going in the manner desired. Although this is a relatively short program, the design was far more challenging than that of *Magic Cards*, for example. By now, you should realize that some programs which are conceptually simple may generate quite a bit of code, whereas some conceptually complex programs may be surprisingly short. You should also realize that it is important to spend quite a lot of time planning your program prior to writing the code. Remember the programmer's comment that the sooner you begin writing code, the longer the program will take to complete.

Although creating a game such as *Interceptor* can be a brain-racking experience, the result usually produces a rare satisfaction and sense of accomplishment. One last suggestion: Don't be satisfied simply copying the program. We provide a place to begin, not end. Vary the code. Change the sprite shapes. Change the colors. Explore!

Interceptor

```
9 REM *****
10 REM *** INTERCEPTOR ***
11 REM *****
12 SH=50:HT=0:ML=VL+15:X4=205:TF=20:FL=0
:TR=0
20 PRINT"{CLR}":POKE 53281,15:POKE53280,
15:PRINTCHR$(144)
30 GOSUB 5000:GOSUB 4000
35 PRINT:PRINT"{WHT}"
38 PRINT"      {#213}  {#201}      {#213}{#2
01}  {#213}{#201}  {#192}{#201}{#163}  "
39 PRINT"      {#213}  {#213}{#201}{#163}
{#163 3}{#201}      {#213}{#201}  ";"
{#163 2}{#202}{#203}  {#163 3}"
40 PRINT"      {#202}  {#213}{#203}{#202}{#
201}  {#213}{#163 3}{#203}  {#163}  ";"
      {#202}  {#203}{#163 2}  "
41 PRINT"      {#202}{#203}  {#203}{#163}
      {#163}{#202}{#203}{#163}  ";"
      {#202}{#203}  "
43 PRINT"      {#202}{#164 3}  {#202 3}{#16
4 5}{#203}  {#203}  {#203}  "
44 PRINT:PRINT"{BLU}"
45 PRINT"      {#206}{#163 7}{#206}{#212}  {
#172 2}      {#217}{#205}{#163 7}{#205}"
50 PRINT"      {#207}{#183 6}{#208}  {#212}
{#207}{#208}  ";"      {#217}  {#207}{#183 6
}{#208}  "
60 PRINT"      {#180}      {#170}  {#212}  {#
180}{#170}  ";"      {#217}  {#180}{#172}{#1
87}  {#172}{#187}{#170}  "
70 PRINT"      {#207}{#183 10}{#204}{#175}{#1
80}{#170}  ";"      {#217}  {#204}{#188}{#19
0}  {#188}{#190}{#170}  "
80 PRINT"{#207}{#183 15}{#208}";"      {#2
17}  {#204}{#175 6}{#186}{#175}  {#221}{#1
91 2}{#180}"
90 PRINT"{#180}{#172}{#187}  {#172}{#187
```

```

} {#172}{#187} {#172}{#187} {#170}";"
  {#217} {#180} {#208} {#221} "
91 PRINT"{#180}{#188}{#190} {#188}{#190}
} {#188}{#190} {#188}{#190} {#170}";"
  {#217} {#207}{#208}{#175}{#207}{#208}
{#175}{#207}{#208}{#175}{#207}{#208}{#17
5}{#207}{#208} "
92 PRINT"{#180} {#176}{#174} {#207}{#1
83}{#208} {#176}{#174}{#176}{#174} {#170
}";" {#217} {#180}{#190} {#1
90}{#170} "
93 PRINT"{#180} {#173}{#189} {#180} {#
170} {#173}{#189}{#173}{#189} {#170}";" {
#164 4}{#217} {#180}{#190 2}{#188}{#190}
{#188}{#190 2}{#188}{#190}{#188}{#190} {
#170} "
94 PRINT"{#178 7}{#180} {#170}{#178 6}{#
170}";" {#206} {#205} {#180}{#219 14}"
95 PRINT"{#177 6}{#206} {#205}{#177 5}
{#170}";" {#205}{#180}{#219 14}"
100 PRINT"{BLK}"
110 GET B$
120 IF B$="J" THEN GOSUB 800
130 IF B$="K" THEN GOSUB 850
140 GOTO 160
155 X3=INT(RND(1)*50)+50:SD=INT(RND(1)*6
+4):RETURN
160 GOSUB155
165 GET A$:IF A$="G" THEN G=1:FL=1
166 IFG THEN GOSUB 5500:G=0
167 IF A$="E" THEN GOTO2000
168 POKE V+3,X3:REM SPRITE 1 Y POS
170 IF FLTHEN Y1=Y1-5:IF Y1<0 THEN Y1=
205:FL=0
171 IF PEEK(V+30)AND 6=6 THEN POKEV+5,Y1
:GOSUB6000:HT=HT+1:GOTO 175
175 X2=X2+SD
180 IF X2>330 THEN SP=1:GOSUB 155
185 IF SPTHENPOKE 53240,15:X2=0:POKEV+16
,PEEK(V+16)AND253:POKEV+2,0:POKEV+40,0
186 IF X2=0THEN SP=0

```



```

190 IF X2<256 THEN POKE V+2,X2:POKEV+16,
PEEK(V+16) AND 253
200 IF X2>255 THEN POKEV+16,PEEK(V+16) O
R 2:POKE V+2,(X2-255)
205 POKE V+5,Y1
210 GET B$
220 IF B$="J" THEN GOSUB 800
230 IF B$="K" THEN GOSUB 850
235 IF B$="G" THEN G=1:FL=1:GOTO 166
240 IF B$="E" THEN GOTO 2000
250 GOTO 165
800 VL=VL-5
805 MI=VL+15
806 IF VL<=0 THEN VL=0
807 IF MI<15 THEN MI=15
808 IF MI>255 THEN MI=255
809 IF VL>255 THEN VL=255
810 POKEV+0,VL:POKEV+4,MI
820 RETURN
850 VL=VL+5
852 MI=VL+15
855 IF VL>245 THEN VL=245
857 IF MI>255 THEN MI=255
860 POKEV+0,VL:POKEV+4,MI
870 RETURN
2000 REM END
2005 HT=HT/2
2010 PRINT"YOU MADE ";HT;" HITS OUT OF "
;TR;" TRIES!":PRINT
2011 SC=(HT*100)/TR
2012 PRINT"YOUR SCORE IS ";SC;" PERCENT"
:PRINT
2020 IF SC>75THEN PRINT"CONGRATULATIONS!
":PRINT
2025 IF SC>50 AND SC<76 THEN PRINT"YOU D
ID OK--BUT NOT GREAT!":PRINT
2030 IF SC<=50THEN PRINT"BETTER LUCK NEX
T TIME!":PRINT
2040 END
4000 REM INSTRUCTIONS
4010 PRINT"{CLR}":PRINT"YOU WILL SEE AN

```

```

AIRPLANE FLY OVER"
4020 PRINT"THE BUILDINGS.  A MISSILE IS
MOUNTED ON"
4030 PRINT"A TRUCK.  IT IS YOUR JOB TO F
IRE THE"
4035 PRINT"MISSILE, ATTEMPTING TO HIT TH
E PLANE."
4040 PRINT"J MOVES THE TRUCK TO THE LEFT
, K TO THE"
4050 PRINT"RIGHT, G FIRES THE MISSILE, A
ND E ENDS      THE GAME."
4060 PRINT"YOUR SCORE IS BASED UPON ATTE
MPTS/HITS."
4070 PRINT:PRINT
4080 PRINT"{BLU}"
4100 PRINT:PRINT:INPUT"PRESS RETURN TO C
ONTINUE. ";XZ$
4110 PRINT"{CLR}":RETURN
5000 REM SPRITE
5010 RESTORE
5020 V=53248:VL=150:X2=0:VH=60:Y1=205
5030 POKE V+21,7
5040 POKEV+39,2:POKEV+0,VL:POKEV+1,229
5042 POKEV+40,11
5043 POKE V+2,X2
5045 POKE V+3,VH
5048 POKEV+41,7:POKEV+4,VL+15:POKEV+5,20
5
5050 FOR V1=12288 TO 12350:READ Q1:POKE
V1,Q1:NEXT
5052 FOR V2=12352 TO 12414:READ Q2:POKE
V2,Q2:NEXT
5054 FOR V3=12416 TO 12478:READ Q3:POKE
V3,Q3:NEXT
5060 POKE 2040,192:POKE2041,193:POKE2042
,194
5070 POKE V+23,1:POKEV+29,1
5100 DATA 0,15,248,1,15,248,63,6,48
5110 DATA 39,6,48,255,15,120,255,255,255
5120 DATA 255,255,255,60,15,60,24,6,24
5130 DATA 0,0,0,0,0,0,0,0,0

```

```

5140 DATA 0,0,0,0,0,0,0,0,0,0
5150 DATA 0,0,0,0,0,0,0,0,0,0
5160 DATA 0,0,0,0,0,0,0,0,0,0
5170 DATA 0,0,0,0,0,0,0,0,63,0
5180 DATA 0,28,0,0,30,0,192,15,0
5190 DATA 224,7,128,240,3,192,255,255,25
4
5200 DATA 255,255,255,255,255,254,0,3,19
2
5210 DATA 0,7,128,0,15,0,0,30,0
5220 DATA 0,28,0,0,63,0,0,0,0
5230 DATA 0,0,0,0,0,0,0,0,0
5240 DATA 0,0,0,0,0,0,0,0,0
5250 DATA 0,0,0,0,0,0,0,0,0
5260 DATA 0,1,0,0,3,128,0,3,128
5270 DATA 0,3,128,0,3,128,0,3,128
5280 DATA 0,3,128,0,3,128,0,3,128
5290 DATA 0,3,128,0,7,192,0,15,224
5300 DATA 0,31,240,0,28,112,0,24,48
5310 RETURN
5500 REM GUNSHOT
5510 G1=54296:GW=54276:GA=54277:GH=54273
:GL=54272
5520 POKEG1,15:POKEGW,129:
5530 POKEGA,15
5535 POKEGH,40:POKEGL,200
5537 FOR GX=100 TO 5 STEP-5:POKE GH,40:P
OKEGL,GX:POKEG1,GX:NEXT
5540 POKEGW,0:POKEGA,0
5550 TR=TR+1
5560 IF TR=25 THEN GOTO 2000
5570 RETURN
6000 REM HIT PLANE
6010 POKE 53281,7:POKE 53280,7:POKE 53
280,4:POKE 53281,8
6015 POKE 53281,0:POKE 53280,0:POKE532
81,8:POKE53280,8
6020 FOR PAUSE=1 TO 50:NEXT PAUSE
6022 POKE 53281,0:POKE 53280,0
6023 FOR PAUSE =1 TO100:NEXT PAUSE
6025 POKE 53281,15:POKE 53280,15

```

```

6026 POKE V+3,0:POKEV+5,0
6029 X2=0
6030 POKEG1,15:POKEGW,129:POKEGA,15
6040 FOR GX=200 TO 5 STEP-2:POKEGH,40:PO
KEGL,GX:NEXT
6050 FOR GX=150 TO 5 STEP-2:POKEGH,10:PO
KEGL,GX:NEXT
6060 POKEGW,0:POKEGA,0
6065 FOR PAUSE=1 TO 100:NEXT PAUSE:FL=0
6066 Y1=200
6070 RETURN

```

The following lines show the keyboard graphics characters which comprise the illustration in *Interceptor*. See the complete listing, above, for the ASCII equivalents (Appendix F) of these characters.

```

38 PRINT"
39 PRINT"
40 PRINT"
41 PRINT"
43 PRINT"
44 PRINT:PRINT"
45 PRINT"
50 PRINT"
60 PRINT"
70 PRINT"
80 PRINT"
90 PRINT"
91 PRINT"
92 PRINT"
93 PRINT"
94 PRINT"
95 PRINT"

```

Part 6

STRATEGY AND FANTASY/ADVENTURE GAMES

Chapter 28

STRATEGY AND MAP GAMES

We now move from the visually oriented action games to more thoughtful games involving strategy and role playing. For many centuries people have enjoyed games requiring great concentration and thought. Chess is an especially good example of this. It is also a good illustration of how a basically intellectual game can still have a major graphic element.

The combination of graphics, sound, and intellectual challenge provide us with the most sophisticated games, and the Commodore 64 is an excellent vehicle for the construction of such games. Fantasy/adventure games, which we'll focus on in the second half of this section, involve the player in a very personal way. But before we consider these games, we'll examine strategy and map games.

Strategy and map games cover those simulation games which give the player a sense of control over a large real-life endeavor. Most of us aren't army generals, but with our Commodore 64, we can create and control our own campaigns or refight famous battles from the past. Similarly, our computer allows us to coach football teams, invest fortunes in the stock market, and in general participate in many grand activities beyond the scope of our normal lives.

For our purposes, the word "strategy" refers to the manner in which the player participates in the game. In action games, the player takes part directly in the activity. Consider a military theme, for example. In the action game, the player drives the tank, moves the turret, and fires the gun. In the strategy game, however, the player indirectly commands tank units by giving orders to move in a certain direction, attack enemy positions, and so on.

Likewise, if the subject of the game is football, the strategy game allows you to select offensive plays, defensive formations, etc. In an action football game, the player would actually control a ball-carrier's movements on the field. A mixture of the two types is another possibility. Many newer arcade games allow the player to move a character around a background while simultaneously considering how present moves will affect future situations.

When we use the word "map," we refer to any map, board, or playing field which appears graphically as part of the game. Chess, once more, is the perfect example. The chessboard (map) is indispensable to the playing of the game, but the strategy involves the relationships between the pieces on that board as defined by the rules and style of play.

Types of Strategy and Map Games

What are the types of strategy and map games? We have already mentioned battle simulations and football. Another type which is popular may be called the "mock economy" game. In the mock economy game, the player is given a sum of money to begin. As the game proceeds, the player must decide how to spend the money. One such game involves a wagon train heading toward California in the days of the old west. The player must buy supplies, armaments, etc. Problems surface along the way, such as Indian attacks, plagues, and hunger. If the player spends the money wisely, the wagon train reaches its goal.

An interesting example of the mock economy is a game involving a sidewalk lemonade stand. The stand owner must allocate money for supplies, advertising, and production of lemonade. Each day is different: Some are hot and thirst producing; others are cold. The challenge, as well as the fun, is that the business decisions must be made before the day's weather conditions are known.

Planning Your Simulation

Whatever the subject of the strategy/map game, the designer must answer two questions at the outset: (1) What are the essential characteristics of the real situation to be simulated? and (2) How

can those characteristics be simulated effectively on your Commodore 64?

We want to take a closer look at these two points. What are the essential characteristics of the real situation you want to simulate? This forces us to examine our target reality in great detail. Any activity or reality state is comprised of countless elements. Not all of these elements are crucial. An example?

One of the games in this section, *Goal to Go!*, is a football game. A football game includes the playing field, two teams, coaches, officials, a football, goal posts, injuries, a scoreboard, rules, a stadium, fans, marching bands, announcers, television crews, reporters, groundskeepers, parking lot attendants, hotdog stands, health inspectors. . . . You throw up your hands? Have we gone into too much detail? Now you begin to appreciate the necessity for thoughtful evaluation of which elements of a real situation are essential and, perhaps more importantly, which are not.

Hot dog stands are not essential you say. Very well. What about a scoreboard? Of course! Okay, what should appear on the display? The score, the time remaining, the quarter, who has the ball, how many yards to go for a first down, the down, the total first downs, total passes attempted, total passes completed, passes intercepted, total yards gained rushing, cartoons of players crossing the goal line after a touchdown. . . ? You throw your hands into the air again! But some scoreboards have that, and more.

Okay, the scoreboard should at least have the score, the quarter, the down, yards to go for a first down, and. . . . Well, we are not going to go down this road any farther for the moment. Ultimately, you will have to decide which of the many elements of your target reality can be dropped since there will always be too much to include in your program. For example, quick kicks are a part of football, but you may decide that, in the interest of practicality, this element of the game can be left out of your Commodore 64 version of reality.

Your image of reality can never be a totally accurate mirror of reality. It will always be a compromise. It is the game designer's art to make that compromise as close to the real thing as possible.

Four New Techniques

To help you create strategy and map simulations, here are some techniques which may be of help.

Keeping Track of Players' Positions on a Map

Many games include some sort of playing field or geographical area. In the case of football, position is marked in a linear fashion. Although a football field has width as well as length, for a computer simulation it can be thought of as a line. A variable is selected to represent the current line of scrimmage. Let's call it Y . Since a football field is 100 yards long, Y can have a value of 1 to 99 (0 and 100 would put the line of scrimmage in the end zone!).

Y has a current value, perhaps 20. If the result of the play is a gain of 3 yards, the new value of Y is Y plus 3, or 23. If the next play is a loss of 8 yards, Y becomes Y plus -8 , or 15.

"But which side of the 50 yard line are you on?" you ask. If a team has the ball on its own 20, a gain of 3 makes Y equal 23. On the other hand, if a team has the ball on the opponent's 20, a gain of 3 takes the ball to the 17, not the 23! Quite true.

In this case, it was necessary to create two variables. To mark the position of the ball in absolute terms, the football field is considered a line marked 1-99. Team one defends the goal line at yard 0, while team two defends the goal line at yard 100. (Incidentally, one concession to practicality was made in this regard. In *Goal to Go!*, the teams never change sides. This could have been provided for, and you may prefer to do so yourself. A computer football game does not suffer from the physical conditions of a real game, such as wind and sun, so we felt that this omission was a minor sacrifice.) If team one gains, their gain is added to Y . If they lose, their loss is subtracted from Y . The reverse is the case if team two has the ball. In this fashion the line of scrimmage can always be calculated, no matter which side of the 50 yard line a team is on.

However, since telling the player that his team has the ball on the 73 yard line would damage the game's credibility, a second variable was devised. This variable, DY, stands for Display Yard. Here is where a fundamental test of logic comes in handy. Remember, variable Y locates the line of scrimmage on a line 1-99. If Y is equal to or less than 50, DY equals Y. If either team has the ball on the 43 yard line using the 1-99 scale, then the Display Yard is also 43.

But if either team's position on the 1-99 rule exceeds 50, the following algorithm determines the value of DY:

IF Y>50 THEN DY=100 -Y

If either team has the ball on the 70 yard line (using the 1-99 scale represented by variable Y), then DY is calculated by subtracting the value of Y (70) from 100. That gives us a value of 30 for DY. This is correct, since if the ball is on the 70 yard line (using a 100-yard scale) it is also on the 30 yard line of the other half of the field.

Other variables must be created to represent such factors as first-down yard lines, distance from the goal line, current down, and so on. Most of these will use the Y value as a reference point.

Keeping track of player position on a two-dimensional map is more complicated. Two variables are needed to indicate the coordinates of a player's current position. The process of changing positions is similar to the method just outlined. If the map in question is a 100-by-100-space grid, the current position might be 30,20. Assume that the point of origin is in the upper left-hand corner (it certainly could be set up differently). The 30,20 position would indicate a spot 30 spaces to the right of the upper left-hand corner and 20 spaces directly below that.

The variables might be called H (for the horizontal coordinate) and V (for the vertical). In our example, then, H equals 30 and V equals 20. If the results of play indicate a move of two to the right and three down, the change would be calculated thus:

H=H+2:V=V+3

H now equals 32 and V equals 23. The player's position is therefore at point 32,23. Your program now calls a subroutine which would graphically display the symbol (football, helmet, etc.) at the new coordinate. Study the following program which we include as an example of such a subroutine:

```
1 REM *****
2 REM * PLOT COURSE OF *
3 REM * ROCKET ON *
4 REM * IMAGINARY GRID *
5 REM * UP IS NEG. MOVE; *
6 REM * DOWN IS POS MOVE; *
7 REM * LEFT IS NEG. *
8 REM * RIGHT IS POSITIVE *
9 REM *****
10 PRINT "{CLR}{DOWN 4}THIS PROGRAM ALLOW
S YOU TO MOVE"
11 PRINT "{DOWN}AN OBJECT ON A MAP, AS IN
A BOARD GAME."
12 PRINT "{DOWN}TO MOVE THE OBJECT UP, IN
PUT A NEGATIVE"
13 PRINT "{DOWN}NUMBER. TO MOVE DOWN, A P
OSITIVE NUMBER."
14 PRINT "{DOWN}TO MOVE RIGHT, INPUT A PO
SITIVE NUMBER."
15 PRINT "{DOWN}TO MOVE LEFT, INPUT A NEG
ATIVE NUMBER."
16 FOR I=1 TO 6000: NEXT
100 REM **** VERSION 9 ****
105 REM ***** PAT'S COORDINATES *****
108 :
109 :
110 POKE53281,1:POKE53280,1:PRINT "{BLK}"
130 MX=1:MY=100
140 X=1:Y=100:L=1825:C=L+54272
150 GOSUB 450
160 POKEL,90:POKE56257,0
170 V=22:H=3
180 GOTO 210
190 POKE781,V:POKE782,H:POKE783,0:SYS655
```

```

20
200 RETURN
210 :
220 V=22:H=2:GOSUB190:PRINT"
      "
221 V=23:H=2:GOSUB190:PRINT"
      "
222 V=22:H=2:GOSUB190:INPUT"=INPUT VERTI
CAL MOVE=";YY
223 V=23:H=2:GOSUB190:INPUT"=INPUT HORIZ
ONTAL MOVE=";XX
230 IF YY=999 THEN END
240 NY=YY+Y :NX=XX+X
250 IF NY=Y AND NX=X THEN 420
260 IF NX>0 AND NX<101 AND NY>0 AND NY<1
01 THEN 290
270 YY=Y:XX=X
280 V=22:H=2:GOSUB190:PRINT"INVALID MOVE
, ENTER AGAIN{ROFF}      "
285 FORI=1TO3000:NEXT:GOTO220
290 IF NX=X THEN NL=L:GOTO350
300 IF NX>MX+4 AND NX<MX+10THENNL=L+1:MX
=MX+5:GOTO350
310 IF NX>MX+9 THENNL=L+2:MX=MX+10:GOTO3
50
320 IF NX<MX AND NX>MX-6THEN NL=NL-1:MX=
MX-5:GOTO350
330 IF NX<MX-5 THEN NL=NL-2:MX=MX-10:GOT
O350
340 NL=L
350 IF NX>X+10 THEN NX=X+10:X=NX:GOTO370
355 IFNX<X-10 THEN NX=X-10::X=NX:GOTO 37
0
360 X=NX
370 IF NY>MY+4 AND NY<MX+10 THEN NL=NL+4
0:MY=MY+5
380 IF NY>MY+9 THENNL=NL+80:MY=MY+10
390 IF NY<MY AND NY>MY-6 THEN NL=NL-40:M
Y=MY-5
400 IF NY<MY-5 THENNL=NL-80:MY=MY-10
410 IF NY>Y+10 THEN Y=Y+10:Y=NY:GOTO420

```

```

415 IF NY<Y-10 THEN Y=Y-10
418 Y=NY
420 GOSUB450
430 POKEL,32:L=NL:POKEL,90:POKEL+C,0
440 GOTO 210
450 PRINT "{CLR}{RIGHT}{#185 20}"
460 FORI=1TO20:PRINT "{#182}
      {#161}"
470 NEXT I
480 PRINT" {#184 20}
490 RETURN
10000 V=22:Y=1:GOSUB190
10010 INPUT"ENTER HERE";A$

```

Representing Units of Time

Strategy and map simulations frequently include a time element. This can be measured in minutes, hours, years, etc. If you want to simulate a long war, the unit of measurement could be one month. If you want to simulate a nation's economy, the unit of time might be a year. Whatever unit of time you choose, it must make sense in the context of the game.

You will want a variable to keep track of the passage of time. For this example, let's say you are simulating the performance of an automobile company and want to simulate yearly sales. YR will be the variable. Your game will have a main routine which will call all subroutines. Each time the program flow goes through this main routine, you will increment the YR variable. Here is a ridiculously simple program illustrating the point:

```

100 REM *****
110 REM * CAR SALES *
120 REM *****
130 A$="CARS SOLD THIS YEAR"
140 B$="TOTAL SOLD IN 10 YEARS"
160 SALES=INT(RND(1)*500000)
170 PRINT A$;SALES
175 TSALES=TSALES + SALES
180 YR=YR+1

```

```

190 IF YR=10 THEN GOTO 200
195 GOTO 160
200 PRINT B$;TSALES
210 END

```

Line 160 determines the car sales for the current year. Line 170 displays that sales figure. Line 175 adds the year's sales to the total sales for all years. Line 180 adds 1 to the time variable YR. Line 190 checks to see if YR has reached 10. When it has, the final sales total is displayed, and the program ends.

Time could be measured exactly in reverse. A player could have 10 time units (days, months, whatever) to accomplish a task. Each cycle through the main program routine equals 1 time unit. In each cycle, the time variable would be decremented (decreased) by 1. When the variable reaches 0, the time limit has been reached, and the game ends.

Tasks within Time Limits

In general, the art of creating simulations involves creating variables which represent the essential elements of the reality you are trying to simulate. The interaction of these variables is what determines the outcome of the game.

In this simulation technique, we present a program in which variables interact to produce results based on two factors: increasing money and limited time. Players must double their money every five turns to stay in the game. The following routine suggests a way to handle this:

```

10  REM *****
11  REM * TIME LIMIT TASK DEMO *
12  REM * -THIS PROGRAM SHOWS *
13  REM * HOW TO SET UP A PRO- *
14  REM * GRAM WHICH REQUIRES *
15  REM * THAT A CONDITION BE *
16  REM *      MET WITHIN A      *
17  REM *      TIME LIMIT        *
18  REM *****

```

```

20  :
25  PRINT "{CLR}"
60  PRINT "YOU ARE A GAMBLER IN A HIGH
STAKES GAMBLING GAME."
70  PRINT "IF YOU QUINTUPLE YOUR MONEY
EVERY FIVE"
80  PRINT "ROUNDS YOU MAY CONTINUE."
90  FOR I=1 TO 4000:NEXT
100 M=100:M1=100:M2=100:T2=1:R=100
110 WINNINGS=INT(RND(1)*(2*R))
120 M2=M2+ WINNINGS
125 PRINT "{CLR}YOU HAVE WON "WINNINGS"
DOLLARS."
127 PRINT "{DOWN 1}{RIGHT 3} YOUR NEW
TOTAL IS "M2" DOLLARS."
130 T1=T1+1
140 IF T1=5 THEN IF M2<M1*5 THEN GOTO 200
150 IF T1=5 THEN T1=0:M1=M2:T2=T2+1:R=M2
170 FOR I=1TO 500:NEXT: REM -PAUSE LOOP-
180 GOTO 110
200 FOR I=1 TO 500:NEXT:PRINT"{CLR}{DOWN 4
{RIGHT 5}SORRY, YOU HAVE BEEN ELIMINATED."
210 PRINT "{DOWN 1}{RIGHT 1} YOU MADE "
(M2-M)" DOLLARS AND LASTED "T2 "ROUNDS."

```

The variables are initialized in line 100. M is the player's beginning amount of money. M1 is the starting amount for each round. M2 is the running total of the player's funds. In 110 WINNINGS, the amount won in the current hand is determined by a random number multiplied by twice R; R is set to twice the amount on hand at the beginning of each round (each round has five hands).

In line 120, the player's current total is added to the amount just won. Line 130 increments T1, which stands for the number of hands played in the current round. Line 140 and 150 determine if the round is over. If T1 is less than 5 (five hands per round), the round continues. When T1 reaches 5, line 140 checks to see if the second condition has been met. Line 140 applies the condition that if the

round is over, you can proceed only if your current sum of money is five times more than it was at the beginning of the round. If the player has not won enough, the program branches to 200, then ends.

If the player has earned enough money, then line 150 comes into play. T1 is reset to 0, which starts the next round. M1 is made equal to M2, which sets the starting amount for the next round equal to the total on hand at the end of the previous round. T2, the total of rounds, is incremented (this happens only one-fifth as often as T1 is incremented). Finally, R is made equal to twice the total cash on hand. Otherwise the random numbers produced in line 100 would not, on average, be great enough to allow the player a chance to continue.

Dealing Cards

Card games are always popular. Anyone who has played bridge or poker realizes how much strategy is involved in these games. We include this program not only to help you design card games, though. The technique demonstrated in the following routine can be useful in any program which requires random selections from a list and also requires that each selection only be chosen once.

```
0 REM *****
1 REM *      CARD DEALING PROGRAM      *
2 REM *****
5 GOTO 100
100 :
150 REM *****
155 REM *          C$=CARD NAMES          *
160 REM *          C=CARD NUMBERS        *
165 REM *          D=CARDS DEALT        *
170 REM *****
180 DIM C$(52),D(52)
190 FOR I=1 TO 52:READ C$:C$(I)=C$:NEXT
200 DATA 2 OF CLUBS,3 OF CLUBS,4 OF CLUB
S,5 OF CLUBS,6 OF CLUBS,7 OF CLUBS
210 DATA 8 OF CLUBS,9 OF CLUBS,10 OF CLU
BS,JACK OF CLUBS,QUEEN OF CLUBS
```

```

220 DATA KING OF CLUBS,ACE OF CLUBS,2 OF
  HEARTS,3 OF HEARTS,4 OF HEARTS
230 DATA 5 OF HEARTS,6 OF HEARTS,7 OF HE
ARTS,8 OF HEARTS
240 DATA 9 OF HEARTS,10 OF HEARTS,JACK O
F HEARTS,QUEEN OF HEARTS
250 DATA KING OF HEARTS,ACE OF HEARTS,2
OF DIAMONDS,3 OF DIAMONDS
260 DATA 4 OF DIAMONDS,5 OF DIAMONDS,6 O
F DIAMONDS,7 OF DIAMONDS
270 DATA 8 OF DIAMONDS,9 OF DIAMONDS,10
OF DIAMONDS,JACK OF DIAMONDS
280 DATA QUEEN OF DIAMONDS,KING OF DIAMO
NDS,ACE OF DIAMONDS
290 DATA 2 OF SPADES,3 OF SPADES,4 OF SP
ADES,5 OF SPADES,6 OF SPADES
300 DATA 7 OF SPADES,8 OF SPADES,9 OF SP
ADES,10 OF SPADES
310 DATA JACK OF SPADES,QUEEN OF SPADES,
KING OF SPADES,ACE OF SPADES
330 REM *****
340 REM *   CARD DEALING ROUTINE   *
350 REM *   DL=NUMBER PICKED     *
355 REM * D()=CARDS ALREADY PICKED *
356 REM *   S=CARD OK? FLAG      *
360 REM *****
370 REM
380 FORI=1TO52
390 S=0:DL=INT(RND(1)*52+1)
400 FORJ=1TO52:IF DL=D(J) THEN S=1
410 NEXT
420 IF S=1 GOTO390
430 D(I)=DL
440 PRINT"          "C$(DL)
450 NEXT I
460 PRINT"ALL CARDS DEALT"

```

The trick here is to create two arrays. One array, C\$(52), contains the names of the cards. The other, D(52), will hold the numbers of the cards which are drawn. This is done by means of two loops,

one nested within the other. The main loop begins in line 380, ending at 450. This loop is performed 52 times, one loop for each card in a deck (no joker here!). In line 390 a random number is selected that will stand for the next card picked. The complication is that, without a screening routine, the same number (card) could come up again.

The second loop, beginning at line 400, compares the latest number picked (DL) with all the previous numbers selected. If a match is found, flag S is set to 1. If flag S is set, line 420 will detect it and send the program back to select another number in line 390. If no match has been found, the new number (DL) is added as the next element of array D(52). If that number is picked again, the J loop will spot the match and set flag S once more.

Valid numbers are converted into the name of the card and displayed for the players in line 440. If the number five is selected, the fifth element of array C\$(52) will be displayed. That element is the fifth card in the deck (according to our DATA statement roster), the six of clubs. You will notice that the program slows as the end of the deck is neared. This is because it takes more random number selections to find one which has not already been picked as array D(52) fills up.

We have presented but a few of many techniques which may help you design simulation games. Once you have mastered the fundamentals of BASIC, you will see more and more ways to transfer what you see in real life (or in your imagination) to your Commodore 64.

Chapter 29

ROCKET TO THE GREEN PLANET

Rocket to the Green Planet simulates a spaceflight. A spaceflight simulation should process a player's decisions and move the spaceship according to rules established at the beginning of the game. For example, the player must be told what the ship's speed will be if the main rockets are fired, as opposed to the hyperspace speed. The simulation usually requires the player to decide in what direction to move, including correcting the ship's direction if some influence takes the ship off course. The aim of a spaceflight game should be to create a challenge for the player, making skill count and keeping the goal within reach. Winning should be a fairly strong possibility, but not so strong that there is no challenge.

In *Rocket to the Green Planet*, the player is left alone in a rocket which is headed for the green planet, Esmeralda. The problem is to continue on a trajectory from X1,Y1 to X100,Y100, despite aliens, meteorites, cabin pressure problems, and a finite fuel supply. The player is able to (1) fire the main rockets, (2) go into hyperspace, or (3) coast. It is also possible to correct the X or Y axis to compensate for factors which carry the ship off course. The player learns that hyperspace jumps buy more distance for the fuel expended but may also take the ship unpredictably off course. The player must decide when to use which means of propulsion and when and how much to correct trajectory.

Defining Variables

As in the previous games, we must develop a game road map. First, we will list all of the major variables we know we need, and if necessary, describe each of these variables. To begin, the

spacecraft must have a propulsion system, so we have rockets to be fired. Then we must have a starting location and a destination. We will call these variables X and Y, and they will change value as we progress toward the destination. The green planet must have a location (or we will have no way of finding it), so we have another variable, DIstance. As we move along, we will use FUel. FU must be decremented as the rocket engines consume fuel.

Another important variable which we might overlook at first is TiMe (TM). To simulate an actual flight, we must keep track of how long it takes to reach the green planet. The game could impose a time limit on the player, but whether it does or not, it is necessary to record the passage of time to have a realistic simulation. If there are to be random events introduced to take the ship off course and create interesting complications, we will need at least one variable for selection of random numbers. Since we will move at some variable rate, we need a variable for speed or VELOCITY to be incremented as the rockets are fired. These are the major variables we will use in *Rocket to the Green Planet*, but you probably realize by now that new variables have a way of materializing as the program unfolds and as you discover new needs. For example, if there are to be any sound effects, we will need variables for the beginning of the sound chip memory location, the attack, decay, sustain and release, etc. Also, we might want to move our sprite planet around on the screen, depending upon the proximity of the rocket to the planet. This also requires variable definition.

Getting Started

Line 1 consists of the familiar POKES which set the screen and border color and clear the screen. Line 10 sets the variable values to 0. Line 12 sets the values for variables FUel (FU) and DIstance (DI) at their beginning levels. FU is set at 25 (units), but could be set at any level. DI is set at 100 (units), but could also be set at any level. FU and DI units are relative and have no absolute meaning. FUel might be 25 tons and DIstance might be 100 light-years 'or 100 million miles.

Lines 14-35 contain the game's instructions. These would be varied if the rules are changed by modifications to the program. The idea is to give enough information so the player can make intelligent

game decisions. If you change the position of the spaceship or the planet, you should change the instructions accordingly.

Line 36 displays the prompt in reverse mode and instructs the system to wait for the RETURN key to be pressed, indicating that the player has finished reading the instructions. Line 40 calls subroutine 1100, which reads in a sprite-defining routine, including a DATA statement.

A Subroutine

Subroutine 1100 sets the values for one sprite (sprite one), and the DATA statements beginning on line 1300 define the shape of the sprite, which is the green planet. Refer to Chapter 21, your *Commodore 64 Programmer's Reference Guide*, and your *Commodore 64 User's Guide* for specific instructions regarding the use of DATA statements to define the sprite shapes. Line 1105 sets the beginning memory location for the first sprite. Line 1110 POKES the value for the first sprite at VV+21,1. This enables the sprite by activating the sprite-enable register. Line 1120 sets the planet color at cyan (blue-green) and positions the planet at its vertical and horizontal screen locations.

Line 1130 instructs the system to READ 63 DATA values (0 to 62) and POKE 832+I (memory location where the sprite begins). Line 1160 POKES the sprite data into location 2040,13 (the thirteenth area of memory). The 63 DATA statements (lines 1300-1360) take 63 bytes of memory. After READING the DATA in subroutine 1100, the program RETURNS and line 47 is read.

RETURNING to the Main Routine

Line 47 defines the time variable, setting up a loop so that elapsed time is incremented each time the program goes through the loop. Lines 49-65 comprise the routine to create stars on the screen as background for our ship. Both this routine and the one following, which creates the ship, use keyboard graphics.

Line 69 colors the ship yellow. Lines 69-76 draw the outline of the ship. Line 77 sets the cursor color back to white, so that the text is not yellow. Lines 100-150 create the video readout, which tells the player the elapsed time, the amount of fuel left, the X and Y coordinate location, the velocity, the trajectory, and the distance from the planet.

In line 200, N1 and N2 select random numbers from 1 to 10. In line 210, "ALERT" is PRINTED if N1 selects 1, 2, or 3. Thus, there is a 30 percent chance of having an alert. Line 220 calls subroutine 230, 240 or 250, depending upon the value of N1. Subroutine 230 (CABIN DEPRESSURIZING) causes both X and Y to be decremented, forcing the ship backward. Subroutine 240 (ALIEN ATTACK) increments Y by the value of N2 (the random number). Subroutine 250 (METEORITES) increments X by the value of N2. Due to the randomness of these numbers, the game is constantly varied.

Player Input

If N1 selects a number equal or greater than 4, the program will proceed to line 290, where the player is invited to input the command (M=MAIN ROCKETS, H=HYPERSPACE, C=COAST). Lines 310-330 define what happens when the player decides which option to take. M increases velocity by one unit, decreases fuel by half a unit times the velocity, calls subroutine 1020, then continues. H increases velocity by a random number, decreases fuel according to current velocity, and calls subroutine 1000 before continuing. C decreases velocity but does not decrease fuel. (It is true that in free-fall space, an object would continue indefinitely at a constant velocity; however, in our game energy is drawn to maintain the life-support systems, and this decreases the ship's velocity even if the rockets are not fired.) Line 335 is an error trap which prevents the player from continuing on course when there is no velocity or from getting a readout of negative velocity.

In the event that the player decides to coast, line 340 bypasses the correction routine. Since coasting gets no burst of speed from the rockets, there is no way to change the trajectory through the correction procedure. Lines 350-390 constitute the correction

routine. The player is invited to (1) decrement X, (2) decrement Y, (3) increment X, (4) increment Y, or (5) make no correction. After the player inputs CO, subroutines 365, 370, 380 and 390 are called. Each of these use N2, the random number, to increment or decrement the positions of X and Y.

Lines 450 and 452 set the new X and Y positions of the ship, based upon the velocity (provided that there is velocity). Lines 455-620 complete the time loop. Line 455 creates a new variable (D-DI), and line 470 changes the distance depending upon the positions of X and Y. Lines 550 and 560 call subroutine 1400 or 1500, which brings the sprite planet closer or pushes it farther away, depending upon the ship's distance from the planet. Subroutine 1400 enlarges the sprite and brings it closer to the ship, while subroutine 1500 moves the planet away from the ship by POKING the screen memory locations to reposition the sprite. Line 600 ENDS the game when the ship runs out of fuel, and line 610 ENDS the game when the ship arrives at the planet. The player is shown the elapsed time (line 630), the rating (lines 640-650), and is invited to play again (lines 660-680).

Sound Effects and Pictures

One of the most important enhancements in a spaceship flight program for the Commodore 64 would be a sound effect, like a rocket firing. After you play the game, I think you will agree that this sound is probably at least as effective as the picture of the rocket and the sprite planet. Of course, the sound will only be effective if it's heard while the colors flash to simulate the sight and sound of a rocket firing. Subroutines 1000 and 1020 comprise both the sound and graphic simulations of the rocket being fired.

Remember, if the player chooses to go into hyperspace (line 310), subroutine 1000 is called. Subroutine 1000 also includes subroutine 1020, so that the hyperspace command is really subroutine 1000 plus subroutine 1020. The screen and border POKES make the screen flash orange, yellow, black and orange, in turn. The pause loop allows just enough time for the player's visual sense to comprehend the flash but not enough to distinguish the individual colors. The POKES in subroutine 1020 cause the border of the screen

to flash yellow, then black. The main rockets do not make as big a flash as the one which occurs when going into hyperspace. Lines 1030-1080 comprise the sound creation and RETURN.

Lines 1030-1070 create a noise that is similar to the sound made by a spaceship while taking off. Line 1030 sets the variable values of the sound chip memory location, waveform location, and ADSR envelope. Line 1040 indicates maximum volume, the noise waveform, and low attack. Lines 1050 and 1060 set up loops to vary the duration of high and low frequencies. Line 1070 turns off the first voice control register.

Summary

There are many features in *Rocket to the Green Planet* with which to experiment. You can change the set values of any of the variables, such as distance, fuel level, or time loop. You can change the method of computing velocity, range of the random numbers to be selected, means of moving the ship (recalculating X and Y), screen and border colors of the ship, space or color of rockets when fired, or even the rules of the game.

This is one of the shortest programs presented in the book, but it is also one of the most varied and interesting games, with great potential for reconstruction. This program also provides a basis for learning sound programming techniques, for although short, it calls many subroutines and has nested loops. It also explores and incorporates both sound and graphics, and uses two types of graphics.

There is a lot packed into *Rocket to the Green Planet*, and a lot more could be done to enhance it.

Rocket to the Green Planet

```
2 PRINT "{CLR}": POKE53281,0:POKE53280,0:  
PRINTCHR$(5)  
10 X=0:Y=0:VE=0
```

```

12 FU=25:DI=100:TR=X-Y:SP=0
14 PRINT"YOU ARE IN A SPACE SHIP AT STAR
MAP"
15 PRINT"SECTOR X 1, Y 1; HEADED FOR THE
GREEN"
16 PRINT"PLANET, ESMERALDA, WHICH IS AT
LOCATION"
17 PRINT"X 100, Y 100. YOU MAY BE FACED
WITH A"
18 PRINT"NUMBER OF ADVERSE INCIDENTS WHI
CH WILL"
19 PRINT"CARRY YOU OFF COURSE. YOU MUST
DECIDE"
20 PRINT"WHETHER TO FIRE THE MAIN ROCKET
S, GO"
21 PRINT"INTO HYPERSPACE, OR COAST. HYP
ERSPACE"
22 PRINT"USES AN UNPREDICTABLE FUEL AMOU
NT"
23 PRINT"AND TAKES YOU TO AN UNPREDICTAB
LE"
24 PRINT"LOCATION. THE MAIN ROCKETS USE
1 UNIT"
25 PRINT"OF FUEL AND COASTING USES NO FU
EL."
26 PRINT"AFTER FIRING ROCKETS, YOU ARE A
SKED"
27 PRINT"WHAT CORRECTION FACTOR (- OR +)
YOU"
28 PRINT"NEED TO GET BACK ON COURSE. IF
YOU ARE"
30 PRINT"NOT OFF COURSE YOU WILL CHOOSE
'5=NONE'"
32 PRINT"YOU WILL BE INFORMED OF ELAPSED
TIME"
34 PRINT"AND FUEL LEVELS AS WELL AS PROB
LEMS"
35 PRINT"ENCOUNTERED."
36 INPUT"{RVS}PRESS RETURN TO CONTINUE.(
ROFF)";RP$
40 GOSUB 1100

```

```

47 FOR TM=0 TO 60 STEP.05
49 PRINT"{CLR}"
50 FOR I=1 TO N:PRINT"  {#209}           ":NE
XT
52 FOR BA=1 TO  N
54 FOR TZ=1 TO N
56 NEXT TZ
58 NEXT BA
60 FOR BA=30 TO 1 STEP-1.5
61 PRINT"                               {#209}  ";
62 FOR BA =1 TO 30 STEP 5
63 PRINT"                               *           ";
65 FOR TZ=1 TO 30: NEXT TZ:NEXT BA
70 PRINT"                               00000000  "
71 PRINT"                               00000000  "
72 PRINT"                               00000000  "
73 PRINT"                               00000000  "
74 PRINT"                               00000000  "
75 PRINT"                               00000000  "
76 PRINT"                               00000000  "
77 PRINT"{WHT}"
100 PRINT"{RVS}           STATUS REPORT:"
110 PRINTTM;"TIME ELAPSED ";FU;" LITERS
LEFT"
120 PRINT  "LOCATION: ";X;"X";" ";Y;"Y":
PRINT" VELOCITY: ";VE
125 TR=X-Y
130 PRINTTR;"ON/OFF COURSE           DISTANCE:
";DI
150 PRINT"{ROFF}"
200 N1=INT(RND(1)*10+1):N2=INT(RND(1)*10
+1)
210 IF N1<4 THEN PRINT"{RVS} ALERT:{ROFF
}"
220 ON N1GOSUB 230,240,250:GOTO 290
230 PRINT"CABIN DEPRESSURIZING ":X=X-1:Y
=Y-1:RETURN
240 PRINT"ALIEN ATTACK ":Y=Y+N2:RETURN
250 PRINT"METEORITES ":X=X+N2:RETURN
290 PRINT"{RVS}           COMMAND:{ROFF}"
300 INPUT"(M=MAIN ROCKETS H=HYPERSPACE C

```

```

=COAST)";C$
310 IF C$="H" THEN VE=VE+INT(RND(1)*5+1)
:FU=FU-.5*VE: GOSUB1000:GOTO 350
320 IF C$="M" THEN VE=VE+ 1:FU=FU-.5*VE:
GOSUB1020:GOTO350
330 IF C$="C" THEN VE=VE-.5
335 IF VE<=0 THEN PRINT"{RVS}YOU HAVE LO
ST VELOCITY. FIRE ROCKETS!{ROFF}":GOTO
300
340 GOTO 450
350 PRINT"CORRECTION? (1=-X,2=-Y,3=+X,4=
+Y,5=NONE)"
352 INPUT CO
360 ON CO GOSUB 365,370,380,390:GOTO 450
365 X=X-N2 :RETURN
370 Y=Y-N2 :RETURN
380 X=X+N2 :RETURN
390 Y=Y+N2 :RETURN
450 IF VE>0 THEN X=X+.5*VE
452 IF VE>0 THEN Y=Y+.5*VE
455 D=DI
470 DI=((100-X)+(100-Y))/2
550 IF D<=DI THEN GOSUB 1400
560 IF D>DI THEN GOSUB 1500
600 IF FU<0 THEN PRINT"OUT OF FUEL":GOTO
660
610 IF DI<1 THEN PRINT"ARRIVED":GOTO 630
620 NEXT TM
630 PRINT"THE TRIP TOOK ";TM; " HOURS"
640 R=200*TM
650 PRINT"YOUR RATING IS ";R; "."
660 INPUT"PLAY AGAIN ";Y$
670 IF Y$="Y" THEN RUN
680 END
1000 POKE53281,8:POKE53280,7:POKE53281,0
:POKE53281,7
1010 FOR PAUSE =1 TO 15:NEXT PAUSE
1020 POKE 53280,7:POKE53280,0:POKE53281,
0
1030 V=54296:WW=54276:AA=54277:HF=54273:
LF=54272

```

```

1040 POKEV,15:POKEWW,129:POKEAA,15
1050 FOR X=200 TO 5 STEP-2:POKEHF,20:POK
ELF,X:NEXT
1060 FOR X=150 TO 5 STEP-2:POKEHF,10:POK
ELF,X:NEXT
1070 POKE WW,0
1080 RETURN
1100 REM SPRITE METEORITE
1102 RESTORE
1105 VV=53248
1110 POKEVV+21,1
1120 POKEVV+39,3:POKE VV+0,229 :POKEVV+1
,70
1130 FORI =0 TO 62:READ Q1:POKE832+I,Q1:
NEXT
1160 POKE 2040,13
1300 DATA 0,255,0 ,7,255,224,15,255,240
1310 DATA 31,255,248,63,255,252,127,255,
254
1320 DATA 127,255,254,255,255,255,255,25
5,255
1330 DATA 255,255,255,255,255,255,255,25
5,255
1340 DATA 255,255,255,255,255,255,127,25
5,254
1350 DATA 127,255,254,63,255,252,31,255,
248
1360 DATA 15,255,240,7,255,224,0,255,0
1370 RETURN
1400 POKEVV+23,1 :POKEVV+29,1
1410 POKEVV+39,3:POKE VV+0,229 :POKEVV+1
,70
1420 RETURN
1500 POKEVV+39,3:POKE VV+0,229 :POKEVV+1
,20
1510 RETURN

```

The following lines show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustration in *Rocket to the Green Planet*.

```

70 PRINT"          {#205}{#163 6}{#205}
   {WHT} * {YEL} "
71 PRINT" {#192 4}{#176}{#207}{#183 17}
{#205} "
72 PRINT" {#163 5}{#173}{#180}          {#
215 7} {#205}{WHT} *{YEL}"
73 PRINT" {#183 3}{#176}{#180}
   {#206} "
74 PRINT" {#163 5}{#173}{#204}{#175 17}{
#206} "
75 PRINT"          {#206}{#164 6}{#206}
   "
76 PRINT"          {WHT} *          .
. "

```


Chapter 30

A DAY AT THE RACES

Computer games on the Commodore 64 give us the chance to participate, in a make-believe way, in many activities that would be too dangerous or expensive to undertake in real life. Some computer simulations create fantasy worlds (space adventures, haunted mansion role playing, etc.). Not all simulations depend on the fantastic, however.

Many computer simulations provide the participant with a “dress rehearsal” for important real life situations. Some involve physical equipment controlled by computers. Pilot flight simulators are a good example. The United States Air Force uses extremely detailed computer graphics to enhance its flight simulators. One such simulator allows practicing pilots to look out of their cockpits and see other planes, clouds, and even detailed farmlands below.

Other simulations involve complex social activities, such as national economies. Computer stock market games are really economic simulations. The player has the fun of playing the stock market without the risk of losing a thing (or the chance of getting rich!). Needless to say, battle simulations are much safer than real wars.

In *A Day at the Races*, we present a horse race betting simulation which does not require you to sell your house to step up to the betting window. The prime goal of this game is to give the player at least a little bit of the feel and excitement of going to the track. Pari-mutuel betting is a much more complex system than what we use in *A Day at the Races*; however, the essential elements of chance combined with strategy and experience are present.

Simplifying Your Game

A number of simplifications were made to keep the game within the practical limits of this book. For one, there are only five races in our version of a day at the track. For another, only five horses run in each race. This is not totally unrealistic, since in some races the field of horses is quite small. But the biggest difference between this game and the real thing is that in *A Day at the Races* you can only bet on one horse per race, and only to win. At a real track, bettors can wager on any number of horses in a given race. They can also bet on a horse to place (finish second) or show (finish third) as well as to win. But the most important aspect of race track betting has been preserved: If you bet on losers, you end up with no money!

In each race, one of the five horses is the favorite. Favorite here means the horse with the shortest odds. If a horse has 3-2 odds, the bettor must put up two dollars to win three. On the other hand, a horse with 5-1 odds offers a potential gain of five times the potential loss. Of the two, the first would be the favorite.

We have arbitrarily assigned the favorite horse a 60 percent chance of winning. The other four horses all have an equal chance of winning, even though they have different odds. Later we will discuss ways of modifying the program to give more variety, and hence realism, in the relative chances of all horses. Also, an animation module has been added as a visual enhancement. This sequence could, with further development, be made a more central part of the game.

The Road Map

Here is the road map for *A Day at the Races*:

1. The player is given an amount of money at the start of the day's races.
2. Before each of the five races, the the names of the horses and their betting odds are announced.

3. The player chooses a horse.
4. The player then indicates how much money will be bet on that horse.
5. The race is run (animation module).
6. The winner is announced.
7. If the player's horse is the winner, the winnings for that race are announced and added to the total. Otherwise, the amount bet is subtracted from the total.
8. At the end of the five races, the player's ending total in dollars is announced.

Setup Modules

First, let's cover all the setup modules. Lines 2000-5000, nearly half the program, consist of one-time routines. The animation module utilizes custom characters, and these are set up in lines 2000-3000. The Alternate Character Set routine copies the data for character set one into the memory area beginning at 32768, which is the beginning of memory bank two. The screen memory begins at 34816. The data for the custom characters are POKED into memory in lines 2310-2330. The characters to be modified are CHR\$(186) through CHR\$(189).

The initial variables and arrays are identified in REMARK statements beginning at 4004. The arrays and their functions are ODD\$(50), which contains the betting odds to be displayed on the screen; ODDS(50), which contains the betting odds in decimal fraction form, used to determine the amount won; FAV(10), which identifies the favorite horse in each race; SN(10), which identifies the start of each set of odds; R\$(5), which names the race (first, second, etc.); and H\$(25), which names all 25 horses.

The Main Game Routine

The sections of the main game routine follow, with explanations.

Lines 70-75 contain the pause subroutine. Lines 130-139 draw the grandstand, using standard keyboard graphics. Lines 140-195 explain the rules.

Lines 200-220 display the horses' names. The FOR/NEXT loop in 220 PRINTs horse names from array H\$(25). The element of the array to be printed next is determined by the variable N. N is initialized in line 198. N is incremented after each name is printed so, the next time through the loop, the next element (horse name) in the array will be displayed.

Lines 225-245 determine the odds for the five horses in the next race. Variable SN is a random number from 1 to 10 (line 225). SN selects the start of the set of odds for the coming race. In line 240, element SN of array SN(10) identifies where in arrays ODD\$(25) and ODD(25) to begin. It would be easy to start anywhere in those arrays and print the next five items, but that cannot be allowed because the arrays are arranged in exact groups of five and cannot be mixed.

In lines 250-266, the player selects his horse. The player's pick is YH\$ (YH stands for Your Horse). We are essentially interested in a number here, since it is the number of the horse, not the name, which figures in later calculations. Why, then, the use of the alphanumeric string variable YH\$? It is possible to use a numeric variable, but if the player presses a letter key by mistake the message "?REDO FROM START" appears. By using the string variable YH\$, we can trap accidental wrong inputs. (This concept of error trapping was introduced in Chapter 5, and we cannot overemphasize its importance.) Line 260 ends with IF YH\$="" THEN 260. This means that pressing the return key without making a selection will not foul up the program.

In line 262, the handy VAL statement is used. VAL returns the numerical value of any number or series of numbers at the beginning of a string variable. If the variable is "123", VAL will return

123. If the variable is "12B", VAL will return 12 as the value. If a non-number is the first character in a string, VAL returns 0 (zero) as the value. Line 270, by using VAL, will only allow inputs of numbers from 1 to 5, which is what we want. Later, in line 270, VAL is used to change YH\$ into the numeric variable YH.

In lines 270-295 the player places his bet. The amount bet, variable B, must be more than zero and not more than the amount of money on hand (SM). Line 280 makes sure the amount bet is legal. This is done with logical comparisons through the greater than (>) and less than (<) operators.

Line 300 picks the winner—variable W is a random number from 1 to 10. Line 305 contains the horse-animation sequence. (More on this one later.)

Lines 310-345 calculate and announce the results. In 310, if the value of W is greater than 4, then W is made equal to FAV(SN). This changes W to the number of the favorite horse. Array FAV() contains the numbers of the winning horses in each of the five races for that game (day). If the favorite does not win, a winner from among the other four horses is picked in line 315. Actually, all five horses can be chosen here, so the last portion of the line filters out the favorite's number. (The favorite shouldn't get two chances to win!)

In line 330 the number of the player's horse is compared to the winning horse. If the player's horse wins, the amount won is figured by multiplying the odds factor (RO(W)) by the amount bet (B). The winnings (AW) are then added to the total (SM). If the chosen horse loses, the player's money is decreased (line 335) by the amount bet.

Lines 351-353 determine if the player has money left to bet. As long as SM (total amount of money) is greater than 0, the game goes on. Line 360 checks to see if the last race is over. If R (number of races) exceeds 5, the program branches to 400. Lines 400-440 are the end-game routine. If the player pressed function key one (F1\$) before X is incremented to 3000, the game will start again.

Animation

The last segment of *A Day at the Races* is the animation routine, lines 5000-5280. This routine shows five horses racing across the screen, left to right. In this case, the animation is a combination of moving animation and in-place animation. The horses do not simply move, they change position as they move.

To achieve the moving-horse effect, four custom characters are used. Each horse is shown in two running positions. Each position is made up of two adjacent characters. As the horses move across the screen, they are displayed alternately in one position, then the other. Each position is contained in a string variable (A\$,B\$).

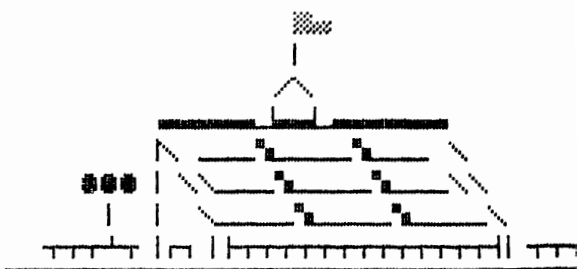
In line 5115 array G(5) is filled with zeros. Line 5120 defines variables A\$ and B\$ with custom characters. The FOR/NEXT loop I1 is set to 20 (the number of times each horse makes a move) in line 5130. In line 5140, the FOR/NEXT loop I2 is set to 5, and the number of horses is drawn. Line 5150 determines the horizontal (H) and vertical (V) coordinates for the next horse to be drawn, and loop I3 in line 5160 erases the horse at its old position. In line 5170, G represents the number of spaces a particular horse will advance in the next move. Array G(5) temporarily stores the advance (G) for each horse, and line 5180 ends the race when any horse has reached the seventeenth move. Line 5190 prints A\$ at the new position for whichever horse is making its move, and a short pause loop follows. Lines 5210-5280 are a mirror image of lines 5115-5200. At the end of this section, B\$ (the second horse) is drawn.

If you would like to make the horse animation more than mere decoration, the place to start is line 5180. Use this line to identify which horse finishes first. Of course, you will have to rethink the concept of having a race favorite and of betting odds as well.

A Day at the Races offers a lot of room for enhancement. Such enhancements as betting on more than one horse or figuring place and show positions are quite feasible. Create the necessary variables and keep track of the how they interact.

A Day at the Races

```
1 POKE53281,1:POKE53280,1:PRINT"{BLK}"
2 PRINT"{CLR}{DOWN 3}{RIGHT 2}THE FIRST
RACE{DOWN 3}WILL BEGIN"
3 PRINT:PRINT: PRINT"{RIGHT 20}IN ONE M
INUTE."
4 FORI=1TO4000:NEXT:PRINT"{CLR}"
10 REM *****
20 REM * A DAY AT THE RACES *
25 REM * VERSION 23 *
30 REM * COPYRIGHT (C) 1983 *
40 REM * BY WILLIAM L. RUPP *
50 REM *****
60 GOTO2000
61 GOTO4000
62 GOTO140
70 PRINT:PRINT" {RVS} PRESS ANY KEY T
O CONTINUE {ROFF}"
71 X=X+1:IF X=1000 THEN 75
72 GET X$:IF X$="" THEN 71
75 X=0:RETURN
100 REM *****
110 REM * MAIN BODY OF PROGRAM *
120 REM *****
125 SM=100::R=1:F1$="{F1}"
130 PRINT"{CLR}{DOWN 2}{RIGHT 2} {RIG
HT 3}{RVS} A DAY AT THE RACES {ROFF}":PR
INT
131 PRINT"
132 PRINT"
133 PRINT"
134 PRINT"
135 PRINT"
136 PRINT"
137 PRINT"
138 PRINT"
139 PRINT"
":PRINT:RETURN
140 GOSUB100: PRINT"{DOWN}{RIGHT 2} YOU
```



```

WILL HAVE A CHANCE TO BET"
150 PRINT" ON ONE HORSE IN EACH OF FIV
E RACES."
160 PRINT" YOU WILL START WITH $100, A
ND YOUR "
170 PRINT" RUNNING TOTAL WILL BE SHOWN
180 PRINT" THE ODDS FOR ALL HORSES WIL
L BE"
190 PRINT" DISPLAYED BEFORE EACH RACE.
"
195 GOSUB 70
198 N=1
200 PRINT"{CLR}{DOWN 4}{RIGHT 7} THIS IS
THE "R$(R)" RACE. "
210 PRINT: PRINT"{RVS} HERE ARE THE
HORSES AND THEIR ODDS."
215 PRINT
220 FORI=1TO5:PRINTI"{LEFT}. "H$(N):N=N+
1:NEXT
225 SN=INT(RND(1)*10+1)
230 PRINT"{UP 5}";:
240 FORI=SN(SN)TO SN(SN)+4:PRINTSPC(25)O
DD$(I):NEXT
243 RO=SN(SN):FOR I=1TO5:RO(I)=ODDS(RO):
RO=RO+1:NEXT
245 PRINT
250 PRINT" SELECT YOUR HORSE. ENTER 1,2
,3,4, OR 5"
260 PRINTNU$"{UP}";:INPUT YH$:IF VAL(YH$
)>0 AND VAL(YH$)<6 THEN 270
265 PRINT"{UP 2}{RVS} PLEASE CHOOSE AGAI
N. {ROFF} "":
266 FORI=1TO1500:NEXT:PRINT"{UP 2}": GO
TO260
270 PRINT" YOU NOW HAVE $"SM".00.":YH=VA
L(YH$)
280 INPUT "WHAT IS YOUR BET ";B:IF B>0 A
NDB<=SM THEN 300
290 PRINT"{UP}{RVS} SORRY, YOU MUST BET
AGAIN {ROFF} "
295 FORI=1TO1500:NEXT:PRINT"{UP}" NU$ "

```



```

{UP 3}:GOTO280
300 W=INT(RND(1)*10+1)
305 GOSUB130:GOSUB5000
310 IF W>5 THEN W=FAV(SN):GOTO 320
315 W=INT(RND(1)*5+1):IF W=FAV(SN)THEN 3
15
320 PRINT:PRINT"{DOWN}{RVS} -THE WINNER
IS HORSE NUMBER-{ROFF}"W" "
330 IF YH=W THEN AW=INT(RO(W)*B):SM=SM+A
W :GOTO340
335 PRINT"SORRY, YOUR HORSE DID NOT WIN"
:SM=INT(SM-B):FORI=1TO2000:NEXT: GOTO351

340 PRINT" YOU HAVE WON "AW"DOLLARS"
345 GOTO 355
350 PRINT
351 IF SM>0 THEN 355
353 PRINT:PRINT"{RVS} *** YOU HAVE NO MO
NEY LEFT ***{ROFF}":PRINT:PRINT"THAT'S A
LL..."
354 GOTO425
355 GOSUB 70
360 R=R+1:IF R=6 THEN 400
370 GOTO 200
400 PRINT"{CLR}{DOWN 3}{RIGHT 4} THE
RACES ARE OVER."
410 PRINT" YOU END THE DAY WITH {RVS}"SM
"{ROFF} DOLLARS"
420 PRINT" IN YOUR POCKET. SEE YOU NEXT
TIME!"
425 PRINT"PRESS 'FUNCTION 1' KEY(F1) TO
PLAY AGAIN"
428 X=0
429 X=X+1:IF X=3000 THEN PRINT"{CLR}":EN
D
430 GET Y$:IF Y$="" THEN 429
435 IF Y$=F1$ THEN62
440 END
2000 REM =====
2010 REM = =
2020 REM = ALTERNATE CHARACTER =

```

```

2030 REM = SET; SET-UP ROUTINE =
2040 REM = COPYRIGHT (C) 1983 =
2050 REM = BY WILLIAM L. RUPP =
2060 REM = =
2070 REM =====
2080 PRINTCHR$(142)
2090 :
2100 REM =====
2110 REM ==PROGRAMABLE CHARACTER==
2120 REM == BASIC ROUTINE ==
2130 REM = MOVES 255 CHAR SET =
2140 REM = TO RAM: CUSTOM CHAR =
2150 REM = CAN THEN BE CREATED =
2160 REM =====
2170 POKE52,128:POKE56,128:CLR
2175 POKE56576,(PEEK(56576)AND252)OR1
2177 POKE 648,136
2180 POKE56334,PEEK(56334)AND 254
2190 POKE1,PEEK(1)AND251
2200 FOR I=0TO2047
2210 POKEI+32768,PEEK(I+53248):NEXT
2220 POKE1,PEEK(1) OR 4
2230 POKE56334,PEEK(56334)OR1
2240 POKE53272,32
2250 REM *****
2260 REM * CHR$(186)+(187) AND *
2270 REM * CHR$(188)+(189) ARE *
2280 REM * RACE HORSE *
2290 REM * CHARACTERS *
2300 REM *****
2310 FOR I=0TO31:READ A:POKE33752+I,A:NE
XT
2320 DATA 0,0,57,79,15,8,20,34,104,204,1
58,240,224,48,72,132
2330 DATA 0,65,49,15,15,24,120,72,200,14
0,30,240,224,56,40,32
3000 GOT061
4000 REM *****
4001 REM * INITIAL VARIABLES & *
4002 REM * DATA ARRAYS *
4003 REM *****

```

```

4004 REM YH=YOUR HORSE,F$(5)=FIELD OF
      FIVE HORSES PER RACE
4005 REM F1,F3,F5,F7=FUNCTION KEYS
4006 REM ODDS(50)=10 SETS(5 EACH) OF
      ODDS
4007 REM OR(5)=ARRAY OF FIVE ODDS FOR
      ANY GIVEN RACE
4008 REM H$(25)=ARRAY OF HORSE NAMES FOR
      ALL FIVE RACES
4009 REM SM=STARTING $,B=AMOUNT BET ON
      A SINGLE RACE
4010 REM AW=TOTAL AMNT OF $ WON
      INCLUDING STARTING AMNT
4011 REM WR=AMNT WON ON A SINGLE RACE
4012 REM R=NUMBER OF CURRENT RACE
4013 REM W=NUMBER OF WINNING HORSE
4014 REM FAV(10)=FAVORITE HORSE, BY
      NUMBER IN EACH GROUP OF ODDS
4015 F1$="{F1}":F3$="{F3}":F5$="{F5}":F7
      $="{F7}"
4016 NU$="
      "
4017 DIM ODD$(50):FORI=1TO50:READ ODD$(I
      ):NEXT
4018 DATA 3-2,2-1,8-1,4-3,12-5,6-2,18-1,
      5-3,4-1,7-5
4019 DATA 4-3,5-2,6-4,30-1,18-7,8-6,4-1,
      3-2,9-5,6-5
4020 DATA 10-9,3-2,5-4,6-1,8-7,45-1,13-1
      1,8-2,16-3,2-1
4021 DATA 9-4,3-1,6-4,8-2,5-3,66-1,15-3,
      8-2,7-4,5-1
4022 DATA 3-2,2-1,11-9,8-7,16-1,7-1,13-3
      ,5-4,3-1,8-5
4023 DIM ODDS(50):FORI=1TO50:READ ODDS(I
      ):NEXT
4024 DATA .5,1,7,.33,1.4,2,17,.667,3,.4,
4025 DATA .111,.5,.25,5,.143,44,.182,3,4
4026 DATA .5,1,.222,.143,14,6,3.33,.25,2
      ,.6
4027 DIM FAV(10):FORI=1TO10 :READFAV(I):

```

```

NEXT
4028 DATA 4,5,1,5,1,2,3,4,4,3
4029 DIM SN(10):FORI=1TO10:READ SN(I):NE
XT
4030 DATA 1,6,11,16,21,26,31,36,41,46
4031 DIM R$(5):FORI=1TO5:READR$(I):NEXT
4032 DATA FIRST,SECOND,THIRD,FOURTH,FIFT
H
4033 DIM H$(25):FORI=1TO25:READ H$(I):NE
XT
4034 DATA SHEER DELIGHT,COUNT ME OUT,CON
FUSION,SWORD DANCER,RELAXIN' FREDDIE
4035 DATA PIQUANT,AZURE SKIES,IRON MAIDE
N,LEMON TWIST, APPLE DOC
4036 DATA OUTSKIRTS OF TOWN,SOCIAL CLIMB
ER,DEBBIE'S DREAM,SPINTOP,MY CHOICE
4037 DATA BLUE CHIP,GUNMETAL,COMMANDO ST
EVE,QUIET AFTERNOON,MICRO MANIA
4038 DATA BIFOCAL,UTTER MADNESS,DIZZY,LA
TE AS USUAL,NO VITAL SIGNS
4039 GOT062
5000 GOT05090
5010 REM *****
5020 REM * PLOT SUBROUTINE *
5030 REM *****
5040 :
5050 POKE781,V:POKE782,H:POKE783,0
5060 SYS 65520
5070 RETURN
5080 :
5090 REM *****
5100 REM * HORSE ANIMATION *
5110 REM *****
5115 FORI=1TO5:G(I)=0:NEXT
5120 A$="{#187}{#190}":B$="{#189}{#188}
"
5130 FORI1 =1TO20
5140 FORI2 =1TO5
5150 V=I2+12:H=G(I2):GOSUB5050
5160 FORI3=0TOG(I2):PRINT" ";:NEXTI3
5170 G=INT(RND(1)*2+1):G(I2)=G(I2)+G

```

```

5180 IF G(I2)>17 THEN GOT05280
5190 PRINTA$:FORI=1T075 :NEXT:REM -PAUSE

5200 NEXT I2
5210 FORI2 =1T05
5220 V=I2+12:H=G(I2):GOSUB5050
5230 FORI3=0T0G(I2):PRINT" ";:NEXTI3
5240 G=INT(RND(1)*2+1):G(I2)=G(I2)+G
5245 IF G(I2)>17 THEN GOT05280
5250 PRINTB$:FORI=1T075 :NEXT:REM -PAUSE

5260 NEXT I2
5270 NEXT I1
5280 FORI=1T05:G(I)=0:NEXT:RETURN

```

The following lines show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustration in *A Day at the Races*.

```

131 PRINT"                {#166}{#168}
132 PRINT"                {#180}
133 PRINT"                {#206}{#205}
134 PRINT"                {#185 5}{#186}{#185
2}{#204}{#185 6}
135 PRINT"                {#170}{#205} {#164 3}
{#191}{#164 4}{#191}{#164 3} {#205}
136 PRINT"                {#209 3}{#170} {#205 2}{
#164 3}{#191}{#164 4}{#191}{#164 3}{#205
2}
137 PRINT"                {#194} {#170} {#205}{#
164 4}{#191}{#164 4}{#191}{#164 4}{#205}

138 PRINT"                {#178 3}{#177}{#178}{#170}
{#176}{#174}{#217}{#171}{#178 13}{#179}{
#165}{#178 3}
139 PRINT"                {#163 32}":PRINT:RETURN

```


Chapter 31

GOAL TO GO!

Goal to Go! is a football simulation game for the Commodore 64. It is designed for two players. Technically, the game is a discrete simulation because the physical actions of an actual football game are portrayed symbolically rather than by mathematical formulas. This does not mean too much to the player, however, since playing in a football game is not what is being simulated!

In *Goal to Go!* the player assumes the role of football coach. What is simulated here is a coach's decision-making process. It not a perfect representation, of course. Many elements of the real game are left out. The "coach" of *Goal to Go!* cannot actually see how well or poorly certain players are performing, nor can he get reports from pressbox spotters assigned to keep track of the opponent's formations.

Within its limitations, however, *Goal to Go!* presents the player with the major decisions a coach must make in the course of a football game. The coach (as we will now refer to the player) must choose offensive plays and defensive formations on each down. The coach may elect to punt on fourth down or to go for it. The coach may try a long pass on the last play of the game, or he may order a field goal attempt. In other words, the coach participates in a strategic simulation. And football, perhaps more than any other team sport, involves strategy.

Goal to Go! is one of the longest games in this book. Even so, not all the strategic considerations of a real game could be included. As you design your games, you will have to decide which elements to leave out, just as we did. For one thing, weather factors have not been included. A dry field on a warm, windless day is assumed.

A somewhat more significant factor has also been left out: penalties. Provisions have been made to add a penalty routine, should you wish to create one. An ideal football game has few penalties anyway, so we felt this compromise was acceptable. Injuries, likewise, are not a factor in this game.

Here is the roadmap for *Goal to Go!*:

1. Both coaches name their teams.
2. A coin toss decides which team receives.
3. The receiving team runs back the kickoff.
4. The team with the ball has four offensive downs. The offensive coach chooses a play from a menu of running and passing plays.
5. The defensive coach chooses a defensive formation from a menu.
6. The results of the play are calculated and displayed. The relevant facts are shown on a screen scoreboard.
7. Play continues in a standard football fashion until the first half is over.
8. The second half begins with the ball received by the team that kicked off to start the game.
9. At the end of the fourth quarter, the winner is proclaimed.

As you'll agree, this is pretty much how most people would describe the major events in a real football game.

We started this chapter by saying that this is a discrete simulation. There are no complicated mathematical formulas constantly figuring the physical interactions of twenty-two men running and colliding on a gridiron. The essential elements or decision points in a football game are calculated by random numbers and logical relationships. The validity of such a game rests on the validity of the assumptions. The reader is free, and encouraged, to substitute

his own assumptions for ours should he believe that his assumptions are more realistic. We merely offer a framework in which specific assumptions can function.

What do we mean by assumptions? Ah, that is where the Monday morning quarterback comes into his own! What are the odds that a certain defensive alignment will stop a given play in a particular game situation? Let's say it is fourth down, with five yards to go for a first down. Team A has the ball on its own 44 yard line. Three minutes remain in the game, and Team A is behind by six points. As coach of Team A, do you punt the ball, hoping to get it back deep in the other team's territory? Do you decide to go for the first down? If you do, should you try a running play or a pass?

As designer of Commodore 64 simulation games, you must decide what probabilities to assign to different combinations of offensive plays and defensive formations in situations such as the one just described. Specifically, what probability of success should a running play have against a seven-man line? What success should the same play have against a defense that is dropping back most of its players to guard against a long pass? It's fun to think of the various possibilities, but first we need to look at the overall game structure.

Game Modules

Goal to Go! has modules, or sections. Each module functions independently of the others. The main game routine ties everything together. Here are the game modules:

1. Clear bottom portion of screen
2. Scoreboard display
3. Reset number of plays in quarter
4. Begin new quarter
5. Begin second half

6. Punt
7. Field goal
8. Fumble
9. End of game
10. Touchdown
11. Safety
12. Begin game
13. Kickoff
14. Play selection
15. Calculate defensive factor
16. Calculate play results

In addition, `DATA` statements define important arrays and a section that sets up the two sprites which enhance the scoreboard display.

Even a limited version of football is extremely complicated. Successfully transferring the game to the Commodore 64 requires, first of all, an understanding of that game's basic components. Next, you must set up variables which will correspond to those components. Finally, you must design program modules (algorithms) which will make those variables interact. The interaction of the variables produces the information which is ultimately interpreted as events. To illustrate how this is done, we will examine how the real game of football was transferred to the Commodore 64's "playing field."

Field Position

The first element in football, after the ball itself, is the playing field. A variable, Y , is assigned to represent the position of the ball on the field. The possible values of Y are 1-99. Zero and 100 represent the goal lines. To be on the 0 or 100 yard line is to score a touchdown; that is why Y never equals those values. Think of the field as a line with 99 yard markers ranging from 1 to 99.

Immediately, a problem arises. Yard lines 1 to 50 are fine, but what about the other half of the field? Did you ever hear of the 65 yard line? A second variable, DY , is needed. Y stands for yardline, and DY stands for Display Yard. DY tells the coaches where the line of scrimmage is. DY is derived from Y . A simple algorithm changes Y into DY for scoreboard display. If Y is 50 or less, then Y and DY are equal. When Y is greater than 50, DY is always 100, less the value of Y . It's really simple. If Y equals 65, then DY equals 100 minus 65, which is 35.

If you are confused, pull out your trusty pencil and paper and draw a long line with markers every five yards, just like a real football field. Now starting at one end, label those markers by fives, all the way to 100 (5, 10, 15 . . . 90, 95, 100). After you've done this, renumber the yard lines as you'd number a real football field (5, 10, 15 . . . 45, 50, 45 . . . 15, 10, 5). Now if the ball is on the 65 on one scale, it's on the 35 on the other.

To repeat, there are two variables: Y to represent the actual location on the imaginary field, and DY which is the translation used to update the scoreboard.

Gains and Losses

Now that we have a method to keep track of the line of scrimmage, the next question is how do we calculate the gains and losses of two teams running in opposite directions? If there were only one team, there would be no problem. The formula $Y = Y + RS$, with RS standing for Results of the Play, would work just fine. If the team gained 5 yards, the formula would be $Y = Y + 5$. If the team lost 5 yards, the formula would be $Y = Y + (-5)$; in other words, Y minus 5.

But the real game of football, and *Goal to Go!*, involves two teams, so let's see how we keep track of both team positions. Team One will defend the left goal; Team Two, the right. Team One always moves left to right (1-99 on the Y field); Team Two, right to left (99-1 on the Y field). If Team One is on the 40 yard line and gains 5 yards, the formula to update the line of scrimmage (Y) would be $Y=40+5$, putting them on the 45. No problem. But if Team Two is on the 40 (Y scale) and gains 5 yards, and we use the formula $Y=40+5$ (the equivalent of $Y=Y+RS$), Team Two is also on the 45, which would be a five yard loss!

The solution is simple: create another variable, AR. AR stands for Adjusted Result. The formula for updating the line of scrimmage is now a two-part process. First, the absolute result (RS) of the play (i.e., gain or loss) is multiplied by the possession variable, PO. If Team One has the ball, then PO equals 1. If Team Two has the ball, then PO equals -1 . The product of this multiplication is AR. AR can then be added to Y to yield the correct line of scrimmage.

A short example will clarify:

1. Team one has the ball on the 35 (Y scale).
2. They gain 6 yards. RS, the result, therefore equals 6.
3. RS (6) times PO (1 for Team One) is 6.
4. Y (35) plus AR (6) is 41. Team One's new line of scrimmage is the 41 yard line.
5. Team One then loses 3 yards ($RS=-3$).
6. AR equals RS (-3) times PO (1), which is -3 .
7. Y (41) equals Y plus AR (-3), which is 38.

8. The line of scrimmage is now the 38 yard line ($Y=38$).
9. On the next play Team Two recovers a fumble at the line of scrimmage ($Y=38$).
10. Team Two gains 9 yards on the next play.
11. AR equals $RS(9)$ times $PO(-1)$, because Team Two has possession), so $AR=-9$.
12. Y equals 38 plus (-9) , which is 29. Team Two gets its gain.
13. If Team Two loses yards, AR will be a positive number, which, when added to Y will move them back towards their goal.

So a gain is a positive number for Team One and a negative number for Team Two. Lines 2280 and 2290 do the calculations necessary to keep track of gains and losses for both teams. The key is the variable PO , which can be a positive or negative 1.

The beauty of having a field that runs from 1-99 instead of stopping at 50 and counting down to 1 again is that it facilitates figuring gains and losses, no matter where the line of scrimmage is or which team has the ball.

We have closely examined the technique for keeping track of each team's movement and field position. This is important to understand, because everything else in the game is useless if there is no way to determine where the teams are and how to change their positions to reflect the results of plays.

Choosing Plays

The main game routine in *Goal to Go!* consists of the modules labeled Play Selection (lines 1660-2000), Calculate Defensive Factor (2010-2120), and Calculate Play Results (2130-2500). The coaches choose their offensive and defensive plays from menus displayed on the screen. The four broad categories are Run, Pass, Punt, and

Field Goal. The Commodore 64's built-in function keys are pressed to make the selection. In line 1680, string variables F1, F3, F5, and F7 are defined by pressing the respective keys inside quotation marks. You will undoubtedly find many other uses for the function keys.

If the choice is Run or Pass, a submenu appears. Lines 1830-1870 contain the Run menu routine. First, the subroutine at 145 is called. This routine clears the lower part of the screen, leaving the drawing of a football field intact at the top. Line 1835 sends the cursor to the top of the page (HOME) and then down 12 lines. In line 1850, a FOR/NEXT loop prints out the running plays. The plays are contained in string array OF\$(). All offensive plays, running and passing, are contained in this array. By using the index variable (FOR I= 1 TO 7 for running plays, and FOR I=8 TO 18 for passing plays), only one section of the array is displayed at a time. Then the coach is asked to choose the play he wants to run (variable OP). Line 1860 ensures that the input is legal; if the number entered is too big or too small, the choice must be made again. (If the coach presses a letter key, the Commodore 64 says "?REDO FROM START", and gives the player another chance.)

Now, in line 1950, it's the defensive coach's turn. Again, the Clear Bottom of Screen routine comes into action, followed by a FOR/NEXT loop which displays the possible defensive formations—from array DF\$(). The formation is selected (D\$). In the offensive play routine, the input was accepted as a numeric variable (OP). Here, the input is in the form of an alphanumeric variable (D\$). To check the validity of an alphanumeric variable, you must match it with all the valid entries, which is done in 1990. Since there are only four valid entries in this case, it's fairly simple. (In cases where many numeric inputs are acceptable, use of a numeric variable and the < and > symbols, as in line 1860, is handier. The alphanumeric variable method is especially useful in data file programming.)

The alphanumeric variable D\$ must be changed to a numeric variable for later calculations. This is done with VAL. DE=VAL(D\$) assigns the numerical value contained at the beginning of D\$ to variable DE. If D\$ begins with a letter, VAL assigns it a numerical value of 0. If D\$ equals "123A", its numerical value is 123.

Probability Factors and Play Results

The smart part of *Goal to Go!* is the Calculate Defensive Factor in lines 2010-2120. Here the offensive play and the defensive formation are compared, and a probability factor created. The underlying philosophy can be expressed in this way: "How good a chance does a particular defensive formation have of stopping a particular play?" The four defensive formations chosen for this game are 4-3-4 Standard, Blitz Pass Rush, Pass Prevent, and Goal Line Seven-Man Line. These are coded with the numbers 1-4.

The Standard defense brings neither advantage nor disadvantage, so the program branches to the next section if $DE = 1$ (line 2040). If DE equals 2, 3, or 4, the defensive factor is calculated in lines 2050-2100. For example, examine line 2050. The defensive formation is Blitz Pass Rush. If offensive play 3, 4, 6, 8, or 10 is chosen by the offensive coach, the defense is at a disadvantage. This results in a defensive factor (DF) of +2. If offensive play 1, 5, 7, 9, or 13 is chosen, the defense has an advantage, which gives a defensive factor of -2. The plus or minus defensive factors are assigned in lines 2110 and 2120.

Finally we get to the Calculate Play Results module (lines 2130-2500). Variable RD is generated in line 2190. RD determines the play's gain or loss from arrays. Look at line 2190:

```
2190 RD=INT(RND(1)* 15+1 + DF)
```

$\text{INT}(\text{RND}(1)*15+1)$ returns a random whole number from 1 to 15. By adding the defensive factor (DF) the result of the play is influenced by the decisions made by the coaches. The actual gain or loss is calculated in line 2210. Here we use a two-dimensional array, $\text{OF}(15,15)$. The 15 levels of this array ($\text{OF}(1,X); \text{OF}(2,X); \text{OF}(3,X); \text{OF}(4,X) \dots$) represent the 15 offensive play choices. The elements in each level are the outcomes ($\text{OF}(1,1); \text{OF}(1,2); \text{OF}(1,3); \text{OF}(1,4) \dots$). Since the outcomes are ranked in order from worst to best, adding a defensive factor of +2 favors the offense, while adding a defensive factor of -2 favors the defense.

Variables at Work

In line 2220, variable PQ is incremented by 1. PQ stands for plays run so far this quarter. The function of PQ is to keep game time. It would be possible to use the Commodore 64's built-in clock (try this one-line program to see the clock in action: `100 PRINT TIME:GOTO 100`). On the other hand, since the plays are being calculated by the C-64, not actually run by players on a field, we feel it is better to keep track of the number of plays run per quarter. The Reset Plays this Quarter routine determines how many plays will be allowed (28-35, determined randomly). When that number has been run (line 2400), the quarter variable (Q) is incremented.

Lines 2410-2470 determine the next quarter. X1, X2, and X3 are flags which prevent each quarter-change routine from being executed twice.

The current down (D) and yards to go for a first down (FD) are determined in the Calculate Play Results section. The opposite of the play results ($-RS$) is added to FD in line 2340 to determine the new yards to go to make a first. In line 2345, a first down is made if FD is less than 1; D is then reset to 1 (first down), and FD is set to 10 (yards to go for a first). In line 2350, ball possession transfers to the defense if the current down variable (D) is 5. Notice that possession (PO) is switched by using $PO = -PO$ ($-PO$ means the opposite of PO). PO could have been made 1 or 2, but by making it 1 for Team One and -1 for Team Two, the variable can be used to calculate which way Y moves on the imaginary field (see 2280).

A touchdown is scored in lines 2300 and 2310 and a safety suffered in 2320 and 2330, based on the value of PO with respect to each end of the field. If PO is 1, and the new position of the ball (Y) is greater than 99, then Team One has scored a touchdown. If PO equals -1 , then Team Two scores a touchdown if the new Y value is less than 1. The reverse in these cases means that the team with the ball has been caught in their own end zone for a safety, which gives two points to the other team. Since the team that has suffered the safety must kick off, the Safety subroutine (line 1310)

calculates where the team that scored the safety gets the ball after the other team kicks off. Variable SK defines where the ball is put into play after the safety. Variable SF\$ names the team which was caught in their own end zone.

The Scoreboard display routine, beginning at line 160, is extremely important. The variables to be PRINTed on the scoreboard are fairly self-explanatory. Note that the gridiron is drawn in another subroutine, beginning at line 10000. If the Y value is 50 or less, the left half of the field is displayed. When the line of scrimmage passes Y=50, the right half of the field is drawn. The two football player sprites are positioned at the line of scrimmage. The Y coordinate is fixed, while the X coordinate fluctuates depending on the value of Y.

Goal to Go!

```
1 POKES3269,0:REM -TURN OFF SPRITES-
20 PRINT"{CLR}":GOSUB340:REM -INITIAL TO
-
30 REM =====
50 REM = GOAL TO GO!  A FOOTBALL      =
60 REM = SIMULATION.  COPYRIGHT (C)  =
70 REM = 1983 BY WILLIAM L. RUPP     =
80 REM =====
90 PRINT"{DOWN 4}{RIGHT 5}{RVS} GOAL TO
GO {ROFF}"
100 GOSUB2510 :D=1:FD=10:Q=1
110 GOTO1340
120 Z=Z+1:IFZ=3500 THEN 140
125 PRINT"{RVS} PRESS ANY KEY TO CONTINU
E {ROFF}"
130 GET X$:IF X$=""THEN 130
140 PRINT"{ROFF}":      RETURN
142 REM *****
143 REM * CLEAR LOWER SCREEN *
144 REM *****
145 PRINT"{HOME}{DOWN 13}":FORI=1TO10:
146 PRINT"
      ":NEXT:RETURN
```

```

148 RETURN
150 REM
160 REM *****
170 REM * SCOREBOARD DISPLAY ROUTINE *
180 REM *****
182 REM
185 PRINT"{CLR}"
190 GOSUB10000:PRINT;:
200 REM --
210 PRINT"{RVS}SCORE {ROFF}"P$-"PS" "0$
   "-OS
220 PRINT"{RVS}DOWN-{ROFF}"D" ";:IF FD>7
0 THEN PRINT"{RVS}=GOAL TO GO={ROFF}":GO
T0230
225 PRINT"{RVS}YARDS TO FIRST-{ROFF}"FD
230 IF Y<50 THEN DY=Y:GOTO260
240 IF Y<50 THEN DY=Y:GOTO260
250 DY=100-Y
260 PRINT"{RVS}BALL ON-{ROFF}"DY" {RVS}P
LAYS THIS QUARTER-{ROFF}"PQ
270 IF PO=1THEN PD$=P$:GOTO290
280 PD$=0$
290 PRINT"{RVS}POSSESSION-{ROFF}"PD$;:P
RINT" {RVS}QUARTER-{ROFF}"Q
320 PRINT"{UP}":RETURN
330 REM *****
339 REM * RESET TOTAL PLAYS IN QUARTER *
340 REM *****
350 TQ=INT(RND(1)*8+28):PQ=0:RETURN
360 REM:
370 REM *****
380 REM * BEGIN SECOND QUARTER ROUTINE *
390 REM *****
395 POKES3269,0:REM TURN OFF SPRITES
400 PRINT"{CLR}{DOWN 3}{RIGHT 2}THAT'S T
HE END OF THE FIRST QUARTER":GOSUB340:RE
TURN
405 FOR I = 1 TO 1500:NEXT:RETURN
410 REM *****
420 REM * BEGIN SECOND HALF *
430 REM *****

```

```

435 POKE53269,0
440 PRINT"{CLR}{DOWN 2}{RIGHT 2}THAT'S T
HE END OF THE FIRST HALF":FD=10:D=1:GOSU
B 340
445 PRINT"{DOWN}      -GET READY FOR THE K
ICKOFF-":FORI=1TO1000:NEXT:PRINT:PRINT:P
RINT:PRINT
450 PO=-KO:GOSUB 1590:RETURN
460 REM *****
470 REM * BEGIN FOURTH QUARTER *
480 REM *****
485 POKE53269,0
490 PRINT"{CLR}{DOWN 2}{RIGHT 2}THAT'S T
HE END OF THE THIRD QUARTER":GOSUB120:GO
SUB340:RETURN
500 REM *****
510 REM * PUNT ROUTINE *
520 REM *****
530 PU=INT(RND(1)*20+35):PQ=PQ+1
540 IF PO*PU+Y <1 OR PO*PU+Y>99 THEN PRI
NT"TOUCHBACK":GOSUB120:GOTO590
550 PRINT"A PUNT OF "PU" YARDS":GOSUB120

560 PR=INT(RND(1)*10 + 5):PRINT"RETURN "
PR:FORI=1TO1500:NEXT
570 PU=PU-PR:Y=Y+PU*PO:PO=-PO
580 PQ=PQ+1:FD=10:D=1:RETURN
590 IF PO=1 AND PU+Y>100 THEN Y=80:FD=10
:D=1:PO=-1:RETURN
595 IF PO=-1 AND -PU+Y<1 THEN Y=20:PO=1:
FD=10:D=1:RETURN
600 RETURN
610 IF PO=-1AND PU+Y<1 THEN Y=20:FD=10:D
=1:RETURN
620 GOTO550
625 REM *****
630 REM * FIELD GOAL ROUTINE *
640 REM *****
650 IF PO =1THENTD =100:GOTO670
660 TD=0
670 FD=ABS(TD-Y):PQ=PQ+1

```

```

675 IF FO=<50 GOTO 690
680 PRINT"NO GOOD":GOSUB 120:PO=-PO:FD=1
  O
685 IF PO=1 THEN Y=INT(RND(1)*15+30):RET
URN
688 IF PO=-1 THEN Y=100-(INT(RND(1)*15+3
  0)):RETURN
690 IFFO>40 THEN FF=1:GOTO740
700 IFFO>30 THEN FF=2:GOTO740
710 IFFO>20 THEN FF=3:GOTO740
720 IFFO>10 THEN FF=4:GOTO740
730 IFFO>1 THEN FF=5:GOTO740
740 FG=INT(RND(1)*15+1+FF)
745 IF FG>10 THEN GOTO 760
750 IF FG<11 THEN PRINT"NO GOOD":PO=-PO:
  GOSUB120:
755 IF PO=1 THEN Y=20:RETURN
758 IF PO=-1 THEN Y=80:RETURN
760 POKE53269,0:PRINT"{CLR}{DOWN 3}{RIGH
  T 2}FIELD GOAL GOOD!":PRINT"{DOWN}{RIGHT
  2}STAND BY FOR THE KICKOFF":GOSUB120
770 IF PO=1 THEN PS=PS+3:GOTO790
780 IF PO=-1 THEN OS=OS+3
790 PO=-PO:
792 IF PO=1 THEN Y=INT(RND(1)*15+15)
794 IF PO=-1 THEN Y=INT(RND(1)*15+70)
795 RETURN
800 REM *****
810 REM * FUMBLE ROUTINE *
820 REM *****
830 FU=INT(RND(1)*15+1):IF FU<7THEN GOTO
  860
840 PRINT"FUMBLE RECOVERED BY DEFENSE":D
  =1:PO=-PO
842 FY=FUM(INT(RND(1)*15+1))
845 Y=Y+FY:RS=FY:GOSUB 120:LF=1:IF Y>89
  OR Y<11 THEN FD=100:RETURN
846 FD=10 :RETURN
860 PRINT"FUMBLE RECOVERED BY OFFENSE":F
  Y=FUM(INT(RND(1)*15+1)):GOSUB120
870 RS=FY: RETURN

```

```

880 REM *****
890 REM * END OF GAME ROUTINE *
895 REM *****
898 POKES3269,0
900 PRINT"{CLR}{DOWN 7}{RIGHT 6}      THE
GAME IS OVER {ROFF}"
910 IF PS>OS THEN 950
920 IF PS=OS THEN PRINT"{DOWN 4}      T
HE GAME ENDS IN A TIE ":END
940 PRINT"{DOWN 4}      THE "O$" WIN,"OS" T
O "PS:END
950 PRINT"{DOWN 4}      THE "P$" WIN,"PS" T
O "OS:END
960 REM *****
970 REM * TOUCHDOWN ROUTINE *
980 REM *****
990 IF PO=1 THEN PS=PS+6:TD$=P$
1000 IF PO=-1 THEN OS=OS+6:TD$=O$
1010 PRINT"THE "TD$" HAVE SCORED A TOUCH
DOWN!"
1020 CN=INT(RND(1)*15+1)
1030 INPUT "CONVERSION: (1) POINT OR (2)
: ";CN$
1040 IF CN$="1"THEN 1070
1050 IF CN$="2"THEN 1090
1060 GOTO 1030
1070 IF CN>5 THEN PRINT"ONE EXTRA POINT
GOOD":GOSUB 120:GOTO 1110
1080 PRINT"NO GOOD":GOSUB 120:PO=-PO:D=1
:FD=10: GOTO1145
1090 IF CN>10 THEN PRINT"TWO EXTRA POINT
S GOOD":GOSUB120:GOTO 1130
1100 PRINT"NO GOOD":GOSUB 120:PO=-PO:D=1
:FD=10:GOTO1145
1110 IF TD$=P$ THEN PS=PS+1:PO=-PO:D=1:F
D=10:GOTO1145
1120 OS=OS+1:PO=-PO:D=1:FD=10:GOTO1145
1130 IF TD$=P$ THEN PS =PS+2:PO=-PO:FD=1
O:D=1:GOTO1145
1140 IF TD$=O$ THEN OS =OS+2:PO=-PO:FD=1
O:D=1

```

```

1145 GOSUB 2390:GOTO1575
1150 REM :
1160 REM *****
1170 REM * PENALTY ROUTINE *
1180 REM * FOR FUTURE ENHANCEMENT *
1310 REM *****
1325 REM *****
1327 REM * SAFETY ROUTINE *
1328 REM *****
1329 SK=INT(RND(1)*30+30)
1330 IF PO=1THEN PO=-1:SF$=P$:OS=OS+2:Y=
SK:GOTO1332
1331 PO=1:SF$=0$:PS=PS+2:Y=100-SK
1332 PRINT"{DOWN}{RVS}THE "SF$" HAVE BEE
N CAUGHT FOR A SAFETY{ROFF}":GOSUB 2390:
FD=10:D=1
1333 GOSUB120: GOSUB 160:GOTO 1690
1335 REM *****
1337 REM * BEGIN GAME ROUTINE *
1338 REM *****
1340 PRINT"{CLR}{DOWN 4} THIS IS THE B
IG GAME!"
1350 PRINT" HEAD COACHES, NAME YOUR TE
AMS. "
1380 PRINT" COACH NUMBER ONE, WHAT IS"

1390 PRINT" THE NAME OF YOUR TEAM?"
1400 PRINT:INPUT "ENTER TEAM NAME: ";P$
1410 IF P$=""THEN 1400
1420 PRINT:PRINT" COACH NUMBER TWO, WH
AT IS"
1430 PRINT" THE NAME OF YOUR TEAM?"
1440 PRINT:INPUT"ENTER TEAM NAME: ";O$
1450 IF O$=""THEN 1440
1455 PRINT"{CLR}"
1460 PRINT"{CLR}{DOWN 8}{RIGHT 5} GET RE
ADY FOR THE COIN TOSS"
1470 PRINT:PRINT" COACH OF THE {RVS
}"P$ "{ROFF} PLEASE ENTER "
1480 PRINT" HEADS {RVS}(1){ROFF} OR
TAILS {RVS}(2){ROFF}"

```

```

1490 INPUT HT$:IF HT$="" THEN 1490
1500 IF HT$<>"1"AND HT$ <>"2" THEN 1490
1510 HT=VAL(HT$)
1520 FLIP =INT(RND(1)*2+1):KO=FLIP:IF FL
IP=1THEN FLIP$="HEADS ":GOTO1540
1530 FLIP$="TAILS ":
1540 IF HT=FLIP THENPRINT"IT'S "FLIP$:PO
=1:GOTO 1570:REM -P$ RECEIVES-
1550 HT$=FLIP$:PRINT"IT'S "HT$
1560 PO=-1:FD=10:D=1:PRINT" THE "O$" WI
LL RECEIVE.":GOSUB120:KO=PO:GOTO 1640
1570 PRINT"{DOWN 4}{RIGHT 4} THE "P$" WI
LL RECEIVE.":FORI=1TO2000:NEXT: GOTO1640

1575 REM *****
1577 REM * KICK OFF *
1578 REM *****
1579 POKE53269,0
1580 POKE53269,0:PRINT"{CLR}{DOWN 4}{RIG
HT 4}{RVS}{RIGHT}-KICK OFF- {ROFF}":
1581 PRINT"{DOWN 8}":FORI=1TO1000:NEXT
1590 IF PO=1 THEN Y=(INT(RND(1)*35+5)):D
Y=Y: GOTO 1620
1610 XY=(INT(RND(1)*35+5)):DY=XY:Y=100-X
Y
1620 PRINT"BALL RETURNED TO "DY" YARD LI
NE"
1630 GOSUB120:RETURN
1640 GOSUB 1580
1650 PRINT"{CLR}":GOSUB160
1660 REM *****
1670 REM * PLAY SELECTION *
1675 REM *****
1680 F1$="{F1}":F3$="{F3}":F5$="{F5}":F7
$="{F7}"
1690 GOSUB145: PRINT"{HOME}{DOWN 13}"
1700 PRINT"{DOWN} TEAM WITH BALL SELECT
"
1710 PRINT" TYPE OF OFFENSIVE PLAY
"
1720 PRINT" PRESS FUNCTION KEY INDICATI
NG
"
```

```

1730 PRINT" YOUR CHOICE " :PRIN
T
1740 PRINT" F1=RUN F5=PUNT"
1750 PRINT" F3=PASS F7=FIELD GOAL"
1760 PRINT" {RVS} ENTER CHOICE: {ROFF}";
:
1770 GET TP$:IF TP$="" THEN 1770
1780 IF TP$=F1$ THEN1830
1790 IF TP$=F3$ THEN1880
1800 IF TP$=F5$ THEN1930
1810 IF TP$=F7$ THEN 1940
1820 GOTO 1770
1830 GOSUB 145
1835 PRINT"{HOME}{DOWN 13}"
1840 PRINT"{DOWN}{RIGHT}CHOOSE TYPE OF R
UNNING PLAY"
1850 FORI=1TO7:PRINTI" "OF$(I)"
":NEXT:REM-INPUT "ENTER NUMBER : ";OP
1855 INPUT "ENTER NUMBER : ";OP$:OP=VAL(
OP$)
1860 IF OP<1 OR OP >7 GOTO 1830
1870 GOTO1950
1880 GOSUB 145:PRINT"{HOME}{DOWN 13}"
1890 PRINT"{DOWN}{RIGHT}CHOICE OF PASSIN
G PLAYS"
1900 FOR I=8TO13:PRINTI"."OF$(I)" ":NEX
T
1905 INPUT"ENTER NUMBER: ";OP$:OP=VAL(OP
$)
1910 IF OP<8 OR OP>13 GOTO 1880
1920 GOTO1950
1930 GOSUB 510:FD=10:D=1:GOTO2385
1940 GOSUB 630:FD=10:D=1:GOTO2385
1950 GOSUB145:PRINT"{HOME}{DOWN 13}"
1960 PRINT"{DOWN}{RIGHT 2}SELECT DEFENSI
VE FORMATION"
1970 FORI=1TO4:PRINT" "I"."DF$(I)"
":NEXT
1980 PRINT:INPUT"{RIGHT 7}ENTER NUMBER O
F DEFENSE: ";D$
1990 IF D$<>"1" ANDD$<>"2" ANDD$ <>"3"
ANDD$<>"4" THEN GOTO 1950

```



```

2000 DE=VAL(D$)
2010 REM *****
2020 REM * CALCULATE DEFENSIVE FACTOR *
2025 REM *****
2040 IF DE=1 THEN 2140
2050 IF DE=2 THEN IF OP=3 OR OP=4 OR OP=
6 OR OP=8 OR OP=10 THEN 2110
2060 IF DE=2 THEN IF OP=10 OR OP=5 OR OP=7
OR OP=9 OR OP=13 THEN 2120:REM MINUS
2070 IF DE=3 THEN IF OP=6 OR OP=8 OR OP=
11 OR OP=12 THEN 2110
2080 IF DE=3 THEN IF OP=1 OR OP=7 OR OP=
9 OR OP=10 OR OP=13 THEN 2120
2090 IF DE=4 THEN IF OP>1 AND OP<8 OR OP
=13 THEN 2110
2100 IF DE=4 THEN IF OP>7 AND OP<13 THEN
2120
2110 DF=2:GOTO2140
2120 DF=-2
2130 REM *****
2140 REM * CALCULATE PLAY RESULTS *
2150 REM * RS=RESULTS : RD=RANDOM *
2160 REM *****
2190 RD=INT(RND(1)*15+1+DF)
2195 IF RD>15 THEN RD=15
2200 IF RD<1 THEN RD=1
2210 RS=OF(OP,RD):REM -GAIN OR LOSS-
2220 OP=0:D$="" :PQ=PQ+1
2238 IF RS=200 THEN GOSUB 810:IF LF=1 TH
EN LF=0: GOTO 2385
2240 PRINT"RESULT OF PLAY: "RS:GOSUB120
2280 AR=RS*PO: REM -CALCULATE Y MOVE-
2290 Y=Y+AR:
2300 IF PO=1 AND Y>99 THEN GOSUB 970:GOT
O2385
2310 IF PO=1AND Y<1 THEN GOTO 1310
2320 IF PO=-1 AND Y<1 THEN GOSUB 970:GO
TO2385
2330 IF PO=-1 AND Y>99THEN GOTO 1310:GOT
O2385
2335 D=D+1
2340 FD=FD+(-RS)

```

```

2345 IF FD<1 THEN D=1:FD=10
2350 IF D=5 THEN PO=-PO:FD=10:D=1:PRINT"
{RVS}BALL GOES OVER ON DOWNS{ROFF}":GOSU
B 120
2355 IF PO=1 AND D=1 AND Y=>90 THEN FD=
100
2360 IF PO=-1 AND D=1 AND Y=<10 THEN FD=
100
2370 GOSUB 2390: GOTO2490
2385 GOSUB2390:GOTO 2490
2390 REM -CHECK WHICH QUARTER-
2400 IF PQ=TQTHEN Q=Q+1
2410 IF X1 THEN 2430
2420 IF Q=2 THEN GOSUB 380:X1=1:GOSUB120
:RETURN
2430 IF X2 THEN 2450
2440 IF Q=3 THEN GOSUB 420:X2=1:RETURN
2450 IF X3 THEN 2470
2460 IF Q=4 THEN GOSUB 470 :X3=1:RETURN
2470 IF Q=5 GOTO 890:
2480 RETURN
2490 GOSUB 160:DF=0:GOTO1700
2500 END
2510 REM *****
2515 REM *****
2518 REM * *
2520 REM * DATA ROUTINES & ARRAYS *
2525 REM * *
2528 REM *****
2529 REM *****
2530 REM -OF$(15)-OFFENSIVE PLAYS-
2540 DIM OF$(15)
2550 FORI=1TO15:READ OF$(I):NEXT
2560 DATA END RUN,OFF TACKLE SLANT,FULLB
ACK INSIDE DIVE
2570 DATA INSIDE TRAP,QUARTERBACK OPTION
,FLANKER REVERSE,FULLBACK DRAW
2580 DATA DROP BACK-LONG PASS,SWING PASS
TO RUNNING BACK
2590 DATA SHORT LOOK-IN PASS TO TIGHT EN
D,HALFBACK PASS

```

```

2600 DATA QUARTERBACK ROLLOUT PASS, SCRE
EN PASS, FIELD GOAL TRY, PUNT
2610 REM -DF$- DEFFENSIVE FORMATIONS-
2620 FORI=1TO4:READ DF$(I):NEXT
2630 DATA STANDARD 4-3-4, BLITZ-PASS RUSH
, PASS PREVENT, GOAL LINE- 7 MAN FRONT
2640 REM - OFF(15,15)OFFENSIVE PLAY RESU
LT TABLE-
2650 DIM OF(15,15)
2660 FORI=1TO15
2670 FORJ=1TO15
2680 READ OF(I,J):NEXTJ,I
2690 DATA -13,200,-9,-4,-3,-2,-1,0,1,3,5
,8,11,13,22
2700 DATA -5,200,-4,-3,-2,-1,0,1,1,2,4,5
,7,11,15
2710 DATA -3,200,-2,-2,-1,0,1,3,3,4,5,6,
7,8,11
2720 DATA -5,200,-3,-2,-1,0,1,2,4,4,6,7,
9,11,13
2730 DATA -4,200,-3,-1,-1,0,1,3,5,6,6,8,
10,12,15
2740 DATA 200,-11,-8,-5,-2,-1,0,1,3,5,8,
11,15,19,24
2750 DATA -7,200,-5,-4,-2,0,1,2,2,4,5,7,
9,10,17
2760 DATA -8,0,0,200,0,0,0,13,15,19,21,2
5,28,37,41
2770 DATA -8,0,0,0,0,2,200,5,6,7,9,9,11,
13,16
2780 DATA -8,0,0,0,0,200,0,5,5,6,7,8,9,1
0,14
2790 DATA -8,0,0,0,0,0,200,0,0,13,16,22,
30,32,45
2800 DATA -8,0,0,0,0,0,0,100,7,9,11,12,1
4,18,26
2810 DATA 0,0,0,0,200,0,1,3,5,6,6,8,9,16
,23
2820 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0
2830 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0
2960 REM -FUMBLE RESULT ARRAY-

```

```

2970 DIM FUM(15)
2980 FORI=1TO15
2990 READ FUM(I) :NEXTI
3000 DATA -5,0,-8,-3,6,-1,-12,4,10,3,-
6,0
3010 DATA 7,0,5
4000 REM *****
4001 REM * FOOTBALL PLAYER *
4002 REM * SPRITE PROGRAM 4 *
4003 REM *****
4004 POKE53281,1:POKE53280,1
4005 FORI=0 TO 62:READ A:POKE832+I,A:NEX
T
4006 POKE895,0
4007 FORI=0TO62:READ A: POKE896+I,A:NEX
T:POKE959,0
4008 POKE2040,14:REM SPRITE 0 GETS DATA
FROM PAGE 13
4009 POKE2041,13
4010 POKE53269,0
4012 DATA 0,0,0,0,96,0,0,240,0,1,248,0,0
,240,0,0,96,0,0,240,0,1,248,0,7,216,0
4013 DATA 15,216,56,31,31,248,63,31,208,
59,0,0,27,128,0,25,192,0,12,96,0
4014 DATA 12,48,0,24,96,0,48,192,0,123,1
28,0,123,224,0
4015 DATA 0,0,0,0,120,0,0,252,0,0,252,0,
0,120,0,64,48,0,224,120,0,48,252,0
4016 DATA 25,190,0,15,159,0,6,15,128,0,3
1,128,0,59,128,0,115,128,0,227,128,1
4017 DATA 131,0,0,195,240,0,97,248,0,48,
24,0,240,24,0,224,0
4020 RETURN
10000 IF Y<=50 THEN GOTO 11000
10020 PRINT"{HOME} 50 45 40 35 30 25 20
15 10 5 GOAL"
10030 PRINT" ";:FORI=1TO34:PRINT"{#164}"
;:NEXT:PRINT
10040 FORI=1TO5:FORJ=1TO11
10050 PRINT" {#221}";:NEXTJ:PRINT:NEXTI

```

```

10060 PRINT"{UP 5}";:FORI=1TO5:PRINTSPC(
33)"{#166 2}":NEXT
10070 PRINT" ";:FORI=1TO34:PRINT"{#163}"
;:NEXT:PRINT:
10072 IF PO=1 THEN PRINT"          "P$"
--->":PRINT
10075 IF PO=-1 THEN PRINT"          <---
"0$:PRINT
10080 GOSUB 15000
10100 RETURN
11000 :
11010 PRINT"{HOME} GOAL 5 10 15 20 25 30
35 40 45 50
11020 PRINT" ";:FORI=1TO34:PRINT"{#164}"
;:NEXT:
11025 PRINT
11030 FORI=1TO5:FORJ=1TO11
11040 PRINT"  {#221}";:NEXTJ:PRINT:NEXTI
11050 PRINT"{UP 5}";:FORI=1TO5:PRINTSPC(
1)"{#166 2}":NEXT
11060 PRINT" ";:FORI=1TO34:PRINT"{#163}"
;:NEXT:PRINT:
11065 GOSUB 15000
11067 IF PO=1 THEN PRINT"          "P$"
--->":PRINT
11068 IF PO=-1 THEN PRINT"          <---
"0$:PRINT
11070 RETURN
15000 REM *****
15001 REM * MOVE SPRITES *
15002 REM *****
15115 POKE53269,3
15120 POKE53287,0:POKE53288,0
15130 IF Y<51 THEN X=Y*5+20
15132 IF Y>50 THEN X=Y*5+20-250
15135 IF X<20 THEN X=20
15140 POKE53249,65:POKE53251,65:REM-Y
COORDINATE OF BOTH SPRITES
15150 IF X<256 THEN POKE 53250,X:POKE532
64,PEEK(53264)AND 253

```

```
15155 IF X>255 THEN POKE 53250,X-256:POK
E53264,PEEK(53264) OR 2
15160 IF X+18<256 THEN POKE53248,X+18:PO
KE53264,PEEK(53264)AND254
15170 IF X+18>255 THEN POKE53264,PEEK(53
264)OR 1:POKE53248,(X+18)-256
15180 FORJ=1TO250:NEXT:RETURN
```

Chapter 32

FANTASY/ADVENTURE GAMES

Fantasy, role-playing, adventure and other similar games comprise a complete genre, or type, of computer game. These games usually stand out in sharp contrast to the fast-action arcade games commonly found in video game centers. The fantasy/adventure game appeals to people who like to give some thought to the game process (as opposed to merely testing their reflexes) and who like to become extremely involved with what they're doing. Often these games develop characters as the game progresses.

Perhaps the fantasy/adventure game is most appealing to those romantic souls who want to add a little excitement to their lives. The games are fascinating because the players, by using their imaginations, can escape from the problems of everyday life. Some gamers become so involved that they exclude all other activities and think of nothing but the game.

Fantasy/adventure games run the gamut from relatively simple maze games to complex dungeon and dragon games, complete with many characters and scenarios. Generally, the games follow a formula in which there are a number of rooms through which players move, encountering monsters, treasures, and pitfalls along the way. There is also usually some ultimate goal, such as slaying a dragon or finding a secret treasure. Of course, the fun and interest come from overcoming the obstacles between you and the goal. There must be an optimal number of challenges and a high intensity level so the game is neither too easy nor too difficult to win.

Caves and Rooms

Let's begin by thinking about the kind of a game we would like to construct. What are its essential parts? First, we must have a basic cave. (We are not necessarily referring to an actual cave, but rather to a structure or scenario, which we refer to as a "cave," that will be divided into "rooms" or sections.) The rooms may comprise a castle, haunted house, dungeon, city, woods, or any combination of the above. Some games use a large area including buildings, countryside, and caves. Design your cave game as you wish; just remember that it must be planned.

It is also important that you decide how many rooms (actually units through which your characters pass) your game will have. It is possible, but not desirable, to add rooms after your game is completed. Once we begin, you will see that the game structure must be carefully planned before the code writing begins. Break this rule, and you will encounter many errors and will likely become very confused.

Once the number of rooms is decided upon, you should determine the game's objective: Scoring points, whipping the monster, saving the damsel, finding the treasure, or simply surviving—all are acceptable. Next, you will want to populate the cave with monsters, treasures, weapons, whatever. There should be enough of each to maintain interest but not so many that your code fills up the computer's memory. Decide how large a program you are willing to write.

The programs in the following chapters should give you a good feel for planning your own cave game program. In *Danger Dungeon* (Chapter 33), the first of our cave games, we will make further use of subroutines and keyboard graphics. We will use computed variables and make more extensive use of the random number generator. The second game, *Scary Hall* (Chapter 35), is a who-done-it fantasy/adventure game. It uses nearly every feature of the Commodore 64, including lots of graphics and even real music. *Scary Hall* allows the player to have all sorts of strange encounters while attempting to solve the crime.

Chapter 33

DANGER DUNGEON

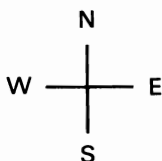
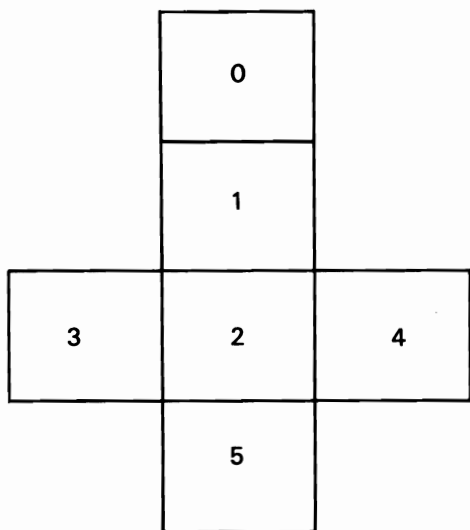
As discussed last chapter, we must make certain decisions regarding the structure of the game. What kind of “cave” are we constructing? How many rooms does it have? These are important questions that we must answer before any code is generated.

A cave game must have enough rooms for a player to move through with some variation, and the game must be sufficiently varied to require the player to play more than once to experience it fully. To meet these requirements, about six rooms, minimum, are necessary. Fewer would seem ridiculously puny to the player. Remember that all locations (rooms) through which the player moves must be recognized in the code, or the game won't work. Thus, a hall is a room. If you only number the rooms, but leave out all connectors (such as foyers, halls, and paths), your player will have no way to get from one room to another. If your game includes a field with foxholes, and the player can move through the holes, then each hole is also a room. Each and every location that will be described or mentioned is a room.

Let's give *Danger Dungeon* six rooms:

- 0 = entrance to Danger Dungeon (south)
- 1 = foyer (south of 0)
- 2 = large arched chamber (south of 1)
- 3 = chamber (west of 2)
- 4 = ballroom (east of 2)
- 5 = amphitheatre (south of 2)

In situations like this a map is helpful.



Identifying Variables

Now, let's begin writing the program. First we should identify the variables. The variable we have just decided upon is ROOM. We will name this RL (Room Location). Each time the player moves into a room, the variable is set at that room number. This is how we know the location of the player. We also must have a way of determining the player's command. We will call this variable C\$. We know that players will not necessarily give their directions in numbers, so we will allow them to tell us in a string.

Next is a list of variables which define the treasure, a score for the player, a variable to compute the hitting power of the player, the number of times hit, and the number of weapons the player has. In short, this list must include every factor which recurs during the game.

```
10 REM SC=SCORE RL=ROOM LOC C$=COMMAND
20 REM F=RANDOM NUMBER HT=HIT FACTOR
30 REM HP=HIT POWER T=TREASURE
40 REM W=WEAPONS
60 F=INT(RND(1)*10)
70 PRINT CHR$(147)
90 PRINT:PRINT:PRINT
```

As usual, we have made our REMarks at the beginning of the program. Line 50 again shows the usual screen-clearing command. In line 60, the random number selection is made, much as it was in *What's My Number?*

Moving from Room to Room

Now we need to figure out a way to get our players into the game and moved around. Suppose we allow them to enter a command to move.

```
100 INPUT"YOUR WISH IS MY COMMAND ";C$
```

Now, our problem is to establish routines to allow movement to take place in response to a command.

```
110 IF LEFT$(C$,1)="N" THEN GOTO 300
120 IF LEFT$(C$,1)="S" THEN 350
130 IF LEFT$(C$,1)="E" THEN 400
140 IF LEFT$(C$,1)="W" THEN 450
```

Just as we have seen in other games in this book, we are looking for only the left-most character typed by the player. This will not work if there are two commands beginning with the same letter. Also, the GOTOs send the program to other lines. This seems fine so far, but what about other commands, such as ATTACKing a monster, asking for your SCORE, GETting a treasure, or FLEEing from danger? We need separate sections in the program for these as well as the direction commands.

```
150 IF LEFT$(C$,1)="A" THEN 800
160 IF LEFT$(C$,1)="F" THEN 850
165 IF LEFT$(C$,1)="G" THEN GOSUB 900:
GOTO 100
170 GOSUB 950:GOTO 100
```

Notice that line 170 involves a subroutine, as opposed to a GOTO. We will come back to this later. We see that, after being sent to the subroutine, line 170 instructs the computer to GOTO line 100. Line 100 begins our input. This looks okay, except that the program begins with the input line, and there is no introduction to the game. Therefore, we must insert line 80.

```
80 GOTO 600
600 RL=0:REM ROOM 0
602 PRINT CHR$(147)
603 PRINT"YOU ARE STANDING BEFORE A
HEAVY"
605 PRINT"STONE DOOR. YOU MAY GO SOUTH"
610 PRINT "THROUGH THE DOOR OR NORTH
BACK"
615 PRINT "THE WAY YOU CAME."
616 GOTO 100
```

Using this technique means that the program begins by informing players where they stand and then inviting them to go north or south. After displaying the instructions, the program returns to line 100 for the command prompt. Do you begin to see how the game will work?

It is necessary for the program to return to the command input each time the player makes a move or issues any command. For this reason, this aspect of the program does not lend itself to the use of a subroutine. Remember, a subroutine must return to its point of origin. Here, each time a command is made the program must return to line 100 to accept a new command. It must return to line 100 regardless of where in the program the player is at a given moment.

It has already been suggested that a good way to write a program is to plan it well. With this in mind, it is a good idea to identify each part of the program which involves a different operation and then to insert these parts in the program in some orderly fashion.

Notice that lines 600-615 take care of the first room description. Suppose that you have all room descriptions following one another in the same area of the program. Let's designate lines 600-799 as the room-description routines. Remember that you should identify each description with a REMARK statement, clear the screen before presenting the description, and insert a RETURN at the end of each subroutine, or a GOTO at the end of each GOTO. Since we have GOTO 600 at the end of line 80, at the end of that room description we must use a GOTO 100 to return.

However, if we had told the program to GOSUB instead, then we would need a RETURN rather than a GOTO. Try writing the lines for the next room description. Look back at the map. Room one is the foyer. Set aside lines 620-639 for this routine. Once you have written this portion of the program, review example lines 600-799 at the end of the chapter. Or, you may use the example as a starting point, but you should not simply copy those lines in the sample program. You should write your own descriptions.

Before you can try these out, however, you need to get into the rooms in order to read their descriptions. Remember lines 110-165? Notice that line 110 sends the program to line 300, line 120 sends it to 350, etc. Again, branching (GOTOS) rather than subroutines is necessary here because the program must return to the command input line. Another way, which would allow the use of subroutines, is to replace the GOTO with a GOSUB and place a colon after the GOSUB line number. Follow this with a GOTO 100, and you will be able to use subroutines.

In either case, we must look at what will happen when the program goes to line 300 in the event the player presses N or NORTH. Remember that RL is the variable name for Room Location of the player. If the variable RL takes on the value of any room the player is occupying at the moment, then for example, RL=0 must mean the player is currently in room zero. But what if the player is not? Then there must be contingencies for each room players find themselves in. Lines 300, 315, 320, and 340 show how such a routine might be constructed.

```
300 REM NORTH
315 IF RL=1 THEN GOTO 600
320 IF RL=2 THEN GOTO 620
340 IF RL=5 THEN GOTO 640
```

Notice that line numbers 600, 620, and 640 are descriptions of rooms one, two, and five from the last exercise. So this is how the player gets into a room and the room gets described! But what about the other rooms? Room zero is the beginning room (and the northernmost) so there is no way to go north from there. Therefore, line 310 cannot direct the program to go to a room description. What would you like to have happen when the player attempts to go away from the cave? You should decide this, but this is a good time to end the game. You can do this by simply typing a line number and END.

```
1070 PRINT"YOU STRUT OUT WITH A SCORE
OF ";SC
1080 END
```

Now that we have somewhere for the player to go from room zero when he selects north as the direction to move, we can finish line 310.

```
310 IF RL=0 THEN PRINT"YOU EXIT
NORTH":GOTO 1070
```

Line 1070 is an individual touch and of course is not essential to the program. Line 310 might just as well have been directed to line 1080.

What happens when your player chooses an option which is not possible, such as moving north from rooms three or four? The answer is that *you* must take care of that eventuality—and any others which are similar in the course of the game. Let's look at the final routine for NORTH:

```
300 REM NORTH
310 IF RL=0 THEN PRINT "YOU EXIT
NORTH":GOTO 1070
315 IF RL=1 THEN GOTO 600
320 IF RL=2 THEN GOTO 620
325 IF RL=3 THEN GOSUB 1000
330 IF RL=4 THEN GOSUB 1000
340 IF RL=5 THEN GOTO 640
```

Subroutine 1000 will take care of mistakes, by informing the player of the error, then RETURNING to lines 325 or 330.

Try your hand at writing the routines for SOUTH, EAST, and WEST. If you do not yet understand how the room-direction and room-description routines interact, review what we've covered so far. (If you still have problems, refer to the sample program listing, but avoid using the listing as a crutch.) Set aside line numbers 300-500 for the direction routines, just as you allowed space for the room description line numbers.

Now that you've written the fundamental program, try playing the game. You should be able to "walk" through the rooms and receive the descriptions. You have given yourself a way to exit the program in lines 1070-1080, as long as you go north to room zero and then go north again.

Monsters, Weapons, and Treasures

As you move through the rooms, you might find yourself wondering what the game does besides provide a vicarious sight-seeing tour. What the game needs is some game ingredients. Ask yourself what you want to happen when you enter a room. Try placing a monster in a room.

```
622 RL=1
625 PRINT CHR$(147):PRINT"YOU ARE IN
A STRANGE ROOM"
630 PRINT"WITH EXITS LEADING NORTH
AND SOUTH"
631 M=M+1
632 PRINT"LOOK OUT-MONSTER!"
635 GOTO 100
```

Line 632 adds a monster. Line 631 sets up a counter for monsters encountered, in case you need to know how many monsters were contacted during the game. Refer back to line 150. The A response from a player was inserted so the player could attack monsters, so now there must then be an attack routine. Line 800 should reflect this, since line 150 sends the program there. The question becomes how to acknowledge the attack and provide an appropriate response to the player.

We have already become familiar with the random number generator, by using it in several games as well as in this one. If you aren't sure exactly how you want to handle this routine, you can experiment with the random number you have already selected in the beginning of the program. You might, for example, set up contingencies for each number selected or for ranges of numbers. Let's suppose you want the player to die if 8, 9, or 10 is selected. Then you tell the computer to PRINT "YOU ARE DEAD" and GOTO the end of the program. Of course, there are many other touches to add. You might want to have a score, so the player gets feedback at least at the end of the game. In our variable list at the beginning of the program we named variables HT, HP, T, SC,

and W for hit factor, hit power, treasure, score and weapons, respectively. Obviously, if you are going to attack, you want credit for hitting power, or for hitting at all.

Assuming that we decide to use the random number generated at the beginning of the program, we may take care of all of the attack contingencies in the same routine which starts at line 800.

```
800 REM ATTACK
805 IF F>7 THEN HP=HP+10
808 IF F<7 THEN HP=HP+5
810 IF F<4 THEN HP=HP+1
820 IF HP<5 AND HT<2 THEN PRINT"HE HIT BA
CK!":HT=HT+1:GOTO 100
822 IF HP<5 AND HT>1 THEN PRINT"YOU ARE
DEAD!":GOTO 1050
823 IF HP=5 AND HT<2 THEN PRINT"HE HIT B
ACK!":HT=HT+1:GOTO 100
824 IF HP=5 AND HT>1 THEN PRINT"HE RAN A
WAY.":SC=SC+50:GOTO 100
825 IF HP>5 THEN PRINT"YOU GOT HIM!":SC=
SC+1000:GOTO 100
```

Lines 805-810 set up the random factor for the hit power score, with HP being greater if a random number greater than seven was selected. Granted, this means that the player is in the hands of fate, but so are we all. Line 815 kills off the player and decreases the SCore by 100 points, then sends the program to the end. Lines 820-825 deal with the various alternatives which depend upon hitting power. Low hitting power and low hitting time mean that the monster will hit back (the lowest will mean the player dies), and highest hitting power means the player gets the monster. When the intermediate situation occurs (hitting power is five but hitting time is higher), the monster runs away.

The next routine we want to take care of is the FLEE portion of *Danger Dungeon*. (Refer to line 160.) In lines 850-860, the command to flee may result in the player finding the treasure (850), the player ending up in room three (855), or ending the game (860).

```

850 IF F=7 THEN PRINT"YOU FOUND THE TREASURE-YOU ARE RICH":GOTO 1070
855 IF F<7 THEN GOTO 660:
860 IF F>7 THEN GOTO 1050

```

You may find it strange that one must give the FLEE command to find the treasure. Of course, this is easily remedied. Simply create another routine or subroutine to allow the player to find the treasure under other circumstances.

Next, line 165 sends us to subroutine 900. This is the Got It subroutine. Since there are things to "get" in rooms three and four, the player must have a way to get them. Upon the command to GET, lines 900-920 take the appropriate action and announce to the player that he has "GOT IT!" Let's see how it works.

```

900 REM GOT IT
902 IF RL=3 THEN W=W+1
905 IF RL=3 AND F>7 THEN SC=SC+500
910 IF RL=3 AND F<=7 THEN SC=SC+50
912 IF RL=4 AND F<4 THEN SC=SC+2000
913 IF RL=4 THEN T=T+1
914 IF RL=4 AND F=>4 THEN SC=SC+500
915 IFW>1 AND RL=3THENPRINT"YOU ALREADY HAVE IT!":GOTO 945
916 IFT>1 AND RL=4THENPRINT"YOU ALREADY HAVE IT!":GOTO 945
920 PRINT"GOT IT! "

```

Lines 902-910 test for room three and, accordingly, increment the Weapon counter or increase the SCore. Lines 912-915 test for room four and carry out similar functions, except that room four contains treasure rather than weapons. Line 920 tells players whether they have the weapon or treasure. Hold the phone! What treasure? What weapon? Refer to lines 660-699, and add descriptions of the weapons or treasure to the general descriptions. Also do this for the monsters you added to the first room description. As in the complete program listing and the Got It subroutine, room three

has a random selection of either a sword or a mace weapon, and room four has options for an emerald or a chalice.

```
671 IF F>7 AND W<2 THEN PRINT"YOU SEE A  
CRYSTAL SWORD "  
672 IF F<=7 AND W<2 THEN PRINT"YOU SEE A  
ROUGH MACE "  
  
688 IF F<4 AND T<2 THEN PRINT"YOU SEE A  
LARGE SPARKLING EMERALD."  
689 IF F=>4 AND T<2 THEN PRINT"THERE IS  
A BEAUTIFUL SILVER CHALICE."
```

To complete the program as in the sample, you will need to add the goblin (lines 710-725) or some other monster. Also, in order to catch players who insist on entering invalid commands, you need a subroutine to reject those not provided for and to display a list of valid commands. Add line 170, then subroutine 950.

```
170 GOSUB 950:GOTO 100  
  
950 PRINT"I ONLY KNOW THESE WORDS:":PRIN  
T:PRINT  
960 PRINT"NORTH SOUTH EAST WEST "  
965 PRINT"ATTACK FLEE GET":PRINT:PRINT  
970 RETURN
```

Now, you are ready to play your game. Be sure to remember that there are nearly limitless ways to customize the program. Don't hesitate to do so.

Danger Dungeon

```
1 PRINTCHR$(147):POKE53281,1:PRINTCHR$(1  
44)  
10 REM SC=SCORE RL=ROOM LOC C#=COMMAND  
20 REM F=RANDOM NUMBER HT=HIT FACTOR
```

```

30 REM HP=HIT POWER T=TREASURE
40 REM W=WEAPONS
50 RL=0:M=0:W=0:SC=0:HP=0:HT=0:T=0
60 F=INT(RND(1)*10)
70 PRINT"{CLR}"
80 GOTO 600
90 PRINT:PRINT:PRINT
100 INPUT"YOUR WISH IS MY COMMAND ";C$
110 IF LEFT$(C$,1)="N" THEN GOTO 300
120 IF LEFT$(C$,1)="S" THEN GOTO 350
130 IF LEFT$(C$,1)="E" THEN GOTO 400
140 IF LEFT$(C$,1)="W" THEN GOTO 450
150 IF LEFT$(C$,1)="A" THEN GOTO 800
160 IF LEFT$(C$,1)="F" THEN GOTO 850
165 IF LEFT$(C$,1)="G" THEN GOSUB 900:GO
TO 100
170 GOSUB 950:GOTO 100
300 REM NORTH
310 IF RL=0 THEN PRINT"YOU EXIT NORTH":G
OTO 1070
315 IF RL=1 THEN:GOTO 600
320 IF RL=2 THEN:GOTO 620
325 IF RL=3 THEN GOSUB 1000
330 IF RL=4 THEN GOSUB 1000
340 IF RL=5 THEN:GOTO 640
345 GOTO 100
350 REM SOUTH
355 IF RL=0 THEN:GOTO 620
360 IF RL=1 THEN:GOTO 640
365 IF RL=2 THEN:GOTO 700
370 IF RL=3 THEN GOSUB 1000
375 IF RL=4 THEN GOSUB 1000
380 IF RL=5 THEN GOSUB 1000
390 GOTO 100
400 REM EAST
410 IF RL=0 THEN GOSUB 1000
415 IF RL=1 THEN GOSUB 1000
420 IF RL=2 THEN:GOTO 680
425 IF RL=3 THEN:GOTO 640
430 IF RL=4 THEN GOSUB 1000
440 IF RL=5 THEN GOSUB 1000

```

```

445 GOTO 100
450 REM WEST
455 IF RL=0 THEN GOSUB 1000
460 IF RL=1 THEN GOSUB 1000
465 IF RL=5 THEN GOSUB 1000
470 IF RL=2 THEN:GOTO 660
475 IF RL=4 THEN:GOTO 640
490 GOTO 100
600 REM ROOM 0
602 RL=0:PRINTCHR$(147)
604 PRINT"YOU ARE STANDING BEFORE A HEAV
Y"
605 PRINT"STONE DOOR. YOU MAY GO SOUTH "
610 PRINT"THROUGH THE DOOR OR NORTH BACK
"
615 PRINT"THE WAY YOU CAME."
*616 GOTO 100
620 REM ROOM 1
622 RL=1
625 PRINTCHR$(147):PRINT"YOU ARE IN A ST
RANGE ROOM"
630 PRINT"WITH EXITS LEADING NORTH AND S
OUTH"
631 M=M+1
632 PRINT"LOOK OUT-MONSTER!"
635 GOTO 100
640 REM ROOM 2
641 RL=2
642 PRINTCHR$(147):PRINT"YOU ARE IN A LA
RGE ARCHED CHAMBER"
644 PRINT"WITH A STRANGE STONE FIGURE ST
ARING"
646 PRINT"DOWN AT YOU. DOORS LEAD IN 4 D
IRECTIONS. "
650 GOTO 100
660 REM ROOM 3
662 RL=3
665 PRINTCHR$(147):PRINT"YOU ARE IN A SM
ALL,STUFFY"
670 PRINT"CHAMBER. THERE IS AN EXIT EAS
T."

```

```

671 IF F>7 AND W<2 THEN PRINT"YOU SEE A
CRYSTAL SWORD "
672 IF F<=7 AND W<2 THEN PRINT"YOU SEE A
ROUGH MACE "
675 GOTO 100
680 REM ROOM 4
682 RL=4
685 PRINTCHR$(147):PRINT"YOU ARE IN A BE
AUTIFUL"
687 PRINT"PARQUET BALLROOM WITH AN EXIT
WEST."
688 IF F<4 AND T<2 THEN PRINT"YOU SEE A
LARGE SPARKLING EMERALD."
689 IF F=>4 AND T<2 THEN PRINT"THERE IS
A BEAUTIFUL SILVER CHALICE."
690 GOTO 100
700 REM ROOM 5
702 RL=5
705 PRINTCHR$(147):PRINT"YOU FIND YOURSE
LF IN A"
710 PRINT"BRICK AMPHITHEATRE. THERE IS
AN EXIT NORTH"
715 PRINT:PRINT "YOU SEE A GOBLIN."
720 M=M+1
725 IF M=2 THEN PRINT"GET THE GOBLIN! QU
ICK!"
730 GOTO 100
800 REM ATTACK
805 IF F>7 THEN HP=HP+10
808 IF F<7 THEN HP=HP+5
810 IF F<4 THEN HP=HP+1
820 IF HP<5 AND HT<2THEN PRINT"HE HIT BA
CK!":HT=HT+1:GOTO 100
822 IF HP<5 AND HT>1 THEN PRINT"YOU ARE
DEAD!":GOTO 1050
823 IF HP=5 AND HT<2 THEN PRINT"HE HIT B
ACK!":HT=HT+1:GOTO 100
824 IF HP=5 AND HT>1 THEN PRINT"HE RAN A
WAY.":SC=SC+50:GOTO 100
825 IF HP>5 THEN PRINT"YOU GOT HIM!":SC=
SC+1000:GOTO 100

```

```

850 IF F=7 THEN PRINT"YOU FOUND THE TREA
SURE-YOU ARE RICH":GOTO 1070
855 IF F<7 THEN GOTO 660:
860 IF F>7 THEN GOTO 1050
900 REM GOT IT
902 IF RL=3 THEN W=W+1
905 IF RL=3 AND F>7 THEN SC=SC+500
910 IF RL=3 AND F<=7 THEN SC=SC+50
912 IF RL=4 AND F<4 THEN SC=SC+2000
913 IF RL=4 THEN T=T+1
914 IF RL=4 AND F=>4 THEN SC=SC+500
915 IFW>1 AND RL=3THENPRINT"YOU ALREADY
HAVE IT!":GOTO 945
916 IFT>1 AND RL=4THENPRINT"YOU ALREADY
HAVE IT!":GOTO 945
920 PRINT"GOT IT! "
945 RETURN
950 PRINT"I ONLY KNOW THESE WORDS:":PRIN
T:PRINT
960 PRINT"NORTH SOUTH EAST WEST "
965 PRINT"ATTACK FLEE GET":PRINT:PRINT
970 RETURN
1000 PRINT"YOU CAN'T GO THAT WAY!":PRINT
:PRINT
1010 RETURN
1050 REM END ROUTINE
1060 PRINT"YOU DIED WITH YOUR BOOTS ON A
ND A SCORE OF ";SC:GOTO 1080
1070 PRINT"YOU STRUT OUT WITH A SCORE OF
";SC
1080 END

```


Chapter 34

DANGER DUNGEON: GRAPHICS AND SOUND

Graphics

Although *Danger Dungeon* is a perfectly good game as far as it goes, it does not utilize all of the Commodore's capabilities. The BASIC program could have been written for any computer with a BASIC interpreter or compiler. The special features of the Commodore 64 allow the user to construct games using techniques which are not available on many computers, and which are only available through expensive hardware or software enhancements on most others. Specifically, the features most important in a cave game such as we have so far constructed in the last chapter are graphics and sound. Let's apply the interesting techniques we have practiced in earlier chapters to *Danger Dungeon*.

First, it would make the game more interesting if we created pictures of the monsters and the rooms where the monsters are located. The possibilities for making monsters are almost endless, and there is no right way to make them. Let your imagination be your guide. The sample program can be varied to produce any pictures once the method is understood.

The easiest way to create a picture is to define a subroutine. For example, if we insert line 633 after line 632, where we have


```

4010 PRINT"      JJJJKKKKK "
4020 PRINT"      JJJJJJKKKKKK "
4030 PRINT"      TTTTTTTTTT "
4040 PRINT"      G U" " "I H "
4045 PRINT"      T          H "
4050 PRINT"      T   Q   Q   H "
4055 PRINT"      T          H "
4060 PRINT"      T      KJ   H "
4070 PRINT"      6=T      JMK   H=5"
4080 PRINT"      T 99999999 H "
4090 PRINT"      T L: L: L: H "
4095 PRINT"      T Z          Y H "
4100 PRINT"      T MOP OP N H "
4110 PRINT"      M//888888//N "
4120 PRINT"      ##### "
4125 PRINT""

```

It is also possible to use DATA statements to draw the pictures, but PRINT statements are the most straightforward means and the easiest to comprehend. Try your hand at illustrating each of the rooms in your game. Don't forget to put a RETURN at the end of each subroutine. It is often convenient to place all of a series of related subroutines in the same area of a program. Designate lines 3000-8000 for those subroutines in your program.

3000 REM STONE FIGURE

```

3010 PRINT:PRINT
3020 PRINT"      ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ "
3030 PRINT"      "
3032 PRINT"      . . . . . "
3034 PRINT"      # # "
3040 PRINT"      "
3050 PRINT"      # # | | "
3060 PRINT"      "
3070 PRINT"      | | "
3080 PRINT"      | | "
3090 PRINT"      "
3100 PRINT"      . . "
3110 PRINT"      "
3120 PRINT"      # — — — — "
3130 PRINT"      — — — — "
3140 PRINT"      # # "

```



```

5520 PRINT"      z b z b      "
5530 PRINT"      o o o o      "
5540 PRINT"      o o o o      "
5550 PRINT"      o o o o      "
5560 PRINT"      o o o o      "
5570 PRINT"      o o o o      "
5580 PRINT"      v p v p      "
5585 PRINT:PRINT
5590 RETURN
6000 REM SPARKLING EMERALD
6010 PRINT:PRINT"{GRN}"
6015 PRINT"      "
6020 PRINT"      / / / /      "
6030 PRINT"      / / / /      "
6040 PRINT"      / / / /      "
6050 PRINT"      / / / /      "
6060 PRINT:PRINT"{BLK}"
6070 RETURN
6500 REM SILVER CHALICE
6510 PRINT:PRINT"{COM-5}"
6520 PRINT"      "
6530 PRINT"      "
6540 PRINT"      | 200000000000 |      "
6550 PRINT"      |  "      "
6560 PRINT"      |  "      "
6570 PRINT"      |  "      "
6580 PRINT"      / / / /      "
6590 PRINT"      / / / /      "
6600 PRINT"      "
6610 PRINT"      | 00000 |      "
6620 PRINT"      |  "      "
6630 PRINT"      |  "      "
6640 PRINT"      |  "      "
6650 PRINT"      |  "      "
6660 PRINT"      |  "      "
6670 PRINT:PRINT"{BLK}"
6680 RETURN
7000 REM AMPHITHEATRE+GOBLIN
7005 PRINT "{COM-2}"

```

```

7010 PRINT"
7020 PRINT"
7022 PRINT"
7024 PRINT"
7030 PRINT"
7040 PRINT"
7050 PRINT"
7060 PRINT"
7070 PRINT"
7080 PRINT"
7120 PRINT"
7140 PRINT"
7160 PRINT"
7170 PRINT"
7180 PRINT"
7190 PRINT"
7240 PRINT"{BLK}"
7250 FOR PAUSE=1 TO 1000:NEXT PAUSE:RETU
RN
7500 REM GOBLIN
7510 PRINT:PRINT
7512 PRINT"
7515 PRINT"
7516 PRINT"
7530 PRINT"
7540 PRINT"
7550 PRINT"
7560 PRINT"
7570 PRINT"
7580 PRINT"
7590 PRINT"
7600 PRINT"
7610 PRINT"
7620 PRINT"
7630 PRINT"
7640 PRINT"
7650 PRINT"
7660 PRINT"
7662 PRINT"{BLK}"

7730 PRINT:PRINT:RETURN

```

Type in these subroutines, then add the appropriate lines after each room description to send the program to each of the subroutines. The line numbers are:

```
633 GOSUB 4000
647 GOSUB 3000
671 GOSUB 5000
672 GOSUB 5500
688 GOSUB 6000
689 GOSUB 6500
710 GOSUB 7000
715 GOSUB 7500
```

Note that each GOSUB (lines 671, 672, 688, 689 and 710) is added after the description which is on the same line. (See the listing at the end of Chapter 33). Type the listing and see your pictures. If you can't wait to see how they look, the full program listing is shown at the end of this chapter.

Sound

As we saw in Chapter 24, the Commodore 64 has a Sound Interface Device (SID) which enables the programmer to produce just about any kind of sound effect. Songs can even be written into programs and played on cue, as we will see in *Scary Hall*, Chapter 35. For now, however, let's apply what we've learned about sound to make some appropriate sounds for *Danger Dungeon*.

The first thing that comes to mind is that monsters should make some sort of sound, or maybe there should be some sort of audible warning when a monster is about to get you. Often sound effects are used in movies to warn the hero of impending danger. Let's see if we can construct a sound to warn our player of some of the pitfalls we have put in the game.

The picture of the room-one monster is in subroutine 4000. We can add the lines to produce sound to the existing subroutine, taking

care to place the RETURN at the end of the entire subroutine, including the lines with the sound code. The first thing to do is to set the starting location of the sound chip by creating a variable to represent this location, and assign variable names to each of the memory locations for the tasks you wish to accomplish.

```
4130 V=54296:W1=54276:A=54277:H=54273:L=
54272
4140 POKEV,15:POKEW1,33:POKEA,15
4150 FORX=200 TO 5 STEP-2:POKEH,5:POKEL,
X:NEXT
4160 FORX=150 TO 5 STEP-2:POKEH,10:POKEL
,X:NEXT
4170 FORX=200 TO 5 STEP-2:POKEH,5:POKEL,
X:NEXT
4180 POKE W1,0:POKEA,0
4190 RETURN
```

After defining the variables (in line 4140 the volume is set at the highest level), the starting note is POKED and the waveform is selected. In lines 4150-4170 the high/low frequency is set, and the loop repeats the tone. Line 4180 turns off the sound.

Try setting up sound effects for other situations encountered during the game. For example, you might set up some sympathetic sound to be heard if the player "dies." Look at lines 1050-1140.

```
1050 REM END ROUTINE
1060 PRINT"YOU DIED WITH YOUR BOOTS ON A
ND A SCORE OF ";SC:GOTO 1080
1070 PRINT"YOU STRUT OUT WITH A SCORE OF
";SC
1080 V=54296:W1=54276:A=54277:H=54273:L=
54272
1090 POKEV,15:POKEW1,33:POKEA,15
1100 FORX=200 TO 5 STEP-2:POKEH,10:POKEL
,X:NEXT
1110 FORX=150 TO 5 STEP-2:POKEH,5 :POKEL
```



```
,X:NEXT
1120 FORX=200 TO 5 STEP-2:POKEH,2 :POKEL
,X:NEXT
1130 POKE W1,0:POKEA,0
1140 END
```

Here the end routine is enhanced with melancholy sounds. Again, these sounds are produced by modifying the monster sounds. The apparent sadness is related to the change in frequency in lines 1120 and 1130. The full game listing includes all of the enhancements discussed above which, you will find, result in an interesting game. Of course, there are many other ways to enhance the game. The most obvious way to increase its excitement is to create more rooms with more artifacts, monsters, and treasures. More pictures and sound effects would follow.

The game can be as good as you are willing to make it.

Danger Dungeon:Graphics and Sound

```
1 PRINTCHR$(147):POKE 53281,1:PRINTCHR$(
144)
10 REM SC=SCORE RL=ROOM LOC C$=COMMAND
20 REM F=RANDOM NUMBER HT=HIT FACTOR
30 REM HP=HIT POWER T=TREASURE
40 REM W=WEAPONS
50 RL=0:M=0:W=0:SC=0:HP=0:HT=0:T=0
60 F=INT(RND(1)*10)
70 PRINT"{CLR}"
80 GOTO 600
90 PRINT:PRINT:PRINT
100 INPUT"YOUR WISH IS MY COMMAND ";C$
110 IF LEFT$(C$,1)="N" THEN GOTO 300
120 IF LEFT$(C$,1)="S" THEN GOTO 350
130 IF LEFT$(C$,1)="E" THEN GOTO 400
140 IF LEFT$(C$,1)="W" THEN GOTO 450
150 IF LEFT$(C$,1)="A" THEN GOTO 800
160 IF LEFT$(C$,1)="F" THEN GOTO 850
```

```

165 IF LEFT$(C$,1)="G" THEN GOSUB 900:GO
TO 100
170 GOSUB 950:GOTO 100
300 REM NORTH
310 IF RL=0 THEN PRINT"YOU EXIT NORTH":G
OTO 1070
315 IF RL=1 THEN:GOTO 600
320 IF RL=2 THEN:GOTO 620
325 IF RL=3 THEN GOSUB 1000
330 IF RL=4 THEN GOSUB 1000
340 IF RL=5 THEN:GOTO 640
345 GOTO 100
350 REM SOUTH
355 IF RL=0 THEN:GOTO 620
360 IF RL=1 THEN:GOTO 640
365 IF RL=2 THEN:GOTO 700
370 IF RL=3 THEN GOSUB 1000
375 IF RL=4 THEN GOSUB 1000
380 IF RL=5 THEN GOSUB 1000
390 GOTO 100
400 REM EAST
410 IF RL=0 THEN GOSUB 1000
415 IF RL=1 THEN GOSUB 1000
420 IF RL=2 THEN:GOTO 680
425 IF RL=3 THEN:GOTO 640
430 IF RL=4 THEN GOSUB 1000
440 IF RL=5 THEN GOSUB 1000
445 GOTO 100
450 REM WEST
455 IF RL=0 THEN GOSUB 1000
460 IF RL=1 THEN GOSUB 1000
465 IF RL=5 THEN GOSUB 1000
470 IF RL=2 THEN:GOTO 660
472 IF RL=3 THEN GOSUB 1000
475 IF RL=4 THEN:GOTO 640
490 GOTO 100
600 REM ROOM 0
602 RL=0:PRINTCHR$(147)
604 PRINT"YOU ARE STANDING BEFORE A HEAV
Y"
605 PRINT"STONE DOOR. YOU MAY GO SOUTH "

```

```

610 PRINT"THROUGH THE DOOR OR NORTH BACK
"
615 PRINT"THE WAY YOU CAME."
616 GOTO 100
620 REM ROOM 1
622 RL=1
625 PRINTCHR$(147):PRINT"YOU ARE IN A ST
RANGE ROOM"
630 PRINT"WITH EXITS LEADING NORTH AND S
OUTH"
631 M=M+1
632 PRINT"LOOK OUT-MONSTER!"
633 GOSUB4000
635 GOTO 100
640 REM ROOM 2
641 RL=2
642 PRINTCHR$(147):PRINT"YOU ARE IN A LA
RGE ARCHED CHAMBER"
644 PRINT"WITH A STRANGE STONE FIGURE ST
ARING"
646 PRINT"DOWN AT YOU. DOORS LEAD IN 4 D
IRECTIONS."
647 GOSUB 3000
650 GOTO 100
660 REM ROOM 3
662 RL=3
665 PRINTCHR$(147):PRINT"YOU ARE IN A SM
ALL, STUFFY"
670 PRINT"CHAMBER.  THERE IS AN EXIT EAS
T."
671 IF F>7 AND W<2 THEN PRINT"YOU SEE A
CRYSTAL SWORD ":GOSUB 5000
672 IF F<=7 AND W<2 THEN PRINT"YOU SEE A
ROUGH MACE ":GOSUB 5500:POKE53281,1
675 GOTO 100
680 REM ROOM 4
682 RL=4
685 PRINTCHR$(147):PRINT"YOU ARE IN A BE
AUTIFUL"
687 PRINT"PARQUET BALLROOM WITH AN EXIT
WEST."

```

```

688 IF F<4 AND T<2 THEN PRINT"YOU SEE A
LARGE SPARKLING EMERALD.":GOSUB6000
689 IF F=>4 AND T<2 THEN PRINT"THERE IS
A BEAUTIFUL SILVER CHALICE.":GOSUB 6500
690 GOTO 100
700 REM ROOM 5
702 RL=5
705 PRINTCHR$(147):PRINT"YOU FIND YOURSE
LF IN A"
710 PRINT"BRICK AMPHITHEATRE.  THERE IS
AN EXIT  NORTH":GOSUB 7000
715 PRINT  "YOU SEE A GOBLIN.":GOSUB 750
0
720 M=M+1
725 IF M=2 THEN PRINT"GET THE GOBLIN! QU
ICK!"
730 GOTO 100
800 REM ATTACK
805 IF F>7THEN HP=HP+10
808 IF F=7THEN HP=HP+5
810 IF F<7THEN HP=HP+1
820 IF HP<5 AND HT<2THEN PRINT"HE HIT BA
CK!":HT=HT+1:GOTO 100
822 IF HP<5 AND HT>1 THEN PRINT"YOU ARE
DEAD!":GOTO 1050
823 IF HP=5 AND HT<2 THEN PRINT"HE HIT B
ACK!":HT=HT+1:GOTO 100
824 IF HP=5 AND HT>1 THEN PRINT"HE RAN A
WAY.":SC=SC+50:GOTO 100
825 IF HP>5 THEN PRINT"YOU GOT HIM!":SC=
SC+1000:GOTO 100
850 IF F=7 THEN PRINT"YOU FOUND THE TREA
SURE-YOU ARE RICH":GOTO 1070
855 IF F<7 THEN  GOTO 660:
860 IF F>7 THEN GOTO 1050
900 REM GOT IT
902 IF RL=3 THEN W=W+1
905 IF RL=3 AND F>7 THEN SC=SC+500
910 IF RL=3 AND F<=7 THEN SC=SC+50
912 IF RL=4 AND F<4 THEN SC=SC+2000
913 IF RL=4 THEN T=T+1

```

```

914 IF RL=4 AND F=>4 THEN SC=SC+500
915 IF W>1 AND RL=3THEN PRINT"YOU ALRE
ADY HAVE IT!":GOTO 945
916 IF T>1 AND RL=4THEN PRINT"YOU ALRE
ADY HAVE IT!":GOTO 945
920 PRINT"GOT IT! "
945 RETURN
950 PRINT"I ONLY KNOW THESE WORDS:":PRIN
T:PRINT
960 PRINT"NORTH SOUTH EAST WEST "
965 PRINT"ATTACK FLEE GET":PRINT:PRINT
970 RETURN
1000 PRINT"YOU CAN'T GO THAT WAY!":PRINT
:PRINT
1010 RETURN
1050 REM END ROUTINE
1060 PRINT"YOU DIED WITH YOUR BOOTS ON A
ND A SCORE OF ";SC:GOTO 1080
1070 PRINT"YOU STRUT OUT WITH A SCORE OF
";SC
1080 V=54296:W1=54276:A=54277:H=54273:L=
54272
1090 POKEV,15:POKEW1,33:POKEA,15
1100 FORX=200 TO 5 STEP-2:POKEH,10:POKEL
,X:NEXT
1110 FORX=150 TO 5 STEP-2:POKEH,5 :POKEL
,X:NEXT
1120 FORX=200 TO 5 STEP-2:POKEH,2 :POKEL
,X:NEXT
1130 POKE W1,0:POKEA,0
1140 END
3000 REM STONE FIGURE
3010 PRINT:PRINT
3020 PRINT" {#201}{#202}{#201}{
#202}{#201}{#202}{#201}{#202}{#201}{#202
}{#201}{#202}{#201}{#202}{#201}{#202}{#2
01}"
3030 PRINT" "
3032 PRINT" {#213} {#209}{#16
3}{#208} {#207}{#163}{#209}{#163}{#20
1} "

```

```

3034 PRINT"                {COM-8}  {#202}{#
203}                {#202}{#203}  "
3040 PRINT"                "
3050 PRINT"                {COM-8}                {#212
} {#217}                "
3060 PRINT"                {#212}  {#21
7}                "
3070 PRINT"                {#212}  {#21
7}                "
3080 PRINT"                {#203}  {#20
2}                "
3090 PRINT"                {#213}  {#2
01}                "
3100 PRINT"                {#202}{#209}
{#209}{#203}                "
3110 PRINT"                {#203}{#202
}                "
3120 PRINT"                {BLK}{#183 3}{#
202}{#192 2}{#203}{#183 3}  "
3130 PRINT"                {#163 2}{#202
}{#203}{#163 2}                "
3140 PRINT"                {COM-8}  {#213
}{#192 2}{#201}                "
3141 PRINT"{BLK}                "
3142 PRINT:PRINT
3150 RETURN
4000 REM MONSTER
4005 PRINT:PRINT
4010 PRINT"    {#202 5}{#203 5}  "
4020 PRINT"    {#202 6}{#203 6}  "
4030 PRINT"    {#212}{#208 10}{#200}  "
4040 PRINT"    {#199} {#213}{#162 2} {#1
62 2}{#201} {#200}  "
4045 PRINT"    {#212}                {#200}  "
4050 PRINT"    {#212}    {GRN}{#209}  {#20
9} {#200}  "
4055 PRINT"    {BLK}{#212}                {#200
}  "
4060 PRINT"    {#212}    {#203}{#202}
{#200}  "
4070 PRINT" {#182}={#212}    {#202}{#215

```

```

2} {#203}    {#200}={#181}"
4080 PRINT"    {#212} {RED}{#185 8} {#200
} "
4090 PRINT"    {#212} {#204}{#186} {#204}
{#186} {#204}{#186} {#200}  "
4095 PRINT"    {#212} {#165}          {#217}
{#200}  "
4100 PRINT"    {#212} {#205}{#207}{#208}
{#207}{#208} {#206} {#200}  "
4110 PRINT"    {#205}{#175 2}{RED}{#184 6
}{#175 2}{#206}  "
4120 PRINT"          {#163 7}  "
4125 PRINT" {BLK}"
4130 V=54296:W1=54276:A=54277:H=54273:L=
54272
4140 POKEV,15:POKEW1,33:POKEA,15
4150 FORX=200 TO 5 STEP-2:POKEH,5:POKEL,
X:NEXT
4160 FORX=150 TO 5 STEP-2:POKEH,10:POKEL
,X:NEXT
4170 FORX=200 TO 5 STEP-2:POKEH,5:POKEL,
X:NEXT
4180 POKE W1,0:POKEA,0
4190 RETURN
5000 REM    CRYSTAL SWORD
5010 PRINT:PRINT
5020 PRINT"          {#213}{#1
92 2} {#201}  "
5030 PRINT"          {#200} {
#199}  "
5032 PRINT"          {#186} {
#204}  "
5040 PRINT"    {#206}{#184 17}{#163 3}{#18
4}{#208}{#223 2}"
5050 PRINT"    {#205}{#162 16}{RED}{#215 4
} {#182}{#223}"
5060 PRINT"          {BLK} {#183
}{#208} {#207}{#183}  "
5070 PRINT"          {#200} {
#199}  "
5080 PRINT"          {#202}{#1

```

```

92 2}{#203} "
5085 PRINT"{BLK}"
5090 RETURN
5500 REM ROUGH MACE
5505 POKE 53281,2
5510 PRINT:PRINT
5520 PRINT"      {#213}{#221}{#201}{#213}
{#221}{#201}
"
5530 PRINT"      {#213}{#201}{#213}{#201}{
#213}{#201}{#213}{#201}
"
5540 PRINT"      {#213}{#201}{#213}{#201}{
#213}{#201}{#213}{#201}{#175 10}{#213}{#
201}"
5550 PRINT"      {#213}{#201}{#202}{#203}{
#202}{#203}{#202}{#203}
{#221}
"
5560 PRINT"      {#202}{#203}{#202}{#203}{
#202}{#203}{#202}{#203}{#183 10}{#202}{#
203}"
5570 PRINT"      {#202}{#203}{#202}{#203}{
#202}{#203}{#202}{#203}
"
5580 PRINT"      {#202}{#221}{#203}{#202}
{#221}{#203}
"
5585 PRINT:PRINT
5590 RETURN
6000 REM SPARKLING EMERALD
6010 PRINT:PRINT"{GRN}"
6015 PRINT"      {#175 4}
"
6020 PRINT"      ' ' {#206}{#205}{#175 2}{
#206}{#205} ' ' "
6030 PRINT"      ' ' ' {#205 2} {#206 2} ' '
"
6040 PRINT"      {#205 2}{#206 2}
"
6050 PRINT"      {#205}{#206}
"
6060 PRINT:PRINT"{BLK}"
6070 RETURN
6500 REM SILVER CHALICE
6510 PRINT:PRINT"{COM-5}"
6520 PRINT"      {#213}{#196 13}{#201}
"

```



```

6530 PRINT"      {#208}{#163 11}{#207}
"
6540 PRINT"      {#217}{BLU}{#209 10} {#
180}      "
6550 PRINT"      {COM-5} {#217}
{#180}      "
6560 PRINT"      {#202}{#192 11}{#203}
"
6570 PRINT"      {#163 11}      "
6580 PRINT"      {#205}      {#206}
"
6590 PRINT"      {#205}      {#206}
"
6600 PRINT"      {#213}{#163 5}{#201}
"
6610 PRINT"      {#221}{PUR}{#209 4}
{#221}      "
6620 PRINT"      {BLK}{#221}      {#22
1}      "
6630 PRINT"      {#202}{#192 5}{#203}
"
6640 PRINT"      {COM-5}      {#201} {#21
3}      "
6650 PRINT"      {#213}{#163 9}{#201}
"
6660 PRINT"      {#184 11}      "
6670 PRINT:PRINT" {BLK}"
6680 RETURN
7000 REM AMPHITHEATRE+GOBLIN
7005 PRINT "{COM-2}"
7010 PRINT"      {#207 2}{#208}{#207}{#208
}{#207}{#208}{#207}{#208}{#207}{#208}{#2
07}{#208}{#207}{#208}{#207}{#208}      "
7020 PRINT"      {#207}{#208}{#207}{#208}{#2
07}{#208}{#207}{#208}{#207}{#208}{#207}{
#208}{#207}{#208}{#207}{#208}{#207}{#208
}{#207}{#208 2}      "
7022 PRINT" {#207}{#208}{#207}{#208}{#20
7}{#208}{#207}{#208}{#207}{#208}{#207}{#
208}{#207}{#208}{#207}{#208}{#207}{#208}
{#207}{#208}{#207}{#208}{#207}{#208}      "

```

```

7024 PRINT" {#184}{#183 23} "
7030 PRINT" {#207}{#191 20}{#208}      "
7040 PRINT"{#207}{#183 25}{#208} "
7050 PRINT"{#181}{#206}{#183 23}{#205}{#
217} "
7060 PRINT"{#180 2}      {#213}*{#201}{#
213}*{#201}{#213}*{#201}{#213}*{#201}
{#170 2} "
7070 PRINT"{#180 2}      {#221} {#221 2}
{#221 2} {#221 2} {#221}      {#170 2} "
7080 PRINT"{#180 2}      {#221} {#221 2}
{#221 2} {#221 2} {#221}      {#170 2} "
7120 PRINT"{#180 2}      {#183 12}      (
#170 2} "
7140 PRINT"{#180 2}      {#184 16}      {#170
2} "
7160 PRINT"{#180 2}      {#202}{#192 14}{#
203}      {#170 2} ""
7170 PRINT"{#180 2}{#207}{#208}{#204}{#1
86}{#207}{#208}{#204}{#186}{#207}{#208}{
#204}{#186}{#207}{#208}{#204}{#186}{#207
}{#208}{#204}{#186}{#207}{#208}{#204}{#1
70 2} "
7180 PRINT"{#180 2}{#204}{#186}{#207}{#2
08}{#204}{#186}{#207}{#208}{#204}{#186}{
#207}{#208}{#204}{#186}{#207}{#208}{#204
}{#186}{#207}{#208}{#204}{#186}{#207}{#1
70 2} "
7190 PRINT"{#180 2}
{#170 2} "
7240 PRINT"{BLK}"
7250 FOR PAUSE=1 TO 1000:NEXT PAUSE:RETU
RN
7500 REM GOBLIN
7510 PRINT:PRINT
7512 PRINT"      {#202}{#201}{#203}{#202
}{#201}{#203}{#202}{#201}{#203}{#202}{#2
01}{#203}{#202}{#201}{#203}{#202}{#201}
"
7515 PRINT"      {#201}{#203}{#202}{#21
3}{#201}{#203}{#202}{#213}{#201}{#203}{#

```

```

202} {#213} {#201} {#203} "
7516 PRINT"          {#202} {#203} {#201} {#21
3} {#202} {#203} {#201} {#213} {#202} {#203} {#
201} {#213} {#202} {#203} {#201} "
7530 PRINT"          {#221}          {#221}
"
7540 PRINT"          {#221} {#213} {#201}
{#213} {#201} {#221} "
7550 PRINT"          {#221} {#209} {#209
} {#221} "
7560 PRINT"          {#221} {YEL} {#215}
{BLK} {#221} "
7570 PRINT"          {#221} {RED} {#213} {#1
92 5} {#201} {#221} "
7580 PRINT"          {#221} {RED} {#221} {#1
90 5} {#221} {BLK} {#221} "
7590 PRINT"          {#221} {RED} {#179} {#1
87 5} {#221} {#221} "
7600 PRINT"          {#221} {#183 6} {#22
1} "
7610 PRINT"          {#202} {#192 9} {#203}
"
7620 PRINT"          ) ( "
7630 PRINT"          {#213} {#166 7} {#201}
"
7640 PRINT"          {#221} {#221} {#22
1} {#221} "
7650 PRINT"          {#215 4} {#215 4}
"
7660 PRINT"          {#213} {#201}
"
7662 PRINT" {BLK} "
7670 V=54296:W2=54276:A=54277:H=54273:L=
54272
7680 POKEV,15:POKEW2,33:POKEA,15
7690 FORX=200 TO 5 STEP-2:POKEH,60:POKEL
,X:NEXT
7700 FORX=150 TO 5 STEP-2:POKEH,80:POKEL
,X:NEXT
7710 FORX=200 TO 5 STEP-2:POKEH,10:POKEL
,X:NEXT

```

```
7720 POKE W2,0:POKEA,0  
7730 PRINT:PRINT:RETURN
```

In the listing of *Danger Dungeon: Graphics and Sound*, the keyboard graphics characters which comprise the illustrations are represented by their ASCII equivalents (see Appendix F).

Chapter 35

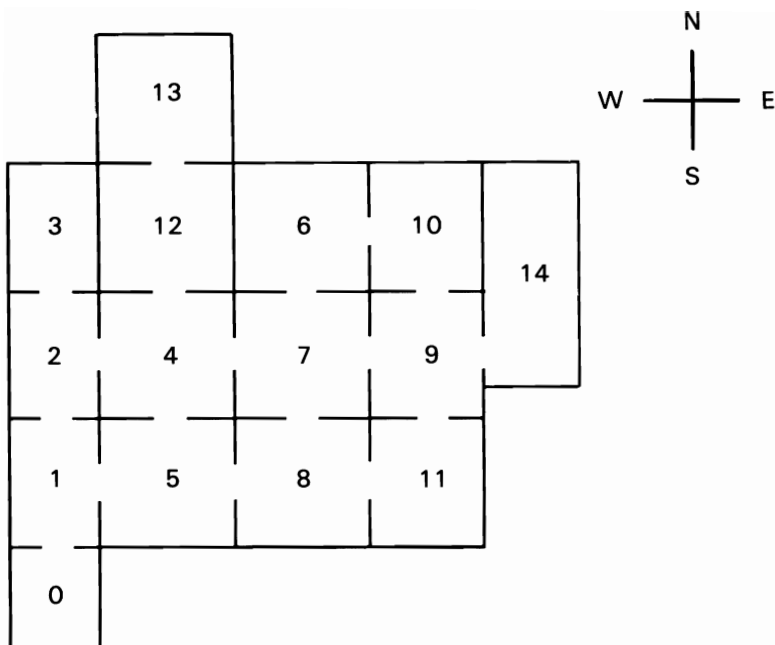
SCARY HALL

Scary Hall is another variation on the cave game. It differs from *Danger Dungeon* in that it is a detective game which asks the player to determine “who done it.” This means that the player can guess the murderer, find bodies and suspects, and view the several rooms in the game. Score is kept, and it is incremented whenever the player makes a valid room move. The score is decremented by one point whenever the player tries moving in an impossible direction, through a wall for example. When the player asks to identify the murderer (expressed by I or IDENTIFY) and is wrong, the score is decremented by 50 points. In addition, players can fall off the cliff, lose all of their points, be inundated by a tidal wave, enter the music room to hear a melody played by a ghost musician, or simply give up and exit the game.

Scary Hall is a long program with lots of subroutines and branching, but there’s still plenty of room for enhancements. Most of the techniques you learned in *Danger Dungeon* apply to *Scary Hall*. An important enhancement in *Scary Hall* is that the number of rooms has been increased to 15. You will find that the additional rooms make *Scary Hall* far more interesting than *Danger Dungeon*, but they also add more code. A new random number technique is introduced to place characters in each of the rooms. In this way characters may move around, whereas the *Danger Dungeon* monsters were placed in fixed-room locations.

Preprogramming

Just as we did with *Danger Dungeon*, we will construct a map of *Scary Hall* before we start programming.



And this time we are going to write the game in its final form, not in stages as we did with *Danger Dungeon*. That is, the graphics and sound will be created as we go along.

Again we will want to identify all of the variables at the beginning. Now that we have written *Danger Dungeon*, we are familiar with the techniques of planning the room subroutines, keeping track of the player's moves for scoring, and also placing graphics and sound routines for the rooms.

The room location variable will be RM. We will also need the following variables: (1) MI is a counter for misses in room moves; (2) RT is a counter for correct direction moves; (3) SC calculates the player's score; (4) DE keeps track of dead/alive status; (5) W is the random number generator; (6) W\$ stores the player's guess of the murderer; (7) SW is the counter for the amount subtracted from the SCORE when an incorrect guess is made; (8) N1 and N2 are random number selectors for the characters and their actions; (9) A\$ stores the character's action phrase; (10) NA\$ stores the character's description; (11) M\$ stores the player's INPUT; (12)

D1 to D9 and DA to DI are the variables that exclude a character who has died; (13) CL\$ stores the clue to the murderer's identity; (14) S, L, HF, A, LF, DR, and T are sound-routine variables; and (15) BA, TM, and N are bat-routine variables.

Now that we have identified the variables, the next task is to plan the room-description routines. Make a list of the rooms and assign line number ranges to each. Next to each room number and beginning line number, list valid moves (where doors are). The following chart lists all valid moves:

LINE	ROOM	VALID MOVES
4000	0	1
4100	1	0, 2, 5
4200	2	1, 3, 4
4300	3	2
4400	4	2, 5, 7, 12
4500	5	1, 4, 8
4600	6	7, 10
4700	7	4, 6, 8, 9
4800	8	5, 7, 11
4900	9	7, 10, 11, 14
5000	10	6, 9
5100	11	8, 9
5200	12	4, 13
5300	13	12
5400	14	9

Now let's take care of the other major portions of the program. We need a subroutine for the introduction, including (1) picture, (2) sound, and (3) statement. Keeping the line numbers easy to remember, we will number the subroutines 1000, 2000, and 2200. Because we don't want to come back to this area of the program, we will have a branch (GOTO) to 2500. This is how the program looks so far:

```

10 REM SCARY HALL
15 W=INT(RND(1)*17+1)

```

```
20 DE=0:MI=0:N2=0:N1=0:RT=0:SC=0:D1=0:D2
=0:D3=0:D4=0:D5=0
25 D6=0:D7=0:D8=0:SW=0
30 GOSUB 1000:GOSUB2000:GOSUB2200:GOTO 2
500
```

After having set aside space in the program for room descriptions and placing routines where they will be out of the way, we can focus on some problems at hand. This is a complex program, so we will go all out and make some bats appear on the screen. Subroutine 1000 generates keyboard characters which simulate flying bats. Then, in subroutine 2000, we see Scary Hall itself, in all its glory. In subroutine 2200 we hear a clock striking the twelfth hour. If our player likes scary settings, this should certainly do the trick. (Refer to the complete program listing at the end of this chapter to review subroutines and line ranges being discussed. Since *Scary Hall* is a very long program, the listings will not be duplicated during the discussion.)

Beginning the Game

Now, we are finally ready to begin the game. The entrance routine begins at line 2500, which is room zero. Look at lines 2500-2700. This part is much like the beginning of *Danger Dungeon*, except that additional commands are accepted. The player is allowed to Identify the murderer and ask for the SCore. Notice that the player must specify SO (rather than S) for SOut, because there are two commands beginning with S. Up and Down have also been added.

Now we need to set up the branching for room directions. Let's set up north to begin at line 3000, south to begin at line 3100, east at line 3200, and west at line 3300. It is good practice to allow plenty of space (large ranges of lines) for each subroutine or branch you plan. That way, you will not fence yourself in. For instance, we have many open areas in this program which we can use to set up other necessary routines, such as those that will identify the murderer, keep score, and describe characters and actions. The lesson here is that you should always set aside blocks of lines for tasks you know you will need later. If you are writing a complex program and you are not sure what you will need, it is doubly important to allow space.

The room-direction GOTOS follow the same logic as in *Danger Dungeon*. For example, if the player elects to go north, then lines 3020-3085 handle the contingencies of what room the player is actually in at the time of giving the N command. If the player tries to go north from rooms 6, 12, 13, or 14, subroutine 6000 PRINTs "YOU CAN'T GO THAT WAY" and increments the miss counter (MI) before RETURNing. When the player moves in a valid direction, the subroutine sends the program to the appropriate branch, and the player is found in the correct room.

By the way, you should RUN your program periodically to pick up obvious mistakes. You don't want a player to choose room two and end up in room three unless, of course, you planned it that way.

Some New Techniques

Now that we have taken care of the room directions, descriptions, and variables, we should consider writing routines for some of the new actions in this game. One of these new features is the IDENTIFY command and the identification of the murderer by the player. The identity of the murderer is determined by the random number selected at the start of the program, and it is up to the player to determine who committed the crime.

Of course, there must be victims, and there are two indicators of a character with problems. One states that the character is "LYING IN A POOL OF BLOOD," and the other states that a character is "STONE COLD DEAD." But only "STONE COLD DEAD" indicates that there has actually been a death. One could be lying in a pool of blood but only be playing dead.

Yet characters are murdered, and the player must determine who the murderer is. The CLue statements provided in lines 7500-7680 (CL\$) combine with the routines in lines 4144-4148, 4744-4748, and 4944-4948 to create a graphic representation of a note containing a clue to the murderer's identity. Lines 4144-4145, 4744-4745 and 4944-4945 announce, "THIS IS A CLUE!" and determine in which room the clue will be shown (depending on the value of random number variable W—which also identifies the murderer). The clue notes are placed in the rooms which have no other graphics. IF

W=4 THEN GOSUB 7500 is found in line 4145, which is the routine for room one. This means that if the murderer is a character with an ID number of one, two, or three (psychologist, scientist, or gardener), then the clues appear in this room. Clues will appear in room seven if the murderer is a monster, programmer, secretary, or hunchback. If the murderer is anyone else, the clues will appear in room nine.

When the player is ready to guess the identity of the murderer, he might want to use the IDENTIFY command. And if the player has already found several victims, then the guessing should be that much easier. If the guess turns out to be wrong, then the player loses 50 points (lines 3490-3491). If the guess is correct, "YOU DID IT" is displayed on-screen, and the program goes to the end routine at line 9000. In the event that a player with less than 50 points makes a bad guess, "AT THIS, A TIDAL WAVE INUNDATES YOU" appears, and the program ends. This routine is contained in lines 3400-3495.

The score-keeping function is located in lines 3500-3530. Line 3510 contains a formula to multiply the number of valid room moves by a factor of 10, multiply the number of room misses by the same factor of 10, and subtract the miss score from the valid score. The total points lost through bad guesses is subtracted from the prior score. The player's score is then displayed, and the command request is repeated by going to line 2500.

One of the new features presented in this game is the introduction and movement of characters throughout the game. Lines 7000-8400 contain the routines for these functions. Beginning in line 8010, the characters are introduced. Dr. Ratheart the Psychologist; Alec Trode, Mad Scientist; and Commodore Byte the Ex-Navy Computer Programmer are but a few of the startling characters. The variable N1 is a random number between 1 and 30, and this number determines whom your player will find in any given room. For example, IF N1=3 THEN NA\$="ZENIA HOTHOUSE, THE GARDENER. NA\$ is a created string variable associated with the random number 3 via a conditional statement. Any time the random number generator selects 3, Zenia Hothouse can be expected to make her appearance.

Next, lines 8200-8300 determine what action the character is undertaking in a given room. In line 7015, N2 is a random number between 1 and 9. The value of N2 is used to select one of the nine possible actions. Since, in line 8100, all numbers selected between 17 and 30 result in NOTHING ELSE being in the room, line 8215 must prevent any action that would not make sense. For example, if no one is in a room (because the random number was between 17 and 30) then it would not make sense to say that "NOTHING ELSE" was "STARING BACK AT YOU." Therefore, line 8215 causes the action variable (A\$) to be null or blank (A\$="").

Now, a real problem exists: what to do when a character has been found STONE COLD DEAD. If the character is really dead, then the poor soul should not be found alive in another room later in the same game. Thus, there must be a way to make a dead character stay dead. We can create a variable DE for DEad and set it equal to 0 for a live character. When a character dies, DE=1. This seems like a neat way to dispose of bodies, but unfortunately it's not that simple.

The problem is in making an association between the positive value of DEad (DE=1) and a specific character of NA\$. The best way to do it is to associate the variable DE with the random number of N1, which is in turn linked with NA\$. Line 8295 sets DE to N1 if N2=4 (STONE COLD DEAD). Then, line 8300 says that IF DE=1 THEN D1=1, line 8310 says IF DE=2 THEN D2=1, and so on, through line 8400. It is important that the value of DE be determined prior to the selection of a new random number under N1. So DE must contain the old value of N1 before it is replaced by the random number selected.

When the program goes back through subroutine 7000, the test is made for a positive and valid value of DEad (in the form of D1 to D9 and DA to DI) in lines 7020-7170. If any of the values representing characters show a character has died, then N2 takes on the value of 4 (STONE COLD DEAD) and associates that value with the current value of N1 or the new character. So when characters get killed, at least we will not find them running around Scary Hall, hale and hearty.

It goes without saying that you may vary any of the characters names, titles, and actions as you see fit. You might find that you are tired of Flo N. Pipes having hysterics. Just change the offending PRINT statements and run the program.

Graphics

All of the graphics in *Scary Hall* are comprised of keyboard characters (the pictures are shown in the program listing). The only color picture is the last, which is the cliff edge, room 14. Here the CTRL-5 was pressed to produce the purple hills. The green tree was produced by pressing CTRL-6. CTRL-1 made the color black. It is your challenge to improve upon the pictures included in our listing. You may alter the shapes simply by listing each line and making your changes. You need not change the line numbers but may use this listing as a framework for your new game.

Sound

The SID is used twice in *Scary Hall*. The first sound effect, a clock chiming twelve, is heard at the beginning of the game and conveys a sense of foreboding. The second, the theme from Beethoven's *Moonlight Sonata*, is heard in room eight (the music room), and creates a spooky atmosphere because no one appears to be sitting at the harpsichord.

Lines 2210-2290 comprise the chiming subroutine, which is based upon an example in the *Commodore 64 Programmer's Reference Guide*. Line 2200 clears the sound chip; S=54272 is the variable value for the starting memory location of the SID. Line 2230 sets the high frequency for voice one, line 2240 sets the attack/decay rate, line 2250 sets high frequency for voice three, and line 2260 sets highest volume. Lines 2270-2290 carry out the ring modulation which produces the clock-like chime. (Ring modulation combines oscillators to produce nonharmonic sounds like bells or gongs.) Line 2270 sets up a counter for the number of rings (12), sets up the triangle waveform, and combines with oscillator three. Lines 2280 and 2290 are timing loops, and line 2280 stops the triangle waveform.

The brief performance of the *Moonlight Sonata* is contained in lines 8500-8890. First, the sound registers are set as they were in other games that used sound (don't forget to clear the sound chip!). Lines 8520-8530 control attack/decay. Line 8540 sets the volume to loudest, line 8570 sets high and low frequencies, and lines 8580 and 8600 give us the pulse waveform. The actual melody is played by READING the DATA statements (lines 8620-8890).

The notes were read from sheet music and the chart contained in the *Commodore 64 Programmer's Reference Guide*, and the *Commodore 64 User's Guide* was consulted for the high/low frequency values. For example, in line 8620, the first two values (14 and 239) represent the high and low frequencies for A-sharp in the third octave. The third value (250) represents the duration of the note. (Here, 125 plays an eighth note, 250 plays a quarter note, 500 a half note, and 1000 a whole note.)

In order to cause a sound effect to occur at the appropriate time and place in a game, the best and easiest procedure is to make the SID starting memory location a variable (S=54272) and to build a subroutine around this variable. The routines already described in this and other chapters and in the *Commodore 64 Programmer's Reference Guide* and the *Commodore 64 User's Guide* should be a guide to technique and theory.

The subroutine you create can then be accessed by the program, and after the sound effect has been heard, the subroutine will RETURN to the starting place (GOSUB) in the program.

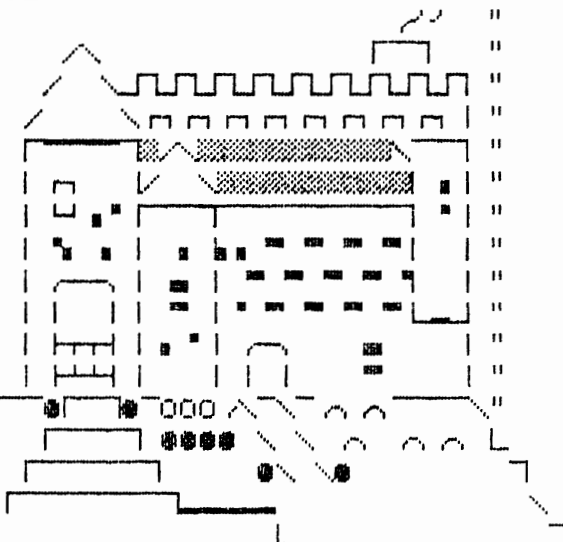
Scary Hall

```
1 PRINTCHR$(147):POKE 53281,1:PRINTCHR$(
144)
10 REM SCARY HALL
15 W=INT(RND(1)*17+1)
20 DE=0:MI=0:N2=0:N1=0:RT=0:SC=0:D1=0:D2
=0:D3=0:D4=0:D5=0
25 D6=0:D7=0:D8=0:SW=0
30 GOSUB 1000:GOSUB2000:GOSUB2200:GOTO 2
```

```

500
1000 FOR I=1 TO N :PRINT" {DOWN} {#223}{
#169}{#223}{#169} " :NEXT
1001 FOR BA=1 TO N
1004 FOR TM=1 TO N
1005 NEXT TM
1006 NEXT BA
1007 FOR BA=30 TO 1 STEP-1
1008 PRINT" {LEFT} {LEFT} {#223}{#1
69}{#223}{#169} {LEFT} ";
1009 FOR TM=1 TO30:NEXT TM:NEXT BA
1010 RETURN
2000 REM HOUSE
2011 PRINT"
2012 PRINT"
2015 PRINT"
2020 PRINT"
2030 PRINT"
2040 PRINT"
2050 PRINT"
2060 PRINT"
2070 PRINT"
2080 PRINT"
2090 PRINT"
2100 PRINT"
2110 PRINT"
2120 PRINT"
2130 PRINT"
2133 PRINT"
2134 PRINT"
2135 PRINT"
2136 PRINT"
2140 PRINT"THIS IS SCARY HALL.":
2145 PRINT"{RVS}ENTER AT YOUR OWN RISK!{
ROFF}"
2150 RETURN
2200 FOR PAUSE=1 TO 1000:NEXT PAUSE
2210 S=54272
2220 FOR C=0 TO 24:POKE S+C,0:NEXT
2230 POKES+1,130
2240 POKES+5,9

```



```

2250 POKES+15,30
2260 POKES+24,15
2270 FOR C=1 TO 12:POKE S+4,21
2280 FOR V=1 TO 1000:NEXT:POKE S+4,20
2290 FOR V=1 TO 1000:NEXT:NEXT
2300 RETURN
2400 PRINT"I ONLY KNOW THESE WORDS:" :PRINT:PRINT
2410 PRINT"NORTH, SOUTH, EAST, WEST, UP,
DOWN"
2420 PRINT"IDENTIFY, SCORE":PRINT:PRINT
2430 RETURN
2500 RM=0:REM DIRECTIONS
2502 RM=0
2505 PRINT"{CLR}"
2510 PRINT"YOU ARE STANDING BEFORE SCARY
HALL."
2515 PRINT"YOUR MISSION IS TO INVESTIGAT
E THE"
2520 PRINT"STRANGE SITUATION THERE. THE
RE HAS BEEN"
2522 PRINT"AN UNPLEASANT CRIME--MURDER!
WHEN"
2524 PRINT"YOU HAVE EVIDENCE, YOU MAY ID
ENTIFY"
2525 PRINT"THE CULPRIT. IF YOU ARE CORR
ECT, YOU"
2530 PRINT"WIN. YOU MAY ASK FOR YOUR SC
ORE AT"
2532 PRINT"ANY TIME. THE SUSPECTS ARE:
":PRINT:PRINT
2535 PRINT"1=PSYCHOLOGIST 2=SCIENTIST"
2540 PRINT"3=GARDENER 4=MONSTER 5=PROGRA
MMER"
2542 PRINT"6=SECRETARY 7=HUNCHBACK 8=MAG
ICIAN"
2544 PRINT"9=COOK 10=DOWAGER 11=INGENUUE"
2546 PRINT"12=GAME KEEPER 13=RECLUSE 14=
HANDYMAN"
2548 PRINT"15=ADMIN ASS'T 16=PLUMBER 17=
MAID":PRINT

```

```

2549 PRINT"GO NORTH TO ENTER. ":PRINT
2550 INPUT"YOU MAY MAKE THE NEXT MOVE.";
M$:PRINT
2560 IF LEFT$(M$,1)="N"THEN GOTO 3000
2570 IF LEFT$(M$,2)="SO"THEN GOTO 3100
2580 IF LEFT$(M$,2)="SC"THEN GOTO 3500
2590 IF LEFT$(M$,1)="E" THEN GOTO 3200
2600 IF LEFT$(M$,1)="W" THEN GOTO 3300
2610 IF LEFT$(M$,1)="I" THEN GOTO 3400
2620 IF LEFT$(M$,1)="U" THEN GOTO 3600
2630 IF LEFT$(M$,1)="D" THEN GOTO 3700
2700 GOSUB 2400:GOTO 2550
3000 REM ROOM MOVEMENTS
3010 REM NORTH
3020 IF RM=0 THENGOTO 4100
3030 IF RM=1 THEN GOTO 4200
3040 IF RM=2THEN GOTO 4300
3050 IF RM=3 ORRM=6 OR RM=12 OR RM=13 OR
RM=14OR RM=10THEN GOSUB 6000
3060 IF RM=4 THEN 5200
3065 IF RM=5 THEN 4400
3070 IF RM=7 THEN GOTO 4600
3075 IF RM=8 THEN GOTO 4700
3080 IF RM=9 THEN GOTO 5000
3085 IF RM=11THEN GOTO 4900
3090 GOTO 2550
3100 REM SOUTH
3110 IF RM=0 THEN PRINT"YOU GAVE UP SO S
OON.":GOTO 9000
3120 IF RM=1 THEN 2500
3130 IF RM=2 THEN 4100
3140 IF RM=3 THEN 4200
3145 IF RM=4 THEN 4500
3150 IF RM=5 OR RM=8 OR RM=11 OR RM=0 OR
RM=13 OR RM=14 THEN GOSUB6000
3155 IF RM=9 THEN 5100
3160 IF RM=6 THEN 4700
3170 IF RM=7 THEN 4800
3180 IF RM=10THEN 4900
3190 IF RM=12THEN 4400
3195 GOTO 2550

```



```

3200 REM EAST
3210 IF RM=1THEN GOTO 4500
3220 IF RM=2THEN GOTO 4400
3230 IF RM=4THEN GOTO 4700
3240 IF RM=5THEN GOTO 4800
3250 IF RM=6THEN GOTO 5000
3260 IF RM=7THEN GOTO 4900
3265 IF RM=8 THEN GOTO 5100
3270 IF RM=0 OR RM=3 OR RM=11 OR RM=12 O
R RM=13 THEN GOSUB 6000
3275 IF RM=9THEN GOTO 5400
3280 IF RM=14 THEN GOSUB 9500
3295 GOTO 2550
3300 REM WEST
3310 IF RM=4THEN 4200
3320 IF RM=5THEN 4100
3330 IF RM=7THEN 4400
3340 IF RM=8THEN 4500
3350 IF RM=9THEN 4700
3360 IF RM=10THEN 4600
3370 IF RM=14THEN 4900
3375 IF RM=11 THEN 4800
3380 IF RM=0 OR RM=2 OR RM=1 OR RM =3 OR
RM=12 OR RM =13ORRM=6 THEN GOSUB 6000
3390 GOTO 2550
3400 REM IDENTIFY
3410 PRINT"WHO DO YOU IDENTIFY?":PRINT
3415 PRINT"1=PSYCHOLOGIST 2=SCIENTIST"
3420 PRINT"3=GARDENER 4=MONSTER 5=PROGRA
MMER"
3430 PRINT"6=SECRETARY 7=HUNCHBACK 8=MAG
ICIAN"
3440 PRINT"9=COOK 10=DOWAGER 11=INGENUUE"
3450 PRINT"12=GAME KEEPER 13=RECLUSE 14=
HANDYMAN"
3460 PRINT"15=ADMIN ASS'T 16=FLUMBER 17=
MAID":PRINT
3480 INPUT"WHO? ";W$
3485 IF VAL(W$)=W THEN PRINT"YOU DID IT!
YOUR SCORE IS ";SC:GOTO 9000
3490 IF VAL(W$)<>W THEN SW=SW+50:SC=((RT

```

```

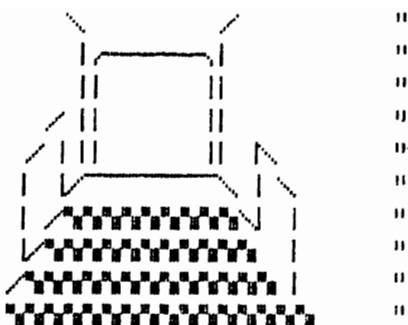
*10)-(MI*10))-SW
3491 IF SC>0 THEN PRINT"YOU LOSE 50 PTS.
    BETTER WATCH IT!":GOTO 2550
3492 IFSC<=0THEN PRINT"FORGET IT! YOU'RE
    ALL WET. AT THIS, A "
3495 PRINT"TIDAL WAVE INUNDATES YOU.":GO
    TO 9000
3500 REM SCORE
3510 SC=((RT*10)-(MI*10))-SW
3520 PRINT"YOUR SCORE IS ";SC
3530 GOTO 2550
3600 REM UP
3610 IF RM=12 THEN GOTO 5300
3620 IF RM<>12 THEN GOSUB 6000
3630 GOTO 2550
3700 REM DOWN
3710 IF RM=13 THEN GOTO 5200
3720 IF RM<13 ANDRM =>0 THEN GOSUB 6000
3730 IF RM=14 THEN GOSUB 9500
3740 GOTO 2550
4100 RM=1:PRINT"{CLR}": REM ROOM 1
4110 PRINT"YOU FIND YOURSELF IN A HUGE H
    ALL."
4120 PRINT"THERE ARE EXITS TO THE NORTH,
    SOUTH AND EAST.":PRINT:GOSUB 7000
4130 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
    ELSE"THEN GOTO 4150
4140 PRINT" WHO IS ";A$
4144 PRINT:PRINT"{RVS}THIS IS A CLUE!{RO
    FF}":PRINT:PRINT
4145 IF W<4 THEN GOSUB 7500
4146 PRINT"{#207}{#183 20}{#208}"
4147 PRINT CL$
4148 PRINT"{#204}{#175 20}{#186}"
4149 PRINT
4150 PRINT:PRINT:RT=RT+1:GOTO 2550
4200 RM=2:PRINT"{CLR}": REM ROOM 2
4210 PRINT"YOU ARE IN THE GREAT HALL. T
    HERE ARE"
4220 PRINT"EXITS TO THE NORTH, SOUTH AND
    EAST":GOSUB7000

```

```

4225 PRINT"
4230 PRINT"
4234 PRINT"
4235 PRINT"
4236 PRINT"
4237 PRINT"
4238 PRINT"
4239 PRINT"
4240 PRINT"
4250 PRINT"
4270 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4290
4280 PRINT" WHO IS ";A$
4290 PRINT:PRINT:RT=RT+1:GOTO 2550
4300 RM=3:PRINT"{CLR}":REM ROOM 3
4310 PRINT"YOU ARE IN THE PANTRY.  THERE
IS"
4320 PRINT"ONLY ONE EXIT TO THE SOUTH.":
GOSUB8000
4330 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4350
4340 PRINT" WHO IS ";A$
4350 PRINT:PRINT:RT=RT+1:GOTO 2550
4400 RM=4:PRINT"{CLR}":REM ROOM 4
4410 PRINT"YOU ARE AT A BEND IN THE LONG
HALL."
4420 PRINT"THERE IS A 4 WAY INTERSECTION
WITH"
4425 PRINT"DOORS LEADING IN ALL DIRECTIO
NS.":GOSUB 7000
4430 PRINT"
4440 PRINT"
4445 PRINT"
4446 PRINT"
4447 PRINT"
4448 PRINT:PRINT
4450 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4470
4460 PRINT" WHO IS ";A$
4470 PRINT:PRINT:RT=RT+1:GOTO 2550
4500 RM=5:PRINT"{CLR}":REM ROOM 5

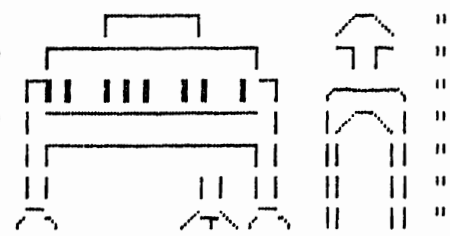
```




```

4725 PRINT"LEAD IN ALL 4 DIRECTIONS.  GO
OD LUCK.":GOSUB 7000
4730 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4750
4740 PRINT" WHO IS ";A$
4744 PRINT:PRINT"<RVS>THIS IS A CLUE!<RO
FF>":PRINT:PRINT
4745 IF W>3AND W<8 THEN GOSUB 7500
4746 PRINT"<#207><#183 20><#208>"
4747 PRINT CL$
4748 PRINT"<#204><#175 20><#186>"
4749 PRINT
4750 PRINT:PRINT:RT=RT+1:GOTO 2550
4800 RM=8:PRINT"<CLR>":REM ROOM8
4810 PRINT"YOU ARE IN THE MUSIC ROOM.  Y
OU GUESS"
4820 PRINT"THIS BECAUSE YOU HEAR A FAINT
MELODY."
4825 PRINT"IT HAS A GHOSTLY SOUND.":PRIN
T
4826 PRINT"
4830 PRINT"
4835 PRINT"
4840 PRINT"
4845 PRINT"
4850 PRINT"
4855 PRINT"
4860 PRINT"

```



```

4870 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4890
4880 PRINT" WHO IS ";A$
4890 PRINT:PRINT:RT=RT+1:GOTO 2550
4900 RM=9:PRINT"<CLR>":REM ROOM9
4910 PRINT"YOU ARE AT THE EAST END OF TH
E HALL."
4920 PRINT"THERE ARE EXITS TO THE NORTH,
SOUTH, WEST AND EAST.":GOSUB 7000
4930 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 4950
4940 PRINT" WHO IS ";A$
4944 PRINT:PRINT"<RVS>THIS IS A CLUE!<RO
FF>":PRINT:PRINT

```

```

4945 IF W>7ANDW <18 THEN GOSUB 7500
4946 PRINT"{#207}{#183 20}{#208}"
4947 PRINTCL$
4948 PRINT"{#204}{#175 20}{#186}"
4949 PRINT
4950 PRINT:PRINT:RT=RT+1:GOTO 2550
5000 RM=10:PRINT"{CLR}":REM ROOM 10
5010 PRINT"YOU STAND BEFORE A LONG, ELAB
ORATELY"
5020 PRINT"SET TABLE. THERE IS A LARGE
FIREPLACE"
5025 PRINT"IN THE CENTER OF THE ROOM. E
XITS TO THE WEST AND SOUTH."
5026 GOSUB 8000
5027 PRINT" "
5030 PRINT" "
5035 PRINT" "
5040 PRINT" "
5045 PRINT" "
5050 PRINT" "
5052 PRINT" "
5054 PRINT" "
5055 PRINT" "
5056 PRINT" "
5057 PRINT:PRINT
5060 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 5080
5070 PRINT" WHO IS ";A$
5080 PRINT:PRINT:RT=RT+1:GOTO 2550
5100 RM=11:PRINT"{CLR}":REM ROOM 11
5110 PRINT"YOU ARE IN A LARGE, WELL APPO
INTED"
5120 PRINT"BEDROOM. YOU SEE A MIRROR, B
UT IT"
5125 PRINT"IS NOT YOU LOOKING BACK. EXI
TS ARE"
5126 PRINT"TO THE WEST AND NORTH.":GOSUB
7000

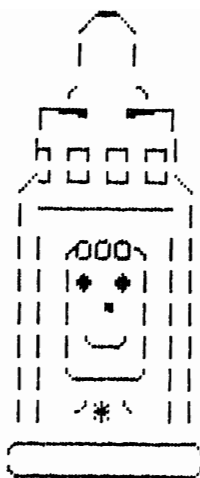
```



```

5130 PRINT"
5132 PRINT"
5135 PRINT"
5140 PRINT"
5145 PRINT"
5146 PRINT"
5150 PRINT"
5152 PRINT"
5153 PRINT"
5154 PRINT"
5155 PRINT"
5156 PRINT"
5157 PRINT"
5158 PRINT"
5159 PRINT"
5160 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 5180
5170 PRINT" WHO IS ";A$
5180 PRINT:PRINT:RT=RT+1:GOTO 2550
5200 RM=12:PRINT"{CLR}":REM ROOM 12
5210 PRINT"YOU STAND AT THE BOTTOM OF A
LONG FLIGHT"
5220 PRINT"OF STAIRS. YOU MAY GO UP, OR
SOUTH"
5225 PRINT"INTO THE HALL.":GOSUB 7000
5230 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 5250
5240 PRINT" WHO IS ";A$
5250 PRINT:PRINT:RT=RT+1:GOTO 2550
5300 RM=13:PRINT"{CLR}":REM ROOM 13
5310 PRINT"THE SUN STREAMS IN THE WINDOW
S. THE"
5320 PRINT"FOG SEEMS TO HAVE VANISHED.
YOUR CARES"
5325 PRINT"ALSO SEEM TO BE GONE. EXIT F
ROM THE"
5326 PRINT"SOLARIUM IS ONLY DOWN THE STA
IRS.":GOSUB 7000

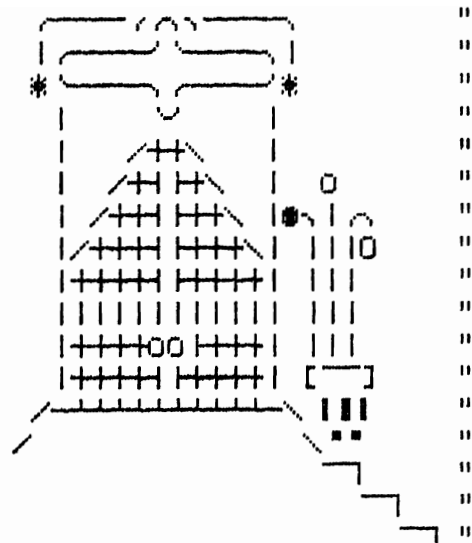
```



```

5330 PRINT"
5335 PRINT"
5338 PRINT"
5340 PRINT"
5345 PRINT"
5346 PRINT"
5347 PRINT"
5348 PRINT"
5350 PRINT"
5355 PRINT"
5356 PRINT"
5357 PRINT"
5358 PRINT"
5359 PRINT"
5360 PRINT"
5361 PRINT"
5362 PRINT"
5365 PRINT"YOU SEE ";NA$:IF NA$="NOTHING
ELSE"THEN GOTO 5380
5370 PRINT" WHO IS ";A$
5380 PRINT:PRINT:RT=RT+1:GOTO 2550
5400 RM=14:PRINT"{CLR}":REM ROOM 14
5410 PRINT"YOU ARE DELICATELY BALANCED U
PON THE"
5420 PRINT"PRECIPICE OF A VERY HIGH CLIF
F. EXITS"
5425 PRINT"ARE BEHIND YOU TO THE WEST AN
D BELOW"
5426 PRINT"YOU TO THE BOTTOM OF THE CLIF
F.":GOSUB 7000
5428 PRINT:PRINT

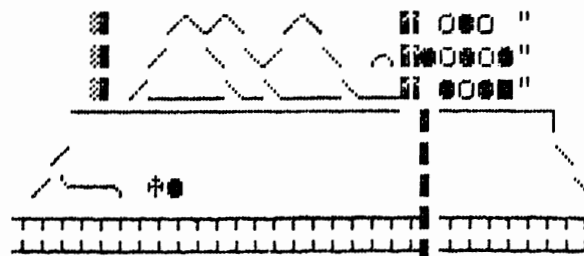
```



```

5430 PRINT"
5432 PRINT"
5434 PRINT"
5435 PRINT"
5440 PRINT"
5445 PRINT"
5446 PRINT"
5447 PRINT"
5450 PRINT:PRINT:RT=RT+1:GOTO 2550
6000 PRINT"YOU CAN'T GO THAT WAY!":PRINT

```




```

:PRINT
6010 MI=MI+1:RETURN
7000 REM TESTS FOR DEATH
7005 N2=0:N1=0
7010 N1=INT(RND(1)*30+1)
7015 N2=INT(RND(1)*9+1)
7020 IF D1>0 AND N1=1 THEN N2=4
7025 IF D2>0 AND N1=2 THEN N2=4
7030 IF D3>0 AND N1=3 THEN N2=4
7040 IF D4>0 AND N1=4 THEN N2=4
7050 IF D5>0 AND N1=5 THEN N2=4
7060 IF D6>0 AND N1=6 THEN N2=4
7070 IF D7>0 AND N1=7 THEN N2=4
7080 IF D8>0 AND N1=8 THEN N2=4
7090 IF D9>0 AND N1=9 THEN N2=4
7100 IFDA >0 AND N1=10THEN N2=4
7110 IFDB >0 AND N1=11THEN N2=4
7120 IFDC >0 AND N1=12THEN N2=4
7130 IFDD >0 AND N1=13THEN N2=4
7140 IFDF >0 AND N1=14THEN N2=4
7150 IFDG >0 AND N1=15THEN N2=4
7160 IFDH >0 AND N1=16THEN N2=4
7170 IFDI >0 AND N1=17THEN N2=4
7180 IF W=N1 AND N2=4 THEN GOTO 7000
7500 REM CLUES
7520 IF W=1 THEN CL$="PSYCHED OUT
"
7530 IF W=2 THEN CL$="SCIENCE IS ART
"
7540 IF W=3 THEN CL$="A ROSE IS A ROSE
"
7550 IF W=4 THEN CL$="FRANKIE, FRANKIE
"
7560 IF W=5 THEN CL$="A BIT OF A BYTE
"
7570 IF W=6 THEN CL$="MARRIES THE BOSS
"
7580 IF W=7 THEN CL$="OUR LADY
"
7590 IF W=8 THEN CL$="HOCUS POCUS
"

```

```

7600 IF W=9 THEN CL$="A RUSSE
"
7610 IF W=10 THEN CL$="LIKES HER SHERRY
"
7620 IF W=11 THEN CL$="THIS IS SCHILLY
"
7630 IF W=12 THEN CL$="I'M GAME, ARE YOU?
"
7640 IF W=13 THEN CL$="I WANT TO BE ALONE
"
7650 IF W=14 THEN CL$="THIS NEEDS WORK
"
7660 IF W=15 THEN CL$="FEEDING TIME
"
7670 IF W=16 THEN CL$="BODIES DRAINED
"
7680 IF W=17 THEN CL$="BOWL AND STORM
"

8000 REM NAMES
8010 IF N1=1 THEN NA$="DR RATHEART THE PSYCH
HOLOGIST"
8015 IF N1=2 THEN NA$="ALEC TRODE, MAD SCI
ENTIST"
8020 IF N1=3 THEN NA$="ZENIA HOTHOUSE, THE
GARDENER"
8025 IF N1=4 THEN NA$="FRANK FURTER, THE M
ONSTER"
8030 IF N1=5 THEN NA$="COMMODORE BYTE THE E
X-NAVY          COMPUTER PROGRAMMER"
8040 IF N1=6 THEN NA$="RITA WRIGHT THE SECR
ETARY"
8045 IF N1=7 THEN NA$="IGOR THE HUNCHBACK"
8050 IF N1=8 THEN NA$="MESMO RISE THE EVIL
MAGICIAN"
8055 IF N1=9 THEN NA$="CHARLOTTE MOUSSE, T
HE COOK"
8060 IF N1=10 THEN NA$="LADY DIMMEWITTE,
THE DOWAGER"
8065 IF N1=11 THEN NA$="SALLY SCHILLY THE
INGENUUE"
8070 IF N1=12 THEN NA$="FORREST RAVEN THE G

```

AME KEEPER"
 8075 IF N1=13 THEN NA\$="THELMA TOWER, THE
 RECLUSE"
 8080 IFN1=14 THENNA\$="MORTIMER ZURD THE
 HANDYMAN"
 8085 IF N1=15 THENNA\$="N D TROFF, THE A
 DMIN ASSISTANT"
 8090 IF N1=16THENNA\$="FLO N PIPES THE PL
 UMBER"
 8095 IF N1=17 THEN NA\$="DUSTY ROSE THE M
 AID"
 8100 IFN1>17ANDN1 <31THENNA\$="NOTHING E
 LSE"
 8200 REM ACTIONS
 8215 IF N1>17 ANDN1 <31 THEN A\$=" "
 8220 IFN2=1THENA\$ ="RIFFLING THROUGH A D
 RAWER"
 8230 IFN2=3THENA\$="RUNNING AWAY"
 8240 IFN2=5THENA\$="SWIGGING A BOTTLE OF
 BEER"
 8250 IF N2=2 THEN A\$="LYING IN A POOL OF
 BLOOD"
 8260 IF N2=4 THEN A\$="STONE COLD DEAD"
 8270 IF N1>17 ANDN1 <31 THEN A\$=" "
 8275 IF N2=6 THEN A\$="STARING BACK AT YO
 U"
 8280 IF N2=7 THEN A\$="WAVING A GUN"
 8285 IF N2=8 THEN A\$="HAVING HYSTERICS"
 8290 IF N2=9 THEN A\$="COMING TOWARD YOU"
 8295 IF N2=4 THEN DE=N1
 8300 IF DE=1 THEN D1=1
 8310 IF DE=2 THEN D2=1
 8315 IF DE=3 THEN D3=1
 8320 IF DE=4 THEN D4=1
 8325 IF DE=5 THEN D5=1
 8330 IF DE=6 THEN D6=1
 8335 IF DE=7 THEN D7=1
 8340 IF DE=8 THEN D8=1
 8345 IF DE=9 THEN D9=1
 8350 IF DE=10THEN DA=1
 8355 IF DE=11THEN DB=1

```

8360 IF DE=12THEN DC=1
8365 IF DE=13THEN DD=1
8370 IF DE=14THEN DF=1
8375 IF DE=15THEN DG=1
8380 IF DE=16THEN DH=1
8390 IF DE=17THEN DI=1
8395 N1=0
8400 RETURN
8500 S=54272
8510 FOR L=S TO S+24:POKEL,0:NEXT:REM CL
EAR SOUND CHIP
8520 POKE S+3,7 :POKE S+2,0
8530 POKE S+5,10 :POKES+6,0 :REM A=0 D=
0 S=0 R=0
8540 POKES+24,15:REM SET VOL TO MAX
8545 RESTORE
8550 READ HF,LF,DR
8560 IF HF <0THEN RETURN
8570 POKES+1,HF :POKES,LF
8580 POKES+4,65
8590 FOR T=1 TO DR:NEXT
8600 POKES+4,64 :FOR T=1 TO 50:NEXT
8610 GOTO 8550
8620 DATA 14 ,239,250,18,209 ,250
8630 DATA 22 ,96 ,250,14,239,250
8640 DATA 18,209,250,22 ,96 ,250
8650 DATA 14 ,239,250,19 ,239,250
8660 DATA 25,30 ,250,14,239,250
8670 DATA 19,239,250,25 ,30 ,250
8680 DATA 14 ,24 ,250,17 ,195,250
8690 DATA 25,30 ,250,14,24 ,250
8700 DATA 18,209,250,22,96 ,250
8710 DATA 12,143,250,16,195,250
8720 DATA 21,31 ,250,11,48 ,250
8730 DATA 15,210,250 ,18,209,250
8740 DATA 14,24 ,250,18,209,250
8750 DATA 22,96 ,250,14,24 ,250
8760 DATA 18,209,250,22,96 ,250
8770 DATA 14,24 ,250
8780 DATA 21,31,250,25,30,250
8790 DATA 14,24 ,250,21,31 ,250

```

```

8800 DATA 25,30,250,14,24 ,250
8810 DATA 18,209,250,22,96,250
8820 DATA 14,239,250,18,209,250
8830 DATA 22,96 ,250,15,210,250,21,31 ,2
50
8840 DATA 25,30 ,250,14,24 ,250
8850 DATA 16,195,250,22,96,250
8860 DATA 14,24 ,250,16,196,250
8870 DATA 22,96 ,250,33,135,500
8880 DATA 33,135,125 ,33,135,1000
8890 DATA -1,-1,-1
8900 RETURN
9000 REM END ROUTINE
9050 END
9500 PRINT"YOU FOOL! YOU JUMPED OFF THE
CLIFF.":GOTO 9000

```

The following line ranges show the ASCII equivalents (see Appendix F) of the keyboard graphics characters which comprise the illustrations in *Scary Hall*.

```

2011 PRINT"                                {#213}{#
203 2}  "
2012 PRINT"      {#206}{#205}
{#207}{#183}{#208}  "
2015 PRINT"      {#206}      {#205}{#186}{#208}
{#186}{#208}{#186}{#208}{#186}{#208}{#18
6}{#208}{#186}{#208}{#186}{#208}{#186}{#
208}{#186}{#208}  "
2020 PRINT"      {#206}      {#205}{#176}{#174
}{#176}{#174}{#176}{#174}{#176}{#174}{#1
76}{#174}{#176}{#174}{#176}{#174}{#176}{#
174}{#170}  "
2030 PRINT"      {#207}{#184 4}{#208}{#166}{#
#206}{#205}{#166 10}{#205}{#183 2}{#208}
"
2040 PRINT"      {#180}{#176}{#174}      {#170}{#
#206}      {#205}{#166 10}{#180}{#172}{#170}
"

```

2050 PRINT" {#180}{#173}{#189}{#172}{#188}{#170}{#183 3}{#208}{#163 10}{#180}{#188}{#170} "
 2060 PRINT" {#180}{#188}{#187} {#187}{#170} {#187}{#170}{#187 2}{#188}{#190}{#188}{#190}{#188}{#190}{#188}{#190}{#180}{#170} "
 2070 PRINT" {#180}{#213}{#192 2}{#201}{#170} {#172}{#187}{#170} {#188}{#190}{#188}{#190}{#188}{#190}{#188}{#190}{#188}{#190}{#188}{#180}{#170} "
 2080 PRINT" {#180}{#221} {#221}{#170} {#188}{#190}{#170} {#190}{#188}{#190}{#188}{#190}{#188}{#190}{#188}{#190}{#188}{#190}{#204}{#175}{#186} "
 2090 PRINT" {#180}{#171}{#178 2}{#179}{#170} {#187}{#188}{#170} {#213}{#192}{#201} {#172}{#187} {#170} "
 2100 PRINT" {#180}{#171}{#177 2}{#179}{#170} {#170} {#221} {#221} {#188}{#190} {#170} "
 2110 PRINT" {#183 3}{#209}{#207}{#183}{#208}{#209}{#183}{#215 3}{#213}{#205}{#183}{#205}{#183}{#213}{#201}{#213}{#201}{#183 4}{#205} "
 2120 PRINT" {#207}{#183 3}{#208} {#209 4} {#205} {#205} {#213}{#201} {#213}{#201}{#213}{#201} {#204} "
 2130 PRINT" {#207}{#183 5}{#208} {#209}{#205} {#205}{#209} {#208} "
 2133 PRINT" {#207}{#183 7}{#208}{#175 5} {#205} "
 2134 PRINT" {#204} {#183 2}{#208} "
 2135 PRINT" {#170} "
 2136 PRINT" {#170} "
 4225 PRINT" {#205} {#206} "
 "

4230 PRINT" {#167}{#213}{#192 5 }{#201}{#165} "
 4234 PRINT" {#167}{#221} {# 221}{#165} "
 4235 PRINT" {#206}{#167}{#221} {#221}{#165} "
 4236 PRINT" {#206}{#170}{#167}{#2 21} {#221}{#165}{#167}{#205} "
 4237 PRINT" {#170} {#167}{#206}{#1 83 7}{#205} {#180}{#205} "
 4238 PRINT" {#170} {#206}{#191 9}{# #205}{#180}{#170} "
 4239 PRINT" {#170}{#206}{#191 11} {#170} "
 4240 PRINT" {#206}{#191 13}{#170} "
 4250 PRINT" {#191 16} "

 4430 PRINT" {#206}{#183 10}"
 4440 PRINT" {#206} "
 4445 PRINT" {#183 9} "
 4446 PRINT" {# 206} "
 4447 PRINT" {#175 20}{#206} "

 4526 PRINT" {#207}{#183 22}{#208} "
 4530 PRINT" {#180}{#212}{#217}{#212}{#1 81}{#161}{#181}{#161 2}{#180}{#221 2}{#1 61}{#181}{#180 2}{#161 7}{#170} "
 4540 PRINT" {#180}{#163 22}{#170}{#161} "
 4550 PRINT" {#180}{#212 3}{#221}{#220}{# #166 2}{#181}{#161}{#181}{#161}{#181}{#1 61}{#181}{#161}{#181 2}{#161}{#181}{#161 2}{#217}{#170} "
 4560 PRINT" {#180 3}{#212 6}{#163 8}{#1 81}{#182 5}{#170} "
 4565 PRINT" {#180}{#162}{#184}{#162}{#1

```

84 2}{#182}{#161}{#182}{#161 3}{#182}{#1
81}{#180 2}{#181 2}{#180 2}{#181 2}{#180
}{#170}{#180}"
4566 PRINT" {#206}{#163 9}{#213}{#201}{#
163 13}{#205}"
4567 PRINT"{#206}          {#213}{#206}{#183
8}{#205}{#213}{#201}          {#205}"
4568 PRINT"          {#207}{#183 8}{#208}
"

4630 PRINT:PRINT" {#213}{#184 3}{#201}
      {#162 5}          "
4635 PRINT " {#221}          {#221}          {#161
}{#213}{#201} {#182}          {#209}{#178 2}{#2
09}          "
4640 PRINT " {#221} {#198}{#221}
{#180}{#163}{#210 3}{#208}{#210 9} "
4645 PRINT " {#221}          {#221}          {#180
}{#167} {#195}{#167}{#170} {#207}{#208}
{#207}{#208} {#170} "
4650 PRINT " {#221}          {#221} {#176}{#1
74} {#165} {#197 3}{#170} {#180}{#170}
{#180}{#170} {#170} "
4655 PRINT " {#197 28}"

4826 PRINT"          {#207}{#163 3}{#208}
      {#206}{#163}{#205}          "
4830 PRINT" {#207}{#183 9}{#208}          {#2
08} {#207}          "
4835 PRINT" {#207}{#181 2} {#181 3} {#18
1 2} {#181}{#208} {#213}{#192 3}{#201}
"
4840 PRINT" {#180}{#163 11}{#170} {#221
}{#206}{#183}{#205}{#221} "
4845 PRINT" {#180}{#207}{#183 9}{#208}{#
170} {#221}{#180} {#170}{#221} "
4850 PRINT" {#180 2}          {#180 2}{#170
2} {#221}{#180} {#170}{#221} "
4855 PRINT"{#213}{#183}{#201}          {#206
}{#178}{#205}{#213}{#183}{#201} {#221}{#
180} {#170}{#221} "

```



```

4860 PRINT" {#163 21} " :GOSUB 8500:GOSUB7
000

5027 PRINT"                {#175 6}                "
5030 PRINT"                {#206} {#218} {#2
18} {#205}                "
5035 PRINT"                {#206} {#221} {#18
0} {#221} {#205}                "
5040 PRINT"                {#206} {#175} {#215} {
#202} {#192 2} {#203} {#215} {#175} {#205}
"
5045 PRINT"                {#206} {#177} {#203
} {#202} {#177} {#205}                "
5050 PRINT"                {#206} {#183} {#202} {#2
03} {#202} {#203} {#183} {#205}                "
5052 PRINT"                {#206} {#189} {#173} {
#215} {#209} {#213} {#201} {#189} {#173} {
#205}                "
5054 PRINT"                {#206} {#202} {#192} {#203
} {#164} {#209} {#215} {#209} {#203} {#215} {
#164} {#202} {#192} {#203} {#205}                "
5055 PRINT"                {#206} {#221} {#205}
{#163 3} {#206} {#221} {#205}                "
5056 PRINT"                {#184 3} {#184 3}
{#184 3}                "

5130 PRINT"                {#206} {#184} {#205}
"
5132 PRINT"                {#180} {#170}
"
5135 PRINT"                {#213} {#201}
"
5140 PRINT"                {#207} {#184} {#190} {
#188} {#184} {#208}                "
5145 PRINT"                {#170} {#174} {#176} {#1
74} {#176} {#174} {#176} {#174} {#180}
"
5146 PRINT"                {#206} {#189} {#173} {#1
89} {#173} {#189} {#173} {#189} {#205}
"
5150 PRINT"                {#180} {#196 7} {#170}
"

```

5152 PRINT" {#180 2}{#213}{#215 3
 }{#201}{#170 2} "
 5153 PRINT" {#180 2}{#221}{#218}
 {#218}{#221}{#170 2} "
 5154 PRINT" {#180 2}{#221} {#188}
 {#221}{#170 2} "
 5155 PRINT" {#180 2}{#221}{#202}{
 #192}{#203}{#221}{#170 2} "
 5156 PRINT" {#180 2}{#202}{#192 3
 }{#203}{#170 2} "
 5157 PRINT" {#180 2} {#203}*{#202
 } {#170 2} "
 5158 PRINT" {#213}{#192 9}{#201}
 "
 5159 PRINT" {#202}{#192 9}{#203}
 "

 5330 PRINT" {#213}{#192 4}{#213 2}{#
 201 2}{#192 4}{#201} "
 5335 PRINT" {#221}{#213}{#192 4}{#20
 3}{#202}{#192 4}{#201}{#221} "
 5338 PRINT" *{#202}{#192 4}{#201}{#2
 13}{#192 4}{#203}* "
 5340 PRINT" {#221} {#202}{#203}
 {#221} "
 5345 PRINT" {#221} {#206}{#219 2}
 {#205} {#221} "
 5346 PRINT" {#221} {#206}{#219}{#1
 79}{#171}{#219}{#205} {#221} {#215}
 "
 5347 PRINT" {#221} {#206}{#219 2}{#
 179}{#171}{#219 2}{#205} {#221}{#209}{#2
 01}{#221}{#213}{#201} "
 5348 PRINT" {#221}{#206}{#219 3}{#1
 79}{#171}{#219 3}{#205}{#221} {#221 3}0
 "
 5350 PRINT" {#221}{#219 4}{#179}{#1
 71}{#219 4}{#221} {#221 3} "
 5355 PRINT" {#221 12} {#221 3}
 "
 "

5356 PRINT" {#221}{#219 4}{#215 2}{
 #171}{#219 3}{#221} {#221 3} "
 5357 PRINT" {#221}{#219 4}{#179}{#1
 71}{#219 4}{#221} [{#183 2} "
 5358 PRINT" {#206}{#192}{#177 10}{#1
 92}{#205} {#181}{#161}{#181} "
 5359 PRINT" {#206} {#205
 }{#188 2} "
 5360 PRINT" {#183}{#2
 08} "
 5361 PRINT" {#183}{
 #208} "
 5362 PRINT" {#183
 }{#208} "

 5430 PRINT" {PUR} {#206}{#205}{#20
 6}{#205} {#206}{#205} {GRN} {#215}{#
 209}{#215} "
 5432 PRINT" {PUR} {#206} {#205} {#
 205}{#206} {#205} {#213}{#201}{GRN}{#20
 9}{#215}{#209}{#215}{#209} "
 5434 PRINT" {PUR} {#206}{#164 4}{#20
 5}{#164}{#205}{#164 3}{#205}{#164 2}{GRN
 } {#209}{#215}{#209} "
 5435 PRINT" {BLK} {#163 17} {#161}{#16
 3 5}{#208} "
 5440 PRINT" {#206} {#
 161} {#205} "
 5445 PRINT" {#206}{#202}{#192 2}{#201}
 {#216}{#209} {#161} {#2
 05}"
 5446 PRINT" {#178 21}{#161}{#178 9}"
 5447 PRINT" {#177 21}{#161}{#177 9}"

Part 7

WRAP-UP

Chapter 36

IDEAS FOR THE FUTURE

There are many areas of game design and programming which we could not cover in sufficient detail in a book of this size. Some of the important areas that you may want to explore on your own are machine language, high-resolution graphics, and artificial intelligence.

Machine Language and Assemblers

To write computer games with many fast-moving screen objects, machine language is essential. As you can see from some of the games in this book, BASIC executes too slowly to support more than a couple dozen of space monsters zooming around at high speed. That doesn't mean that BASIC is not useful in game design, however. As you have learned, it is very useful.

Still, machine language programming is the way to go if speed is what you want. Actually, programmers use what are called assemblers. An assembler is a program which facilitates the creation of machine language programs. Machine language code consists of binary numbers, all 1s and 0s. Assemblers use symbols which aren't quite as understandable as BASIC commands, but they're a whole lot easier to use than strings of 1s and 0s. Some assembly language symbols are `JMP` (jump), `LDA` (load accumulator A), and `RTS` (return from subroutine). When the program is completed, it is "assembled" into machine language code, which is then saved to be run later.

High-Resolution Graphics

Another area for future study is high-resolution graphics. High resolution is very difficult to use from BASIC, so we have not paid it much attention. The demonstration program which follows will illustrate how slowly high-resolution mode works in BASIC. You can actually see each line being drawn and erased.

```
1 REM *****
2 REM *   HI-RES GRAPHICS DEMO   *
3 REM *   THIS PROGRAM SHIFTS TO *
4 REM *   HIGH RESOLUTION MODE AND *
5 REM *   DRAWS A DIAGONAL LINE   *
6 REM *   FROM UPPER LEFT TO LOWER *
7 REM *   RIGHT.  IT'S PRETTY SLOW *
8 REM *   IN BASIC, ISN'T IT?    *
9 REM *****
100 PRINT "{CLR}"
105 REM *****
106 REM *   SWITCH TO HI-RES   *
107 REM *****
110 POKE53265,PEEK(53265)OR32
120 POKE53272,PEEK(53272)OR 8
125 REM *****
126 REM *   CLEAR HI-RES SCREEN *
127 REM *           MEMORY      *
128 REM *****
130 FOR X=8192 TO 16191:POKEX,0:NEXT
140 FOR I=1024 TO 2023:POKEI,3:NEXT
150 X1=0:Y1=0:X2=319:Y2=199
160 DX=X2-X1:DY=Y2-Y1:X=X1:Y=Y1:L=SQR(DX
*DX+DY*DY)
170 XI=DX/L:YI=DY/L
180 FORZ=1TOL
190 GOSUB 230
200 X=X+XI:Y=Y+YI
210 NEXT
220 GOTO270
```



```

225 REM *****
226 REM *   LINES 230-250 PLOT A   *
227 REM *   POINT (PIXEL) ON THE  *
228 REM *       HI-RES SCREEN      *
229 REM *****
230 CH=INT(X/8):RO=INT(Y/8):LN=YAND7
240 BY=8192+RO*320+8*CH+LN:BI=7-(XAND7)
250 POKEBY,PEEK(BY)OR(2^BI)
260 RETURN
270 POKE1024,16
280 GETA$:IFA$<>" THEN280
290 POKE53265,PEEK(53265)AND223
300 POKE53272,PEEK(53272)AND247
310 CLR:PRINT" {CLR} ":END

```

Hi-res graphics is the ultimate in computer graphic realism, and it is worth the money to buy graphics software or hardware which makes high resolution practical. The lightpen is one such device which allows you to work easily in this mode.

Artificial Intelligence

Finally, we would like to mention briefly a fascinating area of programming which has attracted more and more attention lately: artificial intelligence. Artificial intelligence (AI) is crucial to many types of computer programs. Computers that play chess are good examples of AI programs.

Basically, artificial intelligence refers to the computer simulation of the human thought process. The term was first used by a group of M.I.T. scientists to refer to a computer's response to human questions, and in turn, the phrasing of questions on the part of the computer—or more accurately, the computer software. (Obviously, the interaction between a computer and a human subject is really the interaction between a computer program written by a human and another human who uses the software.)

One of the first uses of artificial intelligence with computers was with chess and checkers games. The computer simulates human

intelligence when it anticipates moves on the part of the human player. Of course, the computer can only anticipate chess moves (or anything else) because it is programmed to do so. There is nothing intrinsic in a computer which would give it a sense of its own identity, or as sociologists put it, the computer has no consciousness of its own consciousness. Perhaps it is more correct to say that the computer has no consciousness at all. The artificial intelligence, then, is really the intelligence of a computer programmer who anticipates another person's play, question, or response to a question. To do this, the programmer must plan the scenario for the interaction. This endeavor is not really much different than the games we have already constructed in this book.

An AI program like the chess game is developed by determining all possible moves or, at least, the majority of possible moves. For example, if the human player moves to a given position, the programmer must predetermine what the computer's move will be in such an instance, and act accordingly.

In the classical artificial intelligence game, the player (patient) asks the computer (psychiatrist) a question and receives an appropriate response. One of the first such games, *Eliza*, gives an initial prompt to the player then waits for a response. The player types in a statement or question. The computer then responds. The script might sound something like this:

ELIZA:	How are you today?
PLAYER:	I am not very well.
ELIZA:	Why are you not very well?
PLAYER:	Because I am worried.
ELIZA:	Why are you worried?
PLAYER:	Because someone is after me.
ELIZA:	Why is someone after you?
PLAYER:	Because they want to hurt me.
ELIZA:	Why does someone want to hurt you?
PLAYER:	Because they don't like me.
ELIZA:	Why don't they like you?
PLAYER:	Because they are from outer space.
ELIZA:	How long have you had these thoughts?

Get the drift? The program breaks the player's sentence into parts, which is called parsing; notes key words, matching these words with those in DATA statements; then calls a subroutine to make the appropriate response. The quality of the program depends upon the quality of the programmer and how complex he/she is willing to make the program. A program with a limited vocabulary is likely to make some stupid remarks. For example:

PLAYER: I think you're a son of a gun.
ELIZA: Tell me more about your family.

Here, the program picks up the key word, "son," and moves to the DATA statement where "son" (along with "daughter," "mother," "father," etc.) calls a subroutine which asks about the player's family.

*The programmer can write a program with such an extensive vocabulary and sophisticated parsing method that you could play the game for a very long time without knowing your partner was a computer. One experiment was carried out with players not knowing whether their partners were computers or people. They sat in front of computer screens and typed in their questions and answers on keyboards. Some of the players had people on the other end giving actual responses and questions, while others were actually using a computer program. The players were asked to guess whether they were dealing with a computer program or a person. When these answers were tabulated, it was clear that they were not able to tell whether they were interacting with a person or a machine.

The construction of such a game is beyond the scope of this book; however, AI programming would be a good topic for another book. It would be a real challenge to provide a way of incorporating the great Commodore 64 features in such a game, because these games do not extensively employ graphics or sound effects. Perhaps we could design a game with a bearded, cigar-smoking therapist who growls if he doesn't like your comments.

If this all sounds like shades of Dr. Frankenstein to you, please don't run in terror from your Commodore 64. It really is a very friendly machine.

Chapter 37

SUMMARY

Part One: Planning Your Game

In Chapter 1, we discussed the origin of games, the nature of games, and the types of games. The fact that most games can either be played or simulated on a computer was pointed out. Some questions game programmers must ask were listed. It was emphasized that the construction of a computer game must be planned before the programming begins.

The concept of game road mapping was introduced in Chapter 2. A game road map involves setting out the game's functions, which include: (1) the subject of the game, (2) the meaning of winning, (3) the method of working toward the goal, (4) the factors which must be remembered throughout the course of the game, (5) the decisions to be made by the player in the game, and (6) the decisions to be made by the computer during the game. The idea behind the road map is that the construction of a computer game requires a designing stage in which its creator maps out its essential parts.

Chapter 3 outlines the many powerful features of the Commodore 64, which are helpful in game construction. The sound synthesizer (SID), sprite graphics, keyboard graphics, and memory capacity are examples of these.

Part Two: Games with Words and Numbers

In Part Two, we took the plunge, discussing and creating games with numbers and games with words. Four games were presented

in this section: *What's My Number?*, *Take a Guess*, *LIGHTS! CAMERA! ACTION!*, and *Ask the Wizard*.

What's My Number?, *Take a Guess*, and *LIGHTS! CAMERA! ACTION!* are guessing games. *Ask the Wizard* is almost the opposite in that the player asks the computer questions and receives answers from the computer. These four games are all quite simple in concept, but are good starting points for beginning game programmers. Each introduces BASIC programming commands and concepts that are helpful to anyone interested in learning the BASIC language. Also, in each of these games, road mapping is clearly demonstrated.

Part Three: Putting Pictures into Your Games

In Part Three, we discussed techniques of adding pictures to your games. In this section, we introduced concepts which make games in this book unique to the Commodore 64.

Basic graphic tools are discussed in Chapter 10. Chapter 11 discusses keyboard graphics. Chapter 13 explains how to create special graphics characters, and in Chapters 14 and 15 we used keyboard graphics in two games: *Ask the Wizard: With Graphics* and *Magic Cards*. *Ask the Wizard: With Graphics* shows what a difference keyboard characters make in a game. In *Magic Cards* we saw how keyboard graphics were used very naturally to create images of playing cards.

Part Four: Advanced Game Tools and Techniques

In Chapter 17, we discussed why longer, more complex programs might necessitate the need for memory management of the C-64. In short programs which use no more than three sprites, there will probably be no need to relocate memory; however, if a program uses several sprites, customized characters, sound, and is longer than 10K, this chapter will help you shift memory and get your program to work.

In Chapter 18, animation techniques are presented. Chapter 19 also discusses animation, this time in terms of its use in games. The techniques used to move objects on the screen are explained, as is using animation when memory has been moved. In Chapter 20, joysticks and game paddles are explained.

Sprite graphics, complete with its fantastic capabilities and limitations, is presented in Chapter 21. These versatile graphic characters are a natural for Commodore 64 game construction. Chapter 22 shows the reader how to animate sprites, and in Chapter 23, sequential files are presented as an alternative means of storing sprite definition data.

Chapter 24 introduces the Sound Interface Device (SID), a music synthesizer which allows music and sound effect routines to be incorporated into games.

Part Five: Creating Action Games

In Part Five, the aim was to create action games in the BASIC language. Though most fast-action games are written in assembly languages, it is possible to create the “feel” of arcade games with BASIC. In Chapter 25, computer-simulation games are discussed, and we explain why discrete, rather than continuous, simulations are most feasible for Commodore 64 games.

Monster Maze, Chapter 26, is a run-for-your-life game. Players follow specific rules while moving quickly through the maze, trying to evade the monster. In *Interceptor*, a shoot-em-up game, we present a game of the arcade genre. An enemy airplane flies over a target field, and the defender must fire a missile mounted on a truck to intercept the airplane. When the plane is hit, the screen explodes in a frenzy of color—red, yellow, purple, and black.

Part Six: Strategy and Fantasy/ Adventure Games

In Part Six, we explored strategy and map games, as well as fantasy/adventure games. In *Rocket to the Green Planet*, the player

must fly a spaceship to a distant planet. *A Day at the Races* simulates horse race betting, complete with graphics and betting odds. *Goal to Go!* is a complex football game simulation which incorporates many of the features of the actual sport. Players can name teams and determine offensive and defensive tactics and plays. Sprites enhance the game by displaying football players at the line of scrimmage.

In fantasy/adventure games (role-playing games included), the programmer's imagination has free rein. These games are not simple guessing games or shoot-em-ups. They are designed to challenge the player who desires a more intensive involvement and interaction with the constructed environment. Although this type of game often has a "dungeon and dragons" flavor, it need not.

The fantasy/adventure game is essentially a map game, for the player must always make the decision to move in a specific direction (north, south, east, west, up, or down). There is usually a treasure hidden somewhere and monsters lurking everywhere. The player must elude or kill the monsters and recover the treasure.

The game is an unfolding story, with the player involved in the adventure, much as a medieval knight or fair maiden. One may have a similar experience reading a gothic or mystery novel, but the key difference in the fantasy/adventure game is that the player interacts with the computer program, restructuring the "plot" while playing. Only in computer gaming can the player determine the outcome of the adventure.

Danger Dungeon is a rather primitive form of the adventure game, although some of its features are fairly sophisticated. *Scary Hall* is a more fully developed detective adventure game, complete with graphic displays of rooms, music and sound effects, a murderer, victims, and outcomes based upon the skill and luck of the player-detective.

Part Seven: Wrap-Up

In Chapter 36, we discussed machine language programming and assembly languages, pointing out that BASIC was just too slow

for super high-speed games that use lots of sprites. We also provided a hi-res graphics program to demonstrate the speed limitations of programming games in BASIC. Finally, we touched upon the fascinating area of AI (artificial intelligence) programming, speculating that a computer like the Commodore 64—with its sound and graphics capabilities—might well be used to enhance any AI simulation.

A Final Word

In the *Commodore 64 Game Construction Kit*, we tried to supply you with a set of tools and techniques to use to create your own computer games. The first programs were simple guessing games using only numeric variables and simple BASIC concepts and commands. Gradually, we moved into the area of graphics and sound, while simultaneously introducing more complex games which fully exploit these features. We ran the gamut from the simplest number guessing game to some very complex simulation games, complete with sprite graphics, programmable characters, animation, and music.

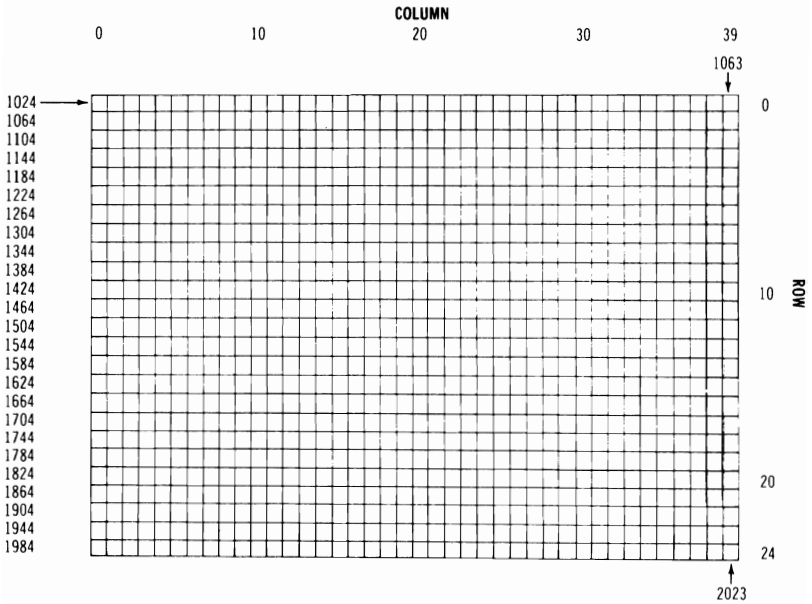
We tried to provide something for everyone and hope we included at least a couple of things for you. Our aim was to offer a comprehensive game construction kit for the Commodore 64 to allow you to realize the many potential benefits of owning that remarkable computer.

We hope our book was instructive and that it helps you enjoy your Commodore 64.

Appendix A

SCREEN MEMORY MAP

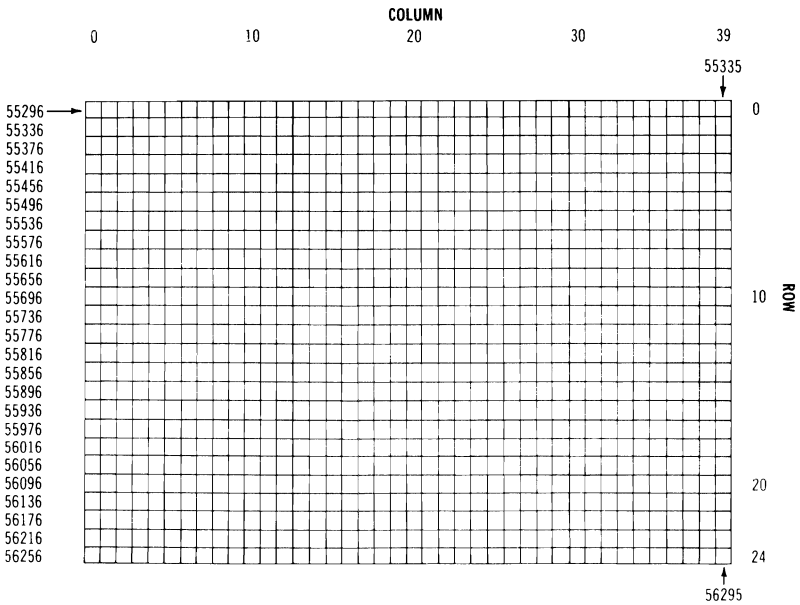
The following chart lists which memory locations control placing characters on the screen.



Appendix B

COLOR MEMORY MAP

The following chart lists which memory locations are used to change individual character colors.



Appendix C

CHARACTER COLOR CODES

The actual values to POKE into a color memory location to change a character's color are:

Ø	BLACK	8	ORANGE
1	WHITE	9	BROWN
2	RED	1Ø	Light RED
3	CYAN	11	GRAY 1
4	PURPLE	12	GRAY 2
5	GREEN	13	Light GREEN
6	BLUE	14	Light BLUE
7	YELLOW	15	GRAY 3

Appendix D

CUSTOM CHARACTER DRAWING GRID

Bit Number	7	6	5	4	3	2	1	0	
Bit Value	128	64	32	16	8	4	2	1	Byte Coding
Byte Number 0									_____
Byte Number 1									_____
Byte Number 2									_____
Byte Number 3									_____
Byte Number 4									_____
Byte Number 5									_____
Byte Number 6									_____
Byte Number 7									_____

Appendix E

SPRITE DRAWING GRID

Bit Value	1			1			1			Byte Coding							
	2	6	3	1	2	6	3	1	2		6	3	1				
	8	4	2	6	8	4	2	1	8	4	2	6	8	4	2	1	
Bytes 1 2 3																	---
Bytes 4 5 6																	---
Bytes 7 8 9																	---
Bytes 10 11 12																	---
Bytes 13 14 15																	---
Bytes 16 17 18																	---
Bytes 19 20 21																	---
Bytes 22 23 24																	---
Bytes 25 26 27																	---
Bytes 28 29 30																	---
Bytes 31 32 33																	---
Bytes 34 35 36																	---
Bytes 37 38 39																	---
Bytes 40 41 42																	---
Bytes 43 44 45																	---
Bytes 46 47 48																	---
Bytes 49 50 51																	---
Bytes 52 53 54																	---
Bytes 55 56 57																	---
Bytes 58 59 60																	---
Bytes 61 62 63																	---

Appendix F

COMMODORE 64 CHARACTER SETS

Screen Display Codes

This chart lists all of the characters built into the Commodore 64 character sets. Any character from Sets 1 and 2 cannot be displayed simultaneously, i.e., you cannot display characters from one set while characters from the other set are displayed. Also, any character may be displayed in REVERSE; the reverse character code is obtained by adding 128 to the value shown.



















SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	M	m	13	Z	z	26
A	a	1	N	n	14	[27
B	b	2	O	o	15	£		28
C	c	3	P	p	16]		29
D	d	4	Q	q	17	↑		30
E	e	5	R	r	18	←		31
F	f	6	S	s	19	SPACE		32
G	g	7	T	t	20	!		33
H	h	8	U	u	21	"		34
I	i	9	V	v	22	#		35
J	j	10	W	w	23	\$		36
K	k	11	X	x	24	%		37
L	l	12	Y	y	25	&		38

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
'		39		E	69			99
(40		F	70			100
)		41		G	71			101
*		42		H	72			102
+		43		I	73			103
,		44		J	74			104
-		45		K	75			105
.		46		L	76			106
/		47		M	77			107
0		48		N	78			108
1		49		O	79			109
2		50		P	80			110
3		51		Q	81			111
4		52		R	82			112
5		53		S	83			113
6		54		T	84			114
7		55		U	85			115
8		56		V	86			116
9		57		W	87			117
:		58		X	88			118
;		59		Y	89			119
<		60		Z	90			120
=		61			91			121
>		62			92			122
?		63			93			123
		64			94			124
	A	65			95			125
	B	66		SPACE	96			126
	C	67			97			127
	D	68			98			

Codes 128-255 are reverse images of codes 0-127

ASCII and CHR\$ Codes

This chart shows what characters will be displayed if you PRINT CHR\$(X), for all values of X. The chart also shows the values obtained by typing PRINT ("x"), where x is any character you can type.

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0		22	,	44	B	66
	1		23	-	45	C	67
	2		24	.	46	D	68
	3		25	/	47	E	69
	4		26	0	48	F	70
	5		27	1	49	G	71
	6		28	2	50	H	72
	7		29	3	51	I	73
DISABLES  	8		30	4	52	J	74
ENABLES  	9		31	5	53	K	75
	10		32	6	54	L	76
	11	!	33	7	55	M	77
	12	"	34	8	56	N	78
	13	#	35	9	57	O	79
	14	\$	36	:	58	P	80
	15	%	37	;	59	Q	81
	16	&	38		60	R	82
	17	.	39	=	61	S	83
 	18	(40		62	T	84
	19)	41	?	63	U	85
	20	*	42	@	64	V	86
	21	+	43	A	65	W	87

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
X	88	114	f1	f8	140	166	
Y	89	115		141		167	
Z	90	116		142		168	
[91	117			143	169	
£	92	118		144		170	
]	93	119		145		171	
↑	94	120		146		172	
←	95	121		147		173	
192	96	122		148		174	
193	97	123		149		175	
194	98	124		150		176	
195	99	125		151		177	
196	100	126		152		178	
197	101	127		153		179	
198	102		128	154		180	
199	103	129		155		181	
200	104		130	156		182	
201	105		131	157		183	
202	106		132	158		184	
203	107	f1	133	159		185	
204	108	f3	134	160		186	
205	109	f5	135	161	225	187	
206	110	f7	136	162	226	188	
207	111	f2	137	163	227	189	
208	112	f4	138	164	228	190	
209	113	f6	139	165	229	191	

Codes 192-223 are the same as 96-127.
Codes 224-254 are the same as 160-190.
Code 255 is the same as 126.

INDEX

A

Action games	13
Alphanumeric variables	33
Alternate character set	305
AND	57-58, 216-217
Animation	183-201
combined (moving and stationary)	183, 185
cross-screen	197
diagonal	197-198
horizontal	197, 199-200
moving	183-185
POKE	189-191
PRINT	185-189
stationary	183-184
vertical	197, 199-200
Arrays	28-32, 55-56, 58
general-purpose	58
multidimensional	31-32
one-dimensional	29
Artificial intelligence	415-417
ASCII	45, 437-438
codes	437-438
Assemblers	413
Assembly language	413

B

BASIC	22-23, 39-41, 174, 178, 422-423
language interpreter	174
programs, raising the start of	178
Binary numbers	106
Bit	21
Board games	13
Byte	21-22

C

Card dealing	287-289
CBM kernal operating system	174
Characters:	
codes	191
color codes	429
grid	104-105
custom	103
CHR\$	90-91, 437-438
codes	437-438

Color:	
and keyboard graphics	85-86
codes	190-191
codes, character	429
extended	79-80
graphics	118-119
memory map	427
mode	79-81
multi-	79-81
RAM	174
sprite	217-218
standard	79-80
Combined animation (moving and stationary)	183, 185
Computer simulations	247-249, 303, 421
continuous	247-249
discrete	247-249
Continuous simulations	247-249
Coordinates, X/Y	223-226
Coordinating sprite movement	265-267
Creating sequential files	235-236
Cross-screen animation	197
Custom characters	103-107, 109-111, 113, 431
digitizing	105
drawing grid	431

D

DATA	88-90, 93
Dealing cards	287-289
Designing sprites	212-213
Diagonal animation	197-198
Diagonal sprite movement	231-232
Digitizing data:	
custom characters	105
sprites	213
DIM	30-31
Disabling MSB register	230
Discrete simulations	247-249, 317-318
Draw-erase-redraw	185
Drawing grid:	
custom character	431
sprite	433

E

Efficient programming	47-48
Enabling MSB register	230
Envelope generators, SID	241
Error trapping	44-45, 66, 306-307
Expanding sprites	232-233
Extended color mode	79-80

F

Fantasy/adventure games	341, 422
Files, sequential	235-239
Fire button:	
joysticks	203, 207
game paddles	207
Flags	57-58
FOR/NEXT	30, 91-92
Free RAM area	174

G

Game modules	305, 319-320, 323
Game paddles	203, 207-209
fire button	207
Games:	
action	13
board	13
defined	13-14
fantasy/adventure	341, 422
map	277-289, 421-422
planning of	14-15
role playing	277, 341, 422
simplification of	304
simulation	14, 277-278, 421
strategy	277-289, 421-422
General-purpose arrays	58
GET	92
GOSUB	66
Graphics:	
characters, keyboard	83-86
color	118-119
high-resolution	79-80, 212, 414-415
keyboard	79
sprite	79-80, 211-221

H

Hardware	21-22
High-resolution graphics	79-80, 212, 414-415
Horizontal animation	197, 199-200
Horizontal sprite movement	227-230

I

IF/THEN	40
Illegal entries	56
Illegal moves	193-194

Input	22
INPUT	40, 51, 54
Input/Output	22
Input/Output registers	174
INT	34-35
with RND	35
Integer variables	28
I/O	22

J

Joysticks	203-209
fire button	203, 207

K

Keeping track of player position on map	280-284
Keyboard characters, custom	79
Keyboard graphics	79, 83-86, 117-119
and color	85-86
characters	83-86

L

LEFT\$	70
Legal moves	193-194
Line numbers	40
Loops	30
Loops, pause	47-48, 132

M

Machine language	22-23, 413
Map games	277-289, 421-422
Memory:	
address	173
banks	176
management	420
map, color	427
map, screen	425
organization	173, 181
reconfiguring	174-176, 181
screen	189
Microcomputer	21
Microprocessor:	
6510	21, 23, 176-177
VIC-II	176-177

MOB	211
Modules, game	305, 319-320, 323
Most Significant Bit register	224-226
Moveable Object Block	211
Moves, legal/illegal	193-194
Moving animation	183-185
Moving screen memory	176-178
MSB	224-226, 230-231
ON/OFF	231
register	224-226
register, enabling	230
register, disabling	230
Multicolor mode	79-81
Multidimensional arrays	31-32

N

Nested subroutines	132
Numeric variables	27-28

O

One-dimensional arrays	29
One-time routines, placement of	252
Operating system memory	174
OR	57, 216-217
Output	22

P

Pause loops	47-48, 132
PEEK	88, 92-93
Planning games	14-15
Player position on map, keeping track of	280-284
Pointers	174
sprite definition	215
POKE	83-85, 88, 93
animation	189-191
Positioning sprites	223-227
PRINT	40, 83-85, 91, 185-189
animation	185-189
with FOR/NEXT	91
Probability	34, 36-37, 325
Prompt	51, 54

R

Raising the start of BASIC programs	178
RAM	173-174
Random Access Memory	173-174
Random numbers	32-34, 36-37
READ	88-90, 93
Read Only Memory	103
Reading sequential files	236-237
Reconfiguring memory	174-176, 181
Registers	174, 224-226
MSB	224-226
sprite position	224
REM	40-41
Representing units of time	284-285
RESTORE	90
RETURN	66
Ring modulation, SID	241
RND	32-34, 36-37
Road maps	17-20, 419
Role playing games	277, 341, 422
ROM	103

S

Saving screen data	238-239
Screen:	
display codes	435-436
editor	131
memory	174, 176, 189
moving screen memory	176
memory map	425
visible viewing area	224
Sequential files	235-239
creating	235-236
reading	236-237
SID	241-244, 365, 386-387
envelope generators	241
ring modulation	241
voices	241
waveforms	241
Simplifying games	304, 421
Simulations	14, 247-249, 277-278, 317-318
computer	247-249
continuous	247-249
discrete	247-249, 317-318
strategic	317
6510 microprocessor	21, 23, 173, 176-177
Software	22
Sound effects	295-296
Sound Interface Device	241-244, 365
Special RAM area	174

Sprites:	
color	217-218
data	213-214
data page	214
definition pointers	215
designing	212-213
digitizing	213
drawing grid	433
expansion	232-233
graphics	79-80, 211-221
movement	227-232, 265-267
coordination of	265-267
diagonal	231-232
horizontal	227-230
vertical	227-230
pointers	174
positioning	223
position registers	224
storing data	213-214
turning ON/OFF	216-217
Standard character set	79
Standard color mode	79-80
Stationary animation	183-184
Storing sprite data	213-214
Strategic simulations	317
Strategy games	277-289, 421-422
String variables	33, 51, 53-54
Subroutines	47, 55, 66, 132, 252
nested	132
placement of	252

T

Tasks within time limits	285-287
Time, representing units of	284-285
Turning sprites ON/OFF	216-217
Turning the MSB ON/OFF	230

V

VAL	44-45, 66, 306-307, 324
Variables:	
alphanumeric	33, 51-54
arrays	28-32
integer	28
names	27
numeric	27, 28
string	33, 51-54
Vertical animation	197-200
Vertical sprite movement	227-230
VIC-II	176-177

registers	174
Visible viewing area of screen	224
Voice, SID	241

W

Waveform, SID	241
---------------------	-----

X

X,Y coordinates	223-226
-----------------------	---------

COMMODORE 64* GAME CONSTRUCTION KIT

On disk!

Want to get the most out of the COMMODORE 64 GAME CONSTRUCTION KIT?

For just \$15.95, you can have all the great games and useful utilities you read about in this book — on one diskette! Every program on the diskette is listable, copyable, and contains REM statements that document each part of the program.

Please RUSH me the COMMODORE 64 GAME CONSTRUCTION KIT diskette.

Send \$15.95 (California residents include 6.5% sales tax) plus \$2 shipping and handling to :

 **DATAMOST**

20660 Nordhoff St., Chatsworth, CA 91311-6152 (818) 709-1202

NAME _____

ADDRESS _____

CITY _____ ZIP _____ STATE _____

PHONE () _____

*Commodore 64 is a trademark of Commodore 64 Business Machines, Inc.



Other Great DATAMOST Books

KIDS AND THE COMMODORE 64 by *Edward H. Carlson*

Thirty-three lessons introduce your child to computers. Each lesson begins with introductory notes for teachers and parents, and ends with review questions. Cleverly illustrated, **KIDS AND THE COMMODORE 64** is great for adults too! \$19.95

THE ELEMENTARY COMMODORE 64 by *William B. Sanders*

Leads you step-by-step through the process of hooking up your C-64, loading and saving programs, creating graphics and sound, using handy utilities, even writing your own programs. Even if you know nothing about computers, this book will help you join the computer revolution. \$14.95

COMPUTER PLAYGROUND ON THE COMMODORE 64 AND VIC-20 by *M.J. Winter*

It's BASIC programming time for kids, grades two through six! Here is a collection of programming activities for the Commodore. Each activity is presented as a problem, with questions leading the child to the correct solution. Simply written and colorfully illustrated, **COMPUTER PLAYGROUND ON THE COMMODORE 64 AND VIC-20** is sure to be a hit with your child. \$9.95

COMMODORE 64 LOGO WORKBOOK by *M.J. Winter*

This great book shows two to six graders how LOGO can be used to solve problems. Kids will learn about "turtles," visual problem solving, variables, geometry, and recursion. Using graphics and words, the **LOGO WORKBOOK** provides a powerful learning experience while teaching LOGO. \$12.95

THE SUPER COMPUTER SNOOPER: COMMODORE 64 by *Dr. Isaac Malitz*
THE SUPER COMPUTER SNOOPER traces the path of a character from the keyboard through memory to disk, screen and printer. You'll learn how to restore programs that have been accidentally erased, to "listen" to the inner workings of the C-64, and to identify deleted or hidden disk files. \$14.95

THE MUSICAL COMMODORE 64 by *Hal Glicksman with Laura Goodfriend*
THE MUSICAL COMMODORE 64 will introduce you to music theory and computing. Written for beginners as well as pros, this book includes BASIC programs that turn the C-64 into a musical instrument. Non-musicians and non-programmers will learn to play music and create sound effects on the Commodore. \$14.95

INSIDE COMMODORE DOS by *Richard Immers and Gerald G. Neufeld*
INSIDE COMMODORE DOS is all about using DOS and the 1541 disk drive. Disk formatting, disk organization, direct-access programming, DOS protection, and recovering damaged data are discussed. It also includes an overview of DOS 2.6, as well as a detailed disk RAM map, an in-depth analysis of ROM, and mathematical conversion routines. \$19.95

Find these great books at your local computer dealer or bookstore.
Or order direct from:



20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202

\$14.95

COMMODORE 64 GAME CONSTRUCTION KIT

*This Book Will Teach You
How to Program Your Own Games!*

The COMMODORE 64 GAME CONSTRUCTION KIT was written for those of you who enjoy playing computer games and are excited by the idea of creating your own! You'll learn about text games, text games with graphics and sound, and games which are primarily graphics oriented.

You don't have to be an advanced programmer to use the CONSTRUCTION KIT, but any BASIC you do know will come in handy. Each chapter has been written in a clear, easy-to-understand style, so you can progress at your own rate.

The book begins with examples of simple games, illustrating the fundamentals of game programming. Later chapters explain basic aspects of graphics and sound, and how these can be added to your games. Gradually the book explores more sophisticated programming techniques.

So if you enjoy computer games and can't wait to write some of your own, the COMMODORE 64 GAME CONSTRUCTION KIT is just what you need to get started.



ISBN 0-88190-293-4

 **DATAMOST**[™]
INC

20660 Nordhoff Street, Chatsworth, CA 91311-6152
(818) 709-1202