COMPUTE!'s FIRST BOOK OF

COMMODORE 64 GAMES

19 games for the Commodore 64™ home computer, ready to type in and enjoy. Unpublished games and the best from COMPUTE! and COMPUTE!'s Gazette in machine language and BASIC.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s FIRST BOOK OF COMMODORE 64 GAMES

COMPUTE!™ Publications, Inc. abc
One of the ABC Publishing Companies
Greensboro, North Carolina

The following article was originally published in *COMPUTE!* Magazine, copyright 1981, Small System Services, Inc.:
"Maze Generator" (December).

The following articles were originally published in *COMPUTE!* Magazine, copyright 1982, Small System Services, Inc.:
"Writing Your First Game" (October),
"Laser Gunner" (November).

The following articles were originally published in *COMPUTE!* Magazine, copyright 1983, COMPUTE! Publications, Inc.:
"Writing an Arcade Game" (February),
"Minefield" (June),
"The Hawkmen Of Dindrin" (June),
"Goblin" (July),
"Rats" (July),
"Writing a Simulation Game" (July),
"Blockhead" (August),
"Diamond Drop" (September),
"Mystery Spell" (September),
"High Speed Mazer" (October),
"Mosaic Puzzle" (October).

The following articles were originally published in *COMPUTE's Gazette* Magazine, copyright 1983, COMPUTE! Publications, Inc.:
"Adding Joysticks To Your Games" (July),
"Snake Escape" (July),
"Cylon Zap" (August),
"The Viper" (August),
"States & Capitals Tutor" (September),
"Oil Tycoon" (October).

Printed in the United States of America

ISBN 0-942386-34-5

10 9 8 7 6 5 4 3 2

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is a subsidiary of American Broadcasting Companies, Inc., and is not associated with any manufacturer of personal computers. Commodore 64 is a trademark of Commodore Electronics Limited.

Contents

Foreword

COMPUTE!'s First Book of Commodore 64 Games is packed full of great games. But this book serves a double purpose.

First, it provides you with a variety of games, which you can merely type into the computer, save on disk or tape, and then play again and again.

Second, because the full program is here in print, you can see exactly how the game's creator brought off the effects you like.

In fact, to make this book as useful as possible, many of the games are accompanied by explanations of how the program works. Chapters at the beginning and end of the book will also help you learn how to write your own games.

In order to make typing in the programs as easy as possible, we have included three aids. Be sure to read over the article in Appendix A "Beginner's Guide to Typing in Programs." Also, review Appendix B "How to Type in Programs."

A number of the programs are written completely or partially in machine language. If you have ever typed in a machine language program with its hundreds of DATA statements, you will appreciate the "Machine Language Editor (MLX)" in Chapter 6. MLX is a BASIC program that will help you type in machine language programs perfectly the first time.

Chapter 1

The 64 as a Game Machine

Why the Commodore 64 Is a Great Game Machine

Eric Brandon

One of the first things a new programmer wants to do is write a game. The programmer soon discovers that there is no "move alien around" command; rather, the computer must be told what to do in hundreds of tiny little steps.

Fortunately, the Commodore 64 is loaded with features that make this arduous task much easier and reduce the number of steps that have to be programmed into the computer. The games in this book try to exploit these features as much as possible, to save the programmer time, and to save you typing.

Parlez-vous BASIC?

What language to program the game in is the first decision the programmer must make. On the Commodore 64 the choice is between BASIC and machine language.

The native language of the computer is machine language. This means that programs written in BASIC have to be translated into machine language while they are running. That translation takes time, so BASIC programs run much slower than programs written in machine language.

Although machine language is much faster, it is also a more difficult language to use; so to speed up writing the game, many programmers opt for BASIC, or some combination of BASIC and machine language. The choice ultimately depends on how critical speed is to the game. Witness the incredible speed of "Munchmaze" or "The Viper," both written in machine language. Other games where speed is not so important, such as "Mystery Spell," use no machine language at all.

The 64 makes machine language programming easier because it has a popular, easy-to-use microprocessor chip, and it has areas of memory where machine language programs can be conveniently tucked away.

Make Your Own Alphabet

Whenever you see a letter or graphic character on the screen, you are looking at one member of a character set. The character set is where the computer goes to see what a character such as A looks like, before it can put it on the screen.

By holding down the SHIFT and Commodore keys, you can switch between two character sets. In one of them, character number one looks like this: A; in the other, it looks like this: a.

This is very important to the game programmer, because with the 64 he can create his own character set. For example, the programmer can tell the computer that character one is a happy face. From then on, moving a happy face around on the screen is just as easy as moving any other character. Here is a short program that changes the A character into a happy face:

```
5 REM DISABLE INTERRUPTS AND REVEAL CHARACTER ROM
10 POKE 56334,PEEK(56334)AND254
20 POKE 1,PEEK(1)AND251
25 REM COPY CHARACTER SET DOWN TO RAM
29 PRINT "PLEASE WAIT 30 SECONDS"
30 FOR I=0 TO 2048
40 POKE 12288+I,PEEK(53248+I)
50 NEXT I
55 REM COVER UP CHARACTER ROM AND REENABLE INTERRU
   PTS
60 POKE 1,PEEK(1)OR4
70 POKE 56334,PEEK(56334)OR1
75 REM ENABLE NEW CHARACTER SET
80 POKE 53272,28
85 REM POKE IN HAPPY FACE OVER "A"
90 FOR I=0 TO 7
100 READ A
110 POKE 12296+I,A
120 NEXT
130 END
195 REM EACH NUMBER IS ONE ROW OF THE DOTS THAT MA
    KE UP THE FACE
200 DATA 60,66,165,129,165,153,66,60
```

Even more powerful is the technique of telling the computer that character one looks like the left half of a spaceship, and character two like the right half. By combining redefined characters, you can create large shapes. This technique is used in "The Hawkmen of Dindrin."

Another Way of Making a Spaceship

Sometimes a game needs objects on the screen that can go through or over other objects, like a spaceship moving over a starfield. Not only can the 64 do this, but also it will automatically detect a collision between objects.

These objects, called sprites, have a number of other useful features. Each of the 504 dots can be assigned a color independent of its neighbor, and the whole sprite can double in size either vertically or horizontally. Although only eight sprites can usually be displayed at a time, most games do not require that many.

Sprites can also be used for animation. The bird in Mystery Spell is a sprite. To make the bird's wings flap, several versions of the bird were drawn, with the wings up, midway, and down. By telling the bird to look like one shape after another, the illusion of flapping wings is achieved.

Small Is Beautiful

Sometimes, instead of large objects, a game needs to work with pixels, the individual dots that make the image on your screen. High-resolution mode allows control over each individual dot on the screen.

With high-resolution graphics it is possible to make very detailed backgrounds on the screen, over which you can move the sprites that play the game. None of the games in this book use this technique because it would require the typing in of 8000 numbers that describe each of the dots on the high-resolution screen.

Color Me 64

Every good game-playing computer has the ability to put color on the screen. Some have as many as 256 different shades of colors, and some have as few as six.

Just as important as how many colors a computer has is how many colors it can display *at once*. The 64 is very good at multicolor graphics. Any character or dot can be any one of 16 colors. Furthermore, each dot within a character or a sprite can have its own color.

Breaking the Sound Barrier

One of the most important features of a good game is sound effects for explosions, fanfares, and other sundry noises.

The Commodore 64 incorporates a minisynthesizer called the SID chip. The SID chip can make three different tones at once, so that harmony and chords are possible. You can hear this in the short songs played by "Richthofen's Revenge."

Furthermore, the SID gives you control over attack, decay, sustain, and release, sophisticated sound characteristics that can make the same note sound like it came from anything from a drum to an underwater oboe.

Join the Party

This book is more than a book of games. The Commodore 64 is a great machine with features that allow arcade-quality games. Some of these features take practice to learn.

Many of the articles include explanations of how the game was designed and how the features of the 64 were exploited. By typing in the games and reading the articles, not only will you have hours of fun playing the games, but you will also be learning many of the techniques needed to design your own games.

Writing Your First Game

Richard Mansfield

Richard Mansfield, senior editor of COMPUTE! Publications, explains the details of a simple game. A beginning programmer can learn a great deal studying this short program.

If you are tempted to write your own games, go ahead. It's a good way to learn to program. Games are basically the same as any other kind of programming.

Computer games fall into two broad categories: 1. imitations of old standards (checkers, Othello) and 2. games which could not be played without a computer (*Space Invaders, Pac-Man*). This second category is more difficult to program for several reasons. For one thing, you've got to think up a whole new, and entertaining, concept and then adjust the action until it is just hard enough to be challenging but not so difficult that people want to give up.

This category (basically arcade games) is especially hard to program precisely because a good computer-only game exploits all of the computer's special attributes: speed, color, and sound. To do this well, to make things look and respond just the way you imagine them, requires a good bit of programming experience. Usually, too, several things are happening *at once* in an arcade game. This often means that such a program must be written in machine language, which is far faster than BASIC.

High Card Slice

Old standards, on the other hand, can often be the best way to get started programming games. You already know the game concept, and cards or dice or game boards are fairly easily constructed and manipulated on your computer screen. To illustrate, let's take a look at a simple simulation of one of the oldest card games, "High Card." The rules are simple: you place a bet, and then you draw a card from the deck. The computer, your opponent, draws a card too, and the highest card wins the money.

One simplification here is that there is no attempt to represent the cards on the screen. The entire game relies simply on words (Ace of Spades, for example) when cards are drawn.

Like most computer programs, the program can be visualized as having four distinct zones: initialization, main loop, subroutines, and data tables. We can go through the steps in programming this game by looking at each zone separately.

Initialization

From lines 10 through 80 we are teaching the computer some basics about this game. Initialization is the activity which must take place before any of the action can begin. Computers are so fast that they will zip up through these lines and start things off in the main loop at line 100 in a flash. However, as programmers, we are aware that several preliminary events took place inside before anything else.

In line 20, the computer discovers that there is a variable called DOLLARS which is set equal to 500. It sets aside a section (like a small box) in its memory which it labels DOLLARS. When the game is running, it will add or subtract from this box (lines 230-240) to keep a running total of how much money you have left to bet. From time to time (line 110), it will check the box and report to the player how much he has. The box labeled DOLLARS is called a *variable* because during the game the amount in it will vary.

Lines 30 through 60 are simple enough—they ask the player to give his or her name. The computer memorizes it in another box called NAME$ and can now speak more personally to the player in lines 140 and 230. Also, the computer prints the rules of the game in line 60.

Line 70 READs four names (the face cards) from the data tables in lines 510 on. It also makes a mental note that it already READ four items. So, when it's asked to READ again (line 80), it will start with the next unread item of data which will be CLUBS. By now, the computer has memorized a variety of important facts: the player's name, the amount of his or her betting purse, the names of the face cards, and the suits of a standard deck. In less than a second, the computer has grasped and filed away the necessary facts to go on to the main loop where all the action takes place.

The Main Loop

After checking that the player has money to bet, the computer asks for the bet, checks again that the bet is possible, and then runs through one cycle of the game starting in line 160. At this point, a programmer might find it worthwhile to visualize the steps involved in the game: draw a card for the player; draw for the computer; decide who won; adjust the player's purse.

Since both draws are essentially identical actions (the only difference will be that we say "Bob draws a . . ." instead of "The computer draws"), we don't need to program the draw twice. This is where subroutines come in handy.

The Subroutine

Twice in the main loop, we GOSUB 300. First the player, then the computer, draws. Line 310 randomly picks two numbers, the card and the suit. If line 320 finds that this selection matches the one drawn just before by the player, it goes back for another draw. Line 330 makes the *name* of the card be the number if it is less than 11 (a face card).

Then line 340 announces the draw using three variables. The first variable (PLAYER$) is set up in either line 160 or 190 as appropriate. Then the CARD$ and SUIT$ variables are selected from the lists that were memorized back in the initialization phase (lines 70-80). The subroutine then RETURNs to the main loop.

Lines 210-240 decide and announce the winner of this round. First, if the variable CARD (the computer's card) is greater than (>) YOURCARD, the computer is declared the winner in line 240, the purse is adjusted, and the main loop is restarted (GOTO 100). If the cards are equal, nothing happens to the purse and the next round begins. Notice that we don't need to say IF YOURCARD > CARD at the start of line 230 to test if the player has won. It's the only possible thing if the computer has gotten this far.

Once you've solved a particular problem, you'll find you can use the solution in many future games. This subroutine which draws cards, for instance, would work just as well for Poker, or Blackjack, or dozens of other games. Subroutines are handy not only because they can be used repeatedly within a program, but because they can also be saved and used repeatedly in future programs. So think up a simple, traditional game and teach it to your computer. There is probably no more pleasurable way to learn programming than to write a game.

High Card

```
10 REM*NECESSARY INITIAL INFORMATION*
20 DOLLARS=500
30 PRINT " WITH WHOM DO I HAVE THE PLEASURE"
40 PRINT " OF PLAYING HIGH CARD SLICE?"
50 INPUT NAME$
60 PRINT " HIGH CARD WINS IN THIS GAME!"
70 DIM SUIT$(4),CARD$(14):FOR I=11 TO 14: READ CAR
   D$(I):NEXT I
80 FOR I=1 TO 4: READ SUIT$(I): NEXT I
90 REM
100 REM*MAIN PROGRAM LOOP*
110 PRINT:PRINT" YOU HAVE $" DOLLARS
120 IF DOLLARS<=0 THEN PRINT" THE GAME IS OVER. YO
    U ARE OUT OF CASH.":END
130 PRINT"WHAT IS YOUR BET";:INPUT BET
140 IF DOLLARS<BET THEN PRINT" YOU ONLY HAVE $"DOL
    LARS" TO BET,"NAME$:GOTO 130
150 YOURCARD=0:YURSUIT=0
160 PLAYER$=NAME$
170 GOSUB300
180 YOURCARD=CARD:YURSUIT=SUIT
190 PLAYER$=" THE COMPUTER"
200 GOSUB300
210 IF CARD>YOURCARD THEN GOTO 240
220 IF CARD=YOURCARD THEN PRINT " A TIE!":GOTO 100
230 PRINT NAME$ " WINS": DOLLARS = DOLLARS + BET:G
    OTO 100
240 PRINT " THE COMPUTER WINS": DOLLARS= DOLLARS-B
    ET:GOTO 100
290 REM
300 REM*SUBROUTINE TO DRAW THE CARDS*
310 CARD=INT(RND(5)*13)+2:SUIT=INT(RND(5)*4)+1
320 IF CARD=YOURCARD AND SUIT=YURSUIT THEN 300:REM
     NO IDENTICAL DRAWS
330 IF CARD<11 THEN CARD$(CARD)=STR$(CARD)
340 PRINT PLAYER$ " DRAWS THE " CARD$(CARD) " OF "
     SUIT$(SUIT)
350 RETURN
490 REM
500 REM* DATA TABLE*
510 DATA JACK,QUEEN,KING,ACE
520 DATA CLUBS,DIAMONDS,HEARTS,SPADES
```

Writing a Simulation Game

Richard Mansfield

A simulation is an imitation of life. It can be the most difficult type of game to create. Thought, rather than fast action, is important. Try the short simulation offered here, then see if you can write one of your own.

There are three basic types of computer games: arcade, adventure, and simulation games. Let's briefly look at the characteristics of arcade and adventure games and then write a simulation.

Realtime Action

Arcade games feature what's called *realtime* action. Unlike chess or bridge, things happen fast. You can't sit back and plan your next move; you must react immediately to the space invaders. In other words, events take place at the same speed as they would in reality: realtime.

Arcade games also have a strong appeal to the eye and ear. There is much animation, color, and sound. In fact, your ability to respond quickly and effectively depends in part on all the clues you get from the graphics and sound effects. Strategy, while often an aspect of arcade play, is clearly secondary. These games are a new kind of athletics: the fun of man versus machine. Like auto racing, arcade games are essentially isometric exercises—you don't run around; you just stay in one place flexing and unflexing your muscles, tensing and relaxing.

Story and Strategy

Strategy, however, is more important in "adventure" games. The emphasis is on planning ahead and solving riddles. It can be like living inside an adventure novel. There is drama, characterization, and plot. You might start out, for example, in a forest with a shovel and a trusty, if enigmatic, companion parrot. As you try to figure out what to do next, the parrot keeps saying "piny dells, piny dells." After wandering aimlessly through the trees, it suddenly comes to you that the bird is saying "pine needles" and you dig through them and find a treasure map.

Your "character" will travel, meet friends and enemies, and have the opportunity to pick up or ignore potentially useful items such as food, magic wands, and medicine. It's customary that you cannot haul tons of provisions. You'd have to decide whether or not to leave the shovel in the forest. Yet you might be sorry that you'd dropped it if you're involved in a cave-in later in the game.

In any case, adventure games are fundamentally verbal. The computer displays the words:

YOU ARE IN A BOAT ON A LAKE. NIGHT IS FALLING.

to which you can respond in any number of ways. You might type:

DIVE OFF BOAT.

and the computer would reply that you now see an underwater cave or whatever. You move through the scenes the way a character moves through a novel. There is generally no penalty if you take time to plan your next move. It's not *realtime*.

Imitations of Life

The third category, simulation, is the least common kind of computer game. This is because to really imitate something, to *simulate* it effectively, you need lots of computer memory to hold lots of variables. However, memory has recently become far less expensive so we can expect to see increasingly effective simulation games. *Star Trek* and *Hammurabi*, both simulations, have long been popular home computer games. Although they are similar to adventure games, simulations are random. That is, there is no secret to discover, no puzzle to solve, no plot. Like real life, things happen with unpredictable, complex results.

Here's a program which simulates investing. The key to simulating is to arrange realistic *interactions* between variables. Look at line 600. If there is "international unrest," the price of gold (PGLD) goes up and the price of Bundtfund stock (PB) goes down. This relationship between gold, stock, and an international crisis is true to life. Alternatively, stock goes up and gold goes down on line 700 during a "market rally."

The game allows you to make investment decisions, and then a "month" passes during which the value of your investments will go up or down. In line 510, three variables are given random values. Stock can gain or lose up to 10 points (variable X), and gold can change by $20 an ounce (Y). Variable Z will be used to simulate flipping a coin. Also notice lines 520 and 525. In 520, we determine whether or not there will be unrest. The variable CH is just a counter. Each "month," CH is raised by one. Two conditions are required for unrest to happen: in a given month, CH must be greater than 4 and it must be less than whatever X turns out to be. If both these conditions are met, CH is reset to zero and we've got international unrest. This has the effect of creating unrest roughly every four to six months. Likewise, another rhythm is set up in line 525 to cause market rallies. In both cases, however, you cannot be certain exactly when to invest in gold or in stocks.

The decision to raise or lower stock prices is made in line 530 and based on the coin toss variable, Z. Again, stocks move in opposition to gold. Prices will rise about 50 percent of the time, but you can never know what will happen in a given month.

Suggested Complications

This is the core, a rough sketch, of an investment simulation game. There is much you can do to make it a more effective simulation and thereby a more enjoyable game. The more variables in a simulation, the better. For example, add leverage and additional "incidents" which affect prices, improve the randomizing, and include other types of investments. You could even use a separate counter which, every five years, causes the X and Y variables to swing more widely to reflect recession/recovery cycles.

As you can see, a simulation should be lifelike. It has interdependent cycles and a degree of unpredictability. Its realism derives from including a sufficient number of variables. And those variables must interact in plausible ways and with just the right amount of randomness. A simulation is a little world you create. You can define cause and effect and then fine-tune the whole thing until it seems well-balanced. Adventure and arcade games are certainly enjoyable, but this investment simulation can be built up to the point where it's just as much fun as any other kind of game.

Mixing Styles

Of course, these three categories—arcade, adventure, and simulation—are somewhat arbitrary. Some of the best games contain elements of each. There are adventure games with graphics—you see the forest, the shovel, the pine needles. After you say DIVE, your character jumps into a lake and the screen transforms into an underwater scene. Likewise, arcade games can include the different "settings" so characteristic of adventure games. Popular arcade games such as *Tron* and *Donkey Kong* change the playfield as you earn more points.

There are several ways to add to the appeal of our investment simulation, beyond just making it a more complex, more accurate simulation. You could add the visuals and sound of arcade games. Try creating a ticker tape across the top of the screen to show price changes and news events. Maybe add a bell sound to indicate the end of further transactions. And from adventure games you could borrow two elements: riddles and the necessity of planning ahead. One easy way to incorporate these two elements would be to make paying taxes a part of the game. After all, the closer it is to real life, the better the simulation.

Investment Simulation

```
5 PRINT"{CLR}"
1Ø CASH=1ØØØØØ:PGLD=4ØØ
15 POKE 53272,23:REM SHIFT TO LOWER CASE
2Ø PB=8Ø
31 PRINT: PRINT"BUNDTFUND IS $"PB" PER SHARE.YOU H
   AVE "B"{4 SPACES}SHARES. -- $"PB*B
33 PRINT" GOLD IS{4 SPACES}$"PGLD" PER OUNCE.
   {2 SPACES}YOU HAVE "GLS" OUNCES. -- $"GLD*PGLD
34 T=PB*B+GLD*PGLD
35 PRINT:PRINT" TOTAL INVESTMENTS -- $"T
36 PRINT:PRINT" YOU HAVE $"CASH" TO SPEND."
4Ø PRINT:PRINT"GRAND TOTAL":PRINT"(INVESTMENTS + C
   ASH){4 SPACES}$"T+CASH
45 IFCK=1THEN5ØØ
5Ø PRINT: PRINT"1.BUY{2 SPACES}2.SELL{2 SPACES}3.D
   ONE"
6Ø INPUTA:IFA=3THENCK=1:GOTO31
1ØØ PRINT"WHICH?{3 SPACES}1.GOLD{4 SPACES}OR
    {4 SPACES}2.STOCK"
11Ø INPUTF
12Ø PRINT"HOW MANY (SHARES{3 SPACES}OR{3 SPACES}OU
    NCES)?"
13Ø INPUTN
14Ø IFF=1THEN16Ø
15Ø PRINCE=PB*N:IFA=1THENCASH=CASH-PRICE:B=B+N:GOT
    O4ØØ
155 CASH=CASH+PRICE:GLD=GLD-N
16Ø PRICE=PGLD*N:IFA=1THENCASH=CASH-PRICE:GLD=GLD+
    N:GOTO4ØØ
17Ø CASH=CASH+PRICE:GLD=GLD-N
```

```
400 GOTO50
500 PRINT"PRESS ANY KEY TO CONT" ;
503 GET C$:IF C$=""THEN 503
505 CK=0:PRINT:PRINT"{CLR}ONE MONTH LATER ...":FOR
    T=1TO700:NEXTT:PRINT
510 X=INT((RND(1)*100)/10):Y=INT((RND(1)*200)/10):
    Z=RND(1)
520 CH=CH+1:IFCH>4ANDCH<XTHENCH=0:GOTO600
525 IFCH=2GOTO600
530 IF Z>.5 THENPB=PB+X:PGLD=PGLD-Y:GOTO31
540 PB=PB-X:PGLD=PGLD+Y:GOTO31
600 PRINT"INTERNATIONAL UNREST...":PGLD=PGLD+2*Y:P
    B=PB-2*X:GOTO31
700 PRINT"MARKET RALLY ...{2 SPACES}":PGLD=PGLD-2*
    Y:PB=PB+3*X:GOTO31
```

1

Writing an Arcade Game

Richard Mansfield

Using the memory-mapped video *could help you create faster moving games. The sample program here will assist you in designing your own fast-moving game.*

When you bring home your computer, usually the first thing everyone expects you to do is to write an arcade game. Who's "everyone"? It could be your children, your friends, even you—anybody who is tired of spending lots of money and wants you to program a game to play at home for free.

The best defense is to politely point out that:

1. Arcade games are among the hardest types of software to write.
2. Professionals, working in teams, can take a year to write one.

However, it is well worth trying to write action games. You might not be able to duplicate the speed or complexity of professional games, but you can create very entertaining games of your own. After you've spent a few weeks getting familiar with BASIC and have typed in a few games, you are ready to take up the challenge. This is one of the best ways to learn some important programming techniques and to explore the graphics and sound capabilities of your computer.

Ten Million IF/THENs

Your main problem is going to be speed. BASIC, though fast enough for most jobs, is pretty slow when it has to keep track of ten aliens, two mother ships, torpedoes, stars, and the player's position. All these things are in motion at once. You need to have a way to control players, to detect collisions, to score points, etc. We at COMPUTE! received a letter from reader John Anderson which touches on these problems:

In order to make a fast, effective "arcade-style" game, I would like to know how to let my computer know where a large number of things are on the screen (like walls in a maze) without 10,000,000 IF/THEN statements. I would also like to know how to keep things, like the little figures racing around during a game, from plowing through walls and wiping them out or coming back onto the other side of the screen.

As Anderson points out, the first solution that comes to mind is to use an IF/THEN test for every possible event in the game. IF the ball hits the target, THEN raise the score. IF the ball misses the target, THEN let it move one more space. And on and on. This quickly slows the action down to a crawl.

POKE Ping-Pong

One of the simpler arcade games is a simulation of Ping-Pong. You need to keep track of only three things: two paddles and one ball. Let's start off by solving the hardest problem. How can we bounce a ball around the screen both quickly and accurately?

The key to the problem is the fact that many computers have an area set aside in RAM which is an *image* of what you see on screen. This is called *memory-mapped video* and most computers have it. It means that if you POKE into that area of RAM, a character will appear on the screen. The next RAM byte address is the next space on screen, and so on. You can use this built-in "map" to tell what is where by using the fast PEEK command, and you can move things quickly with POKEs.

The example program will work on all VICs.

SCR = The address where screen RAM memory starts.

LN = The length of one screen line.

WALL = A solid square that appears when this number is POKEd anywhere into SCR.

BLANK = A blank space character that returns the screen to normal if POKEd into SCR on top of a WALL or FIGURE.

FIGURE = A character that, when POKEd into SCR, looks like a ball.

The memory cells holding the screen image are located in different places. The VIC determines where it starts by using the formula in line 100. First, draw a border around your screen like a picture frame. Perhaps print reversed spaces all around. (See lines 250-310.) This border is very useful. It will let you know when your ball has hit the edge.

LOC is a variable in the program that's always changing whenever the ball changes. It keeps track of the current location of the ball. What you do is keep another variable (VECTR, in this example) which holds the direction and distance of the ball's current motion. When VECTR is added to LOC, we know where to move the ball next.

There are four possible directions to go in the simplest kind of animated games. Traveling up, VECTR = -LN since you subtract the number of spaces in one screen line to move the ball to the line above. Going down is +LN, right is +1, left is -1.

Notice line 180. That is how the computer tells if the ball has reached a border. The next position the figure is supposed to be POKEd into is checked to see if the WALL variable is sitting there. If not, the figure is moved (lines 200-220). If there is a wall, line 190 reverses the figure's direction.

If you type in the example program, you'll be on your way to making a Ping-Pong game that will be as fast as you could want. What's left is to play around with VECTR to get different angles of bounce off walls so the ball can go anywhere. Then add two movable pieces of wall (paddles) and scorekeeping.

Ping-Pong

```
100 SCR=1024:COL=55296:POKE53281,0
110 WALL=160:REM WALL CHARACTER, SOLID SQUARE.TRY
   {SPACE}OTHER CHARACTERS.
120 LN=40
130 GOSUB 260:REM DRAW BORDER
140 LOC=SCR+LN*10+LN/2:CLOC=COL+LN*10+LN/2:REM SCR
   EEN AND COLOR LOCATION
150 VECTR=LN:REM ALSO TRY -1,+1,LN-1,LN+1,ETC.
160 BLANK=32
170 FIGURE=81:REM "BALL"CHARACTER.
180 IF PEEK(LOC+VECTR)<>WALL THEN 200
190 VECTR=-VECTR:REM REVERSE DIRECTION
200 POKE LOC,BLANK:REM ERASE OLD BALL
210 LOC=LOC+VECTR:CLOC=CLOC+VECTR:REM CALCULATE NE
   W POSITION
220 POKELOC+54272,1:POKELOC,81:REM PLACE BALL
230 GOTO180
240 END
250 REM BORDER SUBROUTINE
260 PRINT"{CLR}";:REM CLEAR SCREEN.
270 FOR I=0 TO LN-1:POKE SCR+I,WALL:POKE COL+I,2:N
   EXTI:REM TOP
```

```
280 FOR I=Ø TO LN-1:POKE SCR+LN*24+I,WALL:POKECOL+
    LN*24+I,2:NEXT I:REM BOTTOM
290 FOR I=Ø TO 24: POKESCR+I*LN,WALL:POKECOL+I*LN,
    2:NEXTI:REM LEFT
300 FOR I=Ø TO 24:POKE SCR+LN-1+I*LN,WALL:POKECOL+
    LN-1+I*LN,2:NEXTI:REM RIGHT
310 RETURN
```

1

Adding Joysticks to Your Games

Charles Brannon

Taking advantage of the Commodore 64's fascinating capabilities often involves PEEKs and POKEs which can be confusing at first. This article explains the essentials of using joysticks in your own BASIC programs.

First of all, if you don't yet own a Commodore joystick, you can use the readily available Atari joysticks, or any *Atari-compatible* joystick—which gives you quite a choice. A number of custom sticks are available from outside companies.

The Inside Story

To really understand joysticks, you have to know how they work. Don't worry; joysticks are no more complicated than a light switch. In fact, inside the base of the joystick are five switches that act like push buttons. When you press the joystick north (up), south (down), east (right), or west (left), or press the joybutton, a switch is closed.

You can also move the stick diagonally (NE, SE, SW, NW). How can four buttons give you eight directions? Simple. The joystick is designed so that diagonal movement closes two switches simultaneously.

Joy Bit

Each switch controls one part of a memory location inside your computer. These are called *bits*. A bit can hold only two values—either zero or one. Zero normally means nothing, false, empty, off. One means positive, true, on. Although it may seem confusing at first, the joystick bits are reversed. When the joystick is centered (not deflected in any direction), all the bits are on. They are all ones. But if you move the joystick up, the north bit will become a zero. If you move the joystick diagonally to the lower right, both the south and east bits will become zeros.

Siliconomics

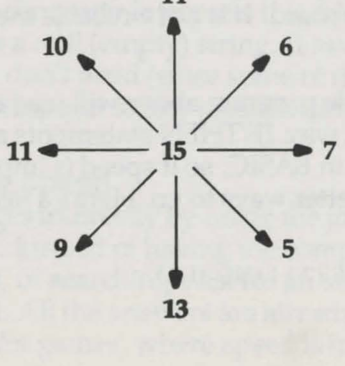
Joysticks would be easier to use if each direction had its own separate memory location. That way, you could check the north, south, east, west, and joybutton bits separately. But to economize (and you always do when designing microchips, where the cost is more than proportional to the amount of silicon used), all the bits are grouped together into a single memory *byte* (eight bits = one byte). The bits are ordered like this:

Direction	**Value When Off (Zero When On)**
North:	**1**
South:	**2**
West:	**4**
East:	**8**
Button:	**16**

As we'll explain shortly, your program will detect which way the joystick is deflected by looking at this byte. The number in the byte will be the sum of all these values. Here's how it works.

Let's ignore the joybutton for a moment. If the stick is not moved, the summed value in the byte would be 15 (1+2+4+8=15). If the stick were moved up (north), the north value would become zero, and the remaining numbers would add up to 14. If the joystick were moved left (west), the west value would become zero, and the remaining numbers would add up to 11.

The easiest way to use the joystick is to read the memory location with the BASIC command PEEK and use IF/THEN statements to take appropriate actions for each direction. Refer to this diagram:



A series of IF/THEN statements might look like this:

```
10 V=PEEK(56321)AND15
20 IF V=14 THEN PRINT "NORTH"
30 IF V=13 THEN PRINT "SOUTH"
40 IF V=7 THEN PRINT "EAST"
50 IF V=11 THEN PRINT "WEST"
60 IF V=6 THEN PRINT "NORTHEAST"
70 IF V=5 THEN PRINT "SOUTHEAST"
80 IF V=9 THEN PRINT "SOUTHWEST"
90 IF V=10 THEN PRINT "NORTHWEST"
100 IF V=15 THEN PRINT "CENTER"
110 GOTO 10
```

Line 10 reads the value of the joystick byte and keeps it in a variable, V. The number 56321 is the memory location for joystick port #1. PEEK reads this location, but you won't get just values from 0-15. Other functions are also read here, such as the joybutton. The AND15 isolates the values we're looking for by turning off all the other unwanted bits. I won't explain here why this works—just take my word for it.

Who's on First?

You can read the second joystick (port #2) by substituting the number 56320 for 56321 in line 10. It might seem logical that the joystick which is read by PEEKing location 56320 should be the first joystick, since it has the lower number, but that's not the way it works. You can't argue with the lettering on the side of your Commodore 64 which clearly shows which is first and which is second.

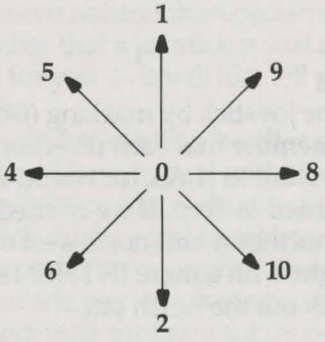
Also, you'll notice that the first joystick will seem to press certain keys on your keyboard. This is a hardware anomaly, but you can play some joystick games by pressing keys in the upper-left part of your keyboard. It is not a reliable method, however.

Another Way

Although the sample program above will read the joystick, it's not necessarily the best way. IF/THEN statements are among the slowest statements in BASIC, so if speed is important (as in games), there are better ways to go. Here's a faster method. Change line 10 to:

```
10 V=15-(PEEK(56321)AND 15)
```

Now the values returned will be:



Notice that the range is smaller here. You can now use the values as the index to an array. Watch how it works. Let's shorten the example program:

```
10 FOR I=0 TO 10:READ A$:MESSAGE$(I)=A$:NEXT I
20 DATA CENTER,NORTH,SOUTH,,WEST,NORTHWEST,SOUTHWE
   ST,,EAST,NORTHEAST,SOUTHEAST
30 V=15-(PEEK(56321)AND15)
40 PRINT MESSAGE$(V):GOTO 30
```

MESSAGE$ (pronounced message-string) is a *string array*. A string array is a single variable name that holds a whole list of strings (a string is any series of characters). Each string has its own box or place in the array. We address the item in the list by calling its number. The READ loop on line 10 fills the MESSAGE$ array with the ten strings. If we say PRINT MESSAGE$(0) we'll get CENTER. PRINT MESSAGE$(5) gives NORTHWEST.

Some of the DATA items are followed by two commas, which are separators. The computer interprets this to mean that between the commas there is a null (empty) string. It saves us from having to include items we don't need (since some of the numbers in the range 0-10 don't correspond to any joystick direction).

Table Look-Up for Speed

Printing the messages indirectly by using the joystick number is a form of *table look-up*. Instead of having the computer go through a bunch of IF/THENs, or searching a list for an answer, table look-up is direct and fast. All the answers are already determined. This is especially useful for games, where speed is important. For example, you could use a different character for any direction the player is facing, and put them into an array to be selected by the joystick number.

Tricky Techniques

You can also read the joystick by masking (isolating) the bits you are looking for. Remember that each direction has a number associated with it. If we want to check for north, we just check to see if the north bit has turned to zero. If we're checking for north this way, we'll capture northeast and northwest as well, which we wouldn't have caught with a mere IF/THEN statement.

Here we'll mask out the north bit:

```
V=(15-PEEK(56321)AND15) AND 1
```

If V=0, the joystick is not deflected north. If V=1, the joystick is being moved north, northeast, or northwest.

To check for left (west):

```
V=(15-PEEK(56321)AND15) AND 4
```

If V=0, there is no movement to the left. If V=4 (yes, 4, not 1), the stick is being pressed left, northwest, or southwest. See how you can separate the original four directions from the eight possible ones?

So, to check for any direction, use:

```
V=PEEK(15-PEEK(56321)AND15) AND number
```

V (or whatever variable you use) will be either zero (not deflected) or nonzero (deflected). Substitute 1, 2, 4, or 8 for *number* (1=up, 2=down, 4=left, 8=right).

The Joybutton

You can check for the joybutton, also called the fire button or trigger, with:

```
B1=PEEK(56321)AND16 (for port #1)
B2=PEEK(56320)AND16 (for port #2)
```

A zero value means the button is pushed. A nonzero value (16) means the button is not pushed. For example, if you are waiting for the user to press the button to begin a game, you could use a loop:

```
500 IF (PEEK(56321)AND16)<>0 THEN 500
```

It's a Natural

Using a joystick in your next game will make it easier to play, since joysticks seem more natural than pressing keys on the keyboard. But remember that a joystick is just a tool. It will not move objects around for you — it will just tell you how the user is deflecting the joystick.

There are other uses for joysticks besides games. Unlike the keyboard, with its 50-odd keys to deal with, the joystick limits input to just nine possibilities (the eight directions and the joybutton). The joystick can be used to select menu options, answer simple questions (left = no, right = yes), and even enter text (as you do with arcade games when you set the high score). Study the following example program for more ideas.

Program Explanation

This program contains three subroutines you can use in your own programs. Lines 10-70 just test the subroutines and show you how to use them. The subroutine at 500 will accept a yes or no answer (left = no, right = yes) and return it in A$.

Lines 700-770 let the user enter a number by counting it up and down with the joystick. The number can be found in the variable C. C will not exceed the limits of MN (minimum) and MX (maximum). The user presses the joybutton to exit. Notice the POKE 198,0. Since the first joystick interferes with the keyboard, this POKE is used to clear it out.

You can use the subroutine at 800 to accept a letter of the alphabet. The letter is returned as a number from 1-26 in the variable C. In the sample program (line 20), it is used to accept a three-digit string of initials.

Joystick Example

```
10 PRINT"ENTER YOUR INITIALS:";
20 GOSUB800:N$=N$+CHR$(C+64):IFLEN(N$)<3THEN20
30 PRINT: PRINT"HOW OLD ARE YOU? ";:GOSUB700:AGE=C
40 PRINT:PRINTN$;", YOU CLAIM TO BE";AGE;"YEARS OL
   D."
50 PRINT:PRINT"IS THAT TRUE?";:GOSUB500
60 PRINTA$:IFA$="YES"THENPRINT"GOOD FOR YOU":END
70 PRINT"SO WHAT IS THE TRUTH?":GOTO 30
500 REM SUBROUTINE FOR YES/NO
505 A$=""
510 V=15-(PEEK(56321)AND15)
520 IF (VAND4)>0 THEN A$="NO"
```

```
530 IF (VAND8)>Ø THEN A$="YES"
540 IFA$=""THEN51Ø
55Ø POKE 198,Ø:REM GET RID OF ANY EXTRA KEYS
56Ø RETURN
6ØØ REM COUNTING SUBROUTINE
61Ø REM C WILL CONTAIN THE COUNT
62Ø REM VARIABLE MX AND MN CONTROL
63Ø REM THE MAXIMUM AND MINIMUM
64Ø REM VALUES ALLOWED.{2 SPACES}USE
65Ø REM GOSUB 7ØØ FOR THE DEFAULT
66Ø REM (1 AND 1Ø), OR GOSUB 71Ø
67Ø REM IF YOU ALTER MX AND MN
7ØØ MN=1:MX=99
71Ø C=MN
72Ø PRINTRIGHT$("{2 SPACES}"+STR$(C),2);"{2 LEFT}"
    ;
73Ø V=15-(PEEK(56321)AND15)
74Ø C=C+((VAND8)=8)*(C<MX)-((VAND4)=4)*(C>MN)
75Ø REM IF FIRE BUTTON PRESSED, EXIT
76Ø IF(PEEK(56321)AND16)=ØTHENPOKE198,Ø:PRINT"
    {2 RIGHT}";:RETURN
77Ø GOTO 72Ø
8ØØ REM TEXT ENTRY:SIMILAR TO NUMBER COUNTING ROUT
    INE
81Ø C=1
82Ø PRINT CHR$(64+C);"{LEFT}";
83Ø V=15-(PEEK(56321)AND15)
84Ø C=C+((VAND8)=8)*(C<26)-((VAND4)=4)*(C>1)
85Ø IF(PEEK(56321)AND16)=ØTHENPOKE198,Ø:PRINT"
    {RIGHT}";:RETURN
86Ø GOTO82Ø
```

Chapter 2

Maze Games

Rats!

Mike Steed 64 Translation by Gregg Peele

This impressive game makes you feel that you are inside *a maze, not just seeing it from above. Three-dimensional views appear as hallways, doors, and corners as you struggle to find the way out.*

You must find your way through a maze displayed from a rat's eye view. After you have solved the maze, the program displays the top view and traces your steps.

First, you are asked what maze size you want, up to 15 by 15 (you may wish to change the DIM statement in line 49—add two to the largest dimension you want — and line 43). Line 45 checks to see if the machine code has been POKEd in, so you have to wait for that only the first time.

The space bar is used to move forward, and the J and L keys are used to turn left and right, respectively (turning doesn't change your location; it just gives you the view in another direction). The M key will display the top view of the maze, mark your position, and tell you in which direction you are headed.

There are five machine language routines in "Rats!" LINE, as its name implies, draws a line; this routine is similar to Applesoft's HPLOT TO or Atari BASIC's DRAWTO command. PLOT sets the hi-res cursor to the position from which the next line is to be drawn, and plots that point on the screen. The COLOR routine fills the screen with color.

INIT removes everything that is not a letter or number from the screen (thus the quarter-square graphics are erased, but not the MOVE XX at the bottom of the screen), and sets all the variables used by the other routines (locations 826-837) to zero.

SCR either loads or saves something to or from the screen. This routine is used to save the screen to memory after the top view of the maze has been displayed the first time, and from then on is used to display the maze almost instantly, so you have to wait only once.

Typing in the Programs

Whenever you run Rats!, you must prepare the computer by first running Program 1. Tape users should *not* enter line 180; likewise, disk users should *not* enter line 190.

Program 1 will automatically LOAD and RUN Program 2. Therefore, it is necessary for tape users to SAVE Program 2 immediately following Program 1, and disk users should SAVE Program 2 on the same disk as Program 1, using the filename Rats.

Program 1. Rats! Part 1

```
100 POKE16384,0:POKE16385,0
110 POKE56578,PEEK(56578)OR3
120 POKE56576,(PEEK(56576)AND252)OR1
130 POKE53272,4:POKE648,128
140 POKE53280,12:POKE53281,12
145 POKE641,0:POKE642,64
150 POKE43,1:POKE44,64:POKE55,0:POKE56,128:POKE646
     ,1:PRINT"{CLR}"
160 REM DISK USERS ENTER LINE 180
170 REM CASSETTE USERS ENTER LINE 190
180 LOAD"RATS",8:RUN:END:REM DISK USERS ONLY
190 POKE 198,1:POKE 631,131:END:REM CASSETTE USERS
     ONLY
```

Program 2. Rats! Part 2

```
2 REM DISK USERS SAVE WITH THE FILENAME RATS
3 PRINT CHR$(142):GX=49152:GOTO 38
4 REM DRAW 3-D VIEW
5 N=2:A=H:B=V:FF=2↑(F-1):SYS IN
6 Z=M%(A,B)*FF:IF ((Z/16) AND 1)=1 THEN RL=-1:GOSU
  B 25:GOTO 8
7 W=M%(A+S,B-R)*FF:IF ((W/128) AND 1)=1 THEN RL=-1
  :GOSUB 21
8 IF ((Z/64) AND 1)=1 THEN RL=1:GOSUB 25:GOTO 10
9 W=M%(A-S,B+R)*FF:IF ((W/128) AND 1)=1 THEN RL=1:
  GOSUB 21
10 IF ((Z/128) AND 1)=1 THEN 14
11 N=N+1:IF N>8 THEN 15
12 A=A+R:B=B+S:IF B<2 THEN 15
13 GOTO 6
14 GOSUB 17
15 RETURN
16 REM DRAW CENTER BACK
17 POKE HX,VX+DX(N):POKE HY,YU(N):SYS PL:POKE HY,Y
   D(N):SYS LI
```

```
18 POKE HX,VX-DX(N):SYS LI:POKE HY,YU(N):SYS LI:PO
   KE HX,VX+DX(N):SYS LI
19 RETURN
20 REM DRAW BACK SIDE
21 POKE HX,VX+RL*DX(N-1):POKE HY,YU(N):SYS PL:POKE
    HX,VX+RL*DX(N):SYS LI
22 POKE HY,YD(N):SYS LI:POKE HX,VX+RL*DX(N-1):SYS
   {SPACE}LI
23 RETURN
24 REM DRAW RIGHT OR LEFT SIDE
25 POKE HX,VX+RL*DX(N-1):POKE HY,YU(N-1):SYS PL:PO
   KE HX,VX+RL*DX(N)
26 POKE HY,YU(N):SYS LI:POKE HY,YD(N):SYS LI:POKE
   {SPACE}HX,VX+RL*DX(N-1)
27 POKE HY,YD(N-1):SYS LI:POKE HY,YU(N-1):IF N>2 T
   HEN SYS LI
28 RETURN
29 REM GET KEYBOARD CHARACTER
30 GET A$:IF A$="" THEN 30
31 RETURN
37 REM INITIALIZE
38 HX=828:HY=829:LINE=12288:PLOT=12665:INIT=12685:
   SCR=12725
39 FL=12726:FH=12730:TL=12734:TH=12738
40 PRINT "{CLR}{5 DOWN}{17 RIGHT}RATS!
41 PRINT "{2 DOWN}{3 RIGHT}SOLVE A MAZE FROM A RAT
   'S EYE VIEW
42 INPUT "{3 DOWN}{7 RIGHT}MAZE SIZE (H,V)
   {3 SPACES}3,3{5 LEFT}";H,V
43 IF H<3 OR H>15 OR V<3 OR V>15 THEN 40
44 PRINT "{CLR}{DOWN}PLEASE WAIT...
45 IF PEEK(LI)=32 AND PEEK(LI+1)=33 AND PEEK(LI+2)
   =48 THEN 48
46 CK=0:FOR L=12288 TO 12761:READ A:POKE L,A:CK=CK
   +A:NEXT:FORK=GXTOGX+23:READGX
47 POKEK,GX:NEXT:IF CK<>50144 THEN PRINT "{DOWN}ER
   ROR IN DATA STATEMENTS":STOP
48 N=H*V-1:H=H+1:V=V+1:D=1
49 DIM M%(17,17),WALK(100),CUT(5),DX(8),YU(8),YD(8
   )
50 FOR J=1 TO V+1:M%(1,J)=4:M%(H+1,J)=1:NEXT
51 MX=79:MY=49:VX=39:VY=24:X=VX
52 FOR J=1 TO 8:DX(J)=X:YU(J)=INT(VY-X*VY/VX):YD(J
   )=INT(VY+X*(MY-VY)/VX)
53 X=INT(X*7/10):NEXT
54 FOR I=2 TO H:M%(I,V+1)=8:M%(I,1)=2:FOR J=2 TO V
   :M%(I,J)=15:NEXT:NEXT
55 R=INT(H/2)+1:S=INT(V/2)+1:M%(R,S)=15
56 PRINT "{CLR}{DOWN}GENERATING MAZE...";:GOSUB 20
   00
```

```
57 REM GENERATE RANDOM MAZE (ALGORITHM FROM ROGERS
   AND STRASSBERGER)
58 FOR IWALK=1 TO N
59 I=Z
6Ø IF M%(R-1,S)>14 THEN I=I+1:CUT(I)=1
61 IF M%(R,S-1)>14 THEN I=I+1:CUT(I)=2
62 IF M%(R+1,S)>14 THEN I=I+1:CUT(I)=3
63 IF M%(R,S+1)>14 THEN I=I+1:CUT(I)=4
64 IF I=Ø THEN 75
65 IF I<>1 THEN I=INT(RND(1)*I)+1
66 ON CUT(I) GOTO 67,69,71,73
67 M%(R,S)=M%(R,S)-(M%(R,S) AND 1):R=R-1
68 M%(R,S)=M%(R,S)-((M%(R,S)/4) AND 1)*4:GOTO 86
69 M%(R,S)=M%(R,S)-((M%(R,S)/8) AND 1)*8:S=S-1
7Ø M%(R,S)=M%(R,S)-((M%(R,S)/2) AND 1)*2:GOTO 86
71 M%(R,S)=M%(R,S)-((M%(R,S)/4) AND 1)*4:R=R+1
72 M%(R,S)=M%(R,S)-(M%(R,S) AND 1):GOTO 86
73 M%(R,S)=M%(R,S)-((M%(R,S)/2) AND 1)*2:S=S+1
74 M%(R,S)=M%(R,S)-((M%(R,S)/8) AND 1)*8:GOTO 86
75 IF D=-1 THEN 79
76 IF R<>H THEN 83
77 IF S<>V THEN 82
78 R=2:S=2:GOTO 84
79 IF R<>2 THEN 83
8Ø IF S<>V THEN 82
81 R=H:S=2:GOTO 84
82 S=S+1:D=-D:GOTO 84
83 R=R+D
84 IF M%(R,S)=15 THEN 75
85 GOTO 59
86 NEXT IWALK
87 MH=H:MV=V:I=INT(RND(1)*(MH-1))+2
88 M%(I,1)=Ø:M%(I,2)=M%(I,2)-((M%(I,2)/8) AND 1)*8
89 H=INT(RND(1)*(MH-1))+2:H1=H:V1=V
9Ø PRINT "{CLR}{DOWN}MAZE COMPLETED.":GOSUB 2ØØØ:G
   OTO 1Ø5
91 REM DISPLAY TOP VIEW OF MAZE
92 HZ=INT(79/MH):VZ=INT(49/MV)
93 SYS IN:POKE 214,24:PRINT TAB(25);"{UP}
   {9 SPACES}{HOME}";
94 POKE HX,1+HZ:POKE HY,1+VZ:SYS PL:POKE HY,MV*VZ+
   1:SYS LI
95 FOR J=1 TO MV:FOR I=2 TO MH:N=M%(I,J):X=I*HZ+1:
   Y=J*VZ+1
96 IF ((N/2) AND 1)=1 THEN POKE HX,X:POKE HY,Y:SYS
    PL:POKE HX,X-HZ:SYS LI
97 IF ((N/4) AND 1)=1 THEN POKE HX,X:POKE HY,Y:SYS
    PL:POKE HY,Y-VZ:SYS LI
98 NEXT:NEXT
```

```
99 RETURN
100 REM MARK PLAYER'S POSITION
101 X=H*HZ-1:Y=V*VZ-1:POKE HX,X+1:POKE HY,Y+1:SYS
    {SPACE}PL
102 POKE HX,X-HZ+2:POKE HY,Y-VZ+2:SYS LI:POKE HY,Y
    +2:SYS PL
103 POKE HX,X+2:POKE HY,Y-VZ+2:SYS LI
104 RETURN
105 FOR X=1 TO MH:FOR Y=1 TO MV:M%(X,Y)=M%(X,Y)+M%
    (X,Y)*16:NEXT:NEXT
106 REM PLAY
107 F=INT(RND(1)*4)+1:ON F GOTO 108,109,110,111
108 R=0:S=-1:GOTO 112
109 R=+1:S=0:GOTO 112
110 R=0:S=+1:GOTO 112
111 R=-1:S=0
112 PRINT "{CLR}{DOWN}PRESS {RVS}J{OFF} TO TURN LE
    FT
113 PRINT "{DOWN}PRESS {RVS}L{OFF} TO TURN RIGHT
114 PRINT "{DOWN}PRESS {RVS}SPACE{OFF} TO GO FORWA
    RD
115 PRINT "{DOWN}PRESS {RVS}M{OFF} TO DISPLAY TOP
    {SPACE}VIEW OF MAZE
116 PRINT "{3 DOWN}{RVS} PRESS ANY KEY TO CONTINUE
     "
117 GOSUB 30:PRINT "{CLR}";:SYS49152:GOSUB 5
118 REM GET KEYSTROKE
119 GOSUB 30
120 ON -(A$="J")-2*(A$="L")-3*(A$=" ")-4*(A$="M")
    {SPACE}GOTO 122,124,131,136
121 GOSUB2000:GOTO 112
122 F=F-1:IF F<1 THEN F=4
123 GOTO 125
124 F=F+1:IF F>4 THEN F=1
125 ON F GOTO 126,127,128,129
126 R=0:S=-1:GOTO 130
127 R=+1:S=0:GOTO 130
128 R=0:S=+1:GOTO 130
129 R=-1:S=0
130 GOTO 135
131 Z=M%(H,V):T=Z*2↑(F-1):T=(T/128) AND 1:IF T=1 T
    HEN GOSUB 2000:GOTO 119
132 NM=NM+1:POKE 214,24:PRINT TAB(25);"{UP}MOVE";N
    M;"{HOME}";
133 IF NM<100 THEN WALK(NM)=F
134 H=H+R:V=V+S:IF V<2 THEN 147
135 GOSUB 5:GOTO 119
136 IF NOT MS THEN 138
```

```
137 POKE FL,218:POKE FH,49:POKE TL,Ø:POKE TH,128:S
    YS SC:GOTO 139
138 GOSUB 92:POKE FL,Ø:POKE FH,128:POKE TL,218:POK
    E TH,49:SYS SC:MS=-1
139 GOSUB 1Ø1:PRINT "{HOME}YOU ARE FACING ";: ON F
     GOTO 14Ø,141,142,143
14Ø PRINT "NORTH";:GOTO 144
141 PRINT "EAST";:GOTO 144
142 PRINT "SOUTH";:GOTO 144
143 PRINT "WEST";
144 PRINT ".{2 SPACES}PRESS ANY KEY TO":PRINT "CON
    TINUE":GOSUB 3Ø
145 PRINT "{HOME}{39 SPACES}":PRINT "{8 SPACES}"
146 GOSUB 5:GOTO 119
147 GOSUB2ØØØ:V=V1:H=H1:IF MS THEN POKE FL,218:POK
    E FH,49:POKE TL,Ø:POKE TH,128
148 IF MS THEN SYS SC:GOTO 15Ø
149 GOSUB 92
15Ø GOSUB 1Ø1
151 PRINT "{HOME}{DOWN}CONGRATULATIONS-YOU'RE OUT
    {SPACE}IN";NM;"STEP!{LEFT}{INST}S"
152 REM DRAW PATH WALKED
153 POKE HX,H*HZ-HZ/2+1:POKE HY,V*VZ-VZ/2+1:SYS PL
154 FOR N=1 TO NM:IF N>1ØØ THEN 158
155 F=WALK(N):V=V+(F=1)-(F=3):H=H+(F=4)-(F=2)
156 POKE HX,H*HZ-HZ/2+1:POKE HY,V*VZ-VZ/2+1:SYS LI
157 NEXT
158 PRINT:END
16Ø DATA 32, 33, 48, 173, 58, 3, 133, 2
17Ø DATA 173, 59, 3, 133, 195, 32, Ø, 49
18Ø DATA 173, 62, 3, 2Ø5, 63, 3, 16, 8
19Ø DATA 24Ø, 6, 32, 173, 48, 76, 3, 48
2ØØ DATA 96, 169, 128, 24, 1Ø9, 6Ø, 3, 56
21Ø DATA 237, 58, 3, 141, 63, 3, 169, 128
22Ø DATA 24, 1Ø9, 61, 3, 56, 237, 59, 3
23Ø DATA 141, 64, 3, 162, 128, 142, 66, 3
24Ø DATA 142, 69, 3, 232, 142, 67, 3, 142
25Ø DATA 68, 3, 173, 63, 3, 2Ø1, 128, 176
26Ø DATA 11, 169, 127, 141, 68, 3, 169, Ø
27Ø DATA 56, 237, 63, 3, 41, 127, 141, 63
28Ø DATA 3, 173, 64, 3, 2Ø1, 128, 176, 11
29Ø DATA 169, 127, 141, 67, 3, 169, Ø, 56
3ØØ DATA 237, 64, 3, 41, 127, 141, 64, 3
31Ø DATA 173, 63, 3, 2Ø5, 64, 3, 176, 32
32Ø DATA 174, 63, 3, 172, 64, 3, 142, 64
33Ø DATA 3, 14Ø, 63, 3, 173, 68, 3, 141
34Ø DATA 66, 3, 173, 67, 3, 141, 69, 3
35Ø DATA 169, 128, 141, 67, 3, 141, 68, 3
36Ø DATA 173, 63, 3, 74, 141, 65, 3, 169
```

```
370 DATA Ø, 141, 62, 3, 96, 173, 68, 3
380 DATA 56, 233, 128, 24, 1Ø9, 58, 3, 141
39Ø DATA 58, 3, 173, 69, 3, 56, 233, 128
4ØØ DATA 24, 1Ø9, 59, 3, 141, 59, 3, 173
41Ø DATA 65, 3, 24, 1Ø9, 64, 3, 141, 65
42Ø DATA 3, 238, 62, 3, 173, 65, 3, 2Ø5
43Ø DATA 63, 3, 48, 35, 24Ø, 33, 56, 237
44Ø DATA 63, 3, 141, 65, 3, 173, 66, 3
45Ø DATA 56, 233, 128, 24, 1Ø9, 58, 3, 141
46Ø DATA 58, 3, 173, 67, 3, 56, 233, 128
47Ø DATA 24, 1Ø9, 59, 3, 141, 59, 3, 96
48Ø DATA 169, Ø, 133, 168, 169, 32, 133, 196
49Ø DATA 165, 2, 2Ø1, 8Ø, 176, 56, 165, 195
5ØØ DATA 2Ø1, 5Ø, 176, 5Ø, 234, 234, 234, 234
51Ø DATA 7Ø, 2, 38, 168, 1Ø6, 38, 168, 133
52Ø DATA 195, 1Ø, 1Ø, 1Ø1, 195, 1Ø, 1Ø, 38
53Ø DATA 196, 1Ø, 38, 196, 234, 234, 234, 133
54Ø DATA 195, 166, 168, 189, 99, 49, 133, 168
55Ø DATA 164, 2, 177, 195, 162, 15, 221, 1Ø3
56Ø DATA 49, 24Ø, 4, 2Ø2, 16, 248, 96, 173
57Ø DATA 98, 49, 24Ø, 6, 138, 5, 168, 17Ø
58Ø DATA 2Ø8, 8, 138, 73, 255, 5, 168, 73
59Ø DATA 255, 17Ø, 189, 1Ø3, 49, 164, 2, 145
6ØØ DATA 195, 96, 1, 1, 2, 4, 8, 32
61Ø DATA 126, 123, 97, 124, 226, 255, 236, 1Ø8
62Ø DATA 127, 98, 252, 225, 251, 254, 16Ø, 234
63Ø DATA Ø, 173, 6Ø, 3, 141, 58, 3, 133
64Ø DATA 2, 173, 61, 3, 141, 59, 3, 133
65Ø DATA 195, 32, Ø, 49, 96, 162, 128, 16Ø
66Ø DATA Ø, 134, 254, 132, 253, 177, 253, 41
67Ø DATA 127, 2Ø1, 64, 48, 2, 169, 32, 145
68Ø DATA 253, 2ØØ, 2Ø8, 241, 232, 224, 132, 2Ø8
69Ø DATA 232, 169, Ø, 17Ø, 157, 58, 3, 232
7ØØ DATA 224, 12, 2Ø8, 248, 96, 169, 218, 133
71Ø DATA 251, 169, 49, 133, 252, 169, Ø, 133
72Ø DATA 253, 169, 128, 133, 254, 162, 4, 16Ø
73Ø DATA Ø, 177, 251, 145, 253, 136, 2Ø8, 249
74Ø DATA 23Ø, 252, 23Ø, 254, 2Ø2, 48, 2, 2Ø8
75Ø DATA 24Ø, 96
1ØØØ DATA 162, Ø, 169, 1, 157, Ø, 216, 157
1Ø1Ø DATA Ø, 217, 157, Ø, 218, 157, Ø, 219
1Ø2Ø DATA 232, 2Ø8, 241, 96, 234, 234, 234, Ø
2ØØØ SØ=54272:FORE=SØTOSØ+28:POKEE,Ø:NEXT
2Ø1Ø POKE54296, 15 :POKE54277, 51 :POKE54278, 211
2Ø2Ø POKE 54276, 33 :POKE 54273, 63 :POKE54272, 75
2Ø3Ø FORT=1TO 2ØØ :NEXT:POKE54276, 32:FORT=1TO 1ØØ
     :NEXT
2Ø4Ø FORE=SØTOSØ+28:POKEE,Ø:NEXT
2Ø5Ø RETURN
```

2
Goblin

Dan Goff   64 Translation by Patrick Parrish

In "Goblin," custom characters are used to create a simple yet entertaining game. The object is to capture the scowling creatures with your goblin while avoiding the many block-shaped obstacles that lie in your path.

After obstacles and sad faces have been positioned, "Goblin" begins when the main character appears at the bottom of the screen. As the game progresses, the goblin moves continually upward and the player controls only its horizontal movement. The O and P keys, in conjunction with the GET command in line 260, enable the player to move the goblin left and right, respectively. Children especially like the cumulative effect of the GET statement; they make rapid key punches and then wait for the delayed effects.

As each sad face is captured by the goblin, the score is updated and printed at the upper left. If the goblin successfully clears the screen of all the faces, an entirely new playfield will be provided. A game lasts as long as you wish.

A single round ends when the goblin crashes into an obstacle. At this point, the remaining sad faces smile, and you are asked if you wish to play again.

If you play again, your previous highest score will be posted as the new game begins. The incentive to exceed a record score makes any game more fun.

Goblin

```
80 POKE 53280,2:POKE 53281,1
90 PRINT"{CLR}{7 DOWN}{4 RIGHT}PLEASE WAIT...DEFIN
   ING CHARACTERS";
100 POKE 52,48:POKE 56,48:CLR:POKE56334,PEEK(56334
   )AND254
105 POKE1,PEEK(1)AND251
108 FORN=0TO2047:POKEN+12288,PEEK(N+53248):NEXTN
109 FOR N=0 TO 7:POKEN+12320,PEEK(N+54064):NEXT N
110 IFS>HSTHENHS=S
112 RESTORE:B=4:Z=1964:Z1=Z+54272:W=0:S=J:G=0
```

```
115 VS=54296:AD=54277:SR=54278:WF=54276:LB=54272:H
    B=54273
120 FOR X=ØTO31:READ A:POKEX+12288,A:NEXT
123 POKE 1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
125 POKE 53272,(PEEK(53272)AND24Ø)+12
13Ø PRINT"{CLR}{GRN}{14 RIGHT}{RVS}G O B L I N"
14Ø PRINT"{HOME}{RED}{2 DOWN}{RVS}"SPC(17)"HS="HS
145 PRINT"{HOME}{BLK}{22 DOWN}{RVS}O=LEFT";SPC(27)
    ;"P=RIGHT"
15Ø FOR I=1 TO 118
16Ø X=INT(RND(1)*68Ø)+1144
17Ø IFPEEK(X)=BTHEN 16Ø
18Ø POKEX,B:POKEX+54272,Ø:NEXTI
19Ø FORI=1TO36
195 G1=Ø
2ØØ X=INT(RND(1)*68Ø)+1144
21Ø IF PEEK(X)=BORPEEK(X)=1ORPEEK(X)=3THEN 2ØØ
22Ø IFPEEK(X+39)=BANDPEEK(X+4Ø)=BANDPEEK(X+41)=BTH
    ENPOKEX,3:POKEX+54272,Ø:G1=1
225 IF G1=1 THEN G=G+1:GOTO 24Ø
23Ø POKEX,1:POKEX+54272,Ø
24Ø NEXT I
25Ø POKEZ,32:Z=Z-4Ø:Z1=Z1-4Ø:IF Z<1144 THEN Z=Z+76
    Ø:Z1=Z1+76Ø
26Ø GET A$:IFA$="O"THENZ=Z-1:Z1=Z1-1
27Ø IFA$="P"THENZ=Z+1:Z1=Z1+1
28Ø IFPEEK(Z)=B THEN 41Ø
29Ø IFPEEK(Z)=1 THEN GOSUB 33Ø
3ØØ POKEZ,Ø:POKEZ1,Ø:FORT=1TO22Ø:NEXT
31Ø IFW=36-G THEN J=S:GOSUB35Ø:GOTO11Ø
32Ø GOTO 25Ø
33Ø W=W+1:S=S+25:PRINT"{HOME}{BLU}{2 DOWN}"S:POKE
    {SPACE}VS,15:POKE AD,3Ø:POKE SR,2ØØ:POKE WF,17
34Ø POKEHB,71:POKELB,12:FORT=1TO9Ø:NEXTT:POKEVS,Ø:
    POKEHB,Ø:POKELB,Ø:RETURN
35Ø PRINT"{HOME}{RED}{18 DOWN}{8 RIGHT}{RVS}******
    ALL RIGHT!******"
355 FORI=1TO1Ø:GETC$:NEXTI:REM COLLECT GARBAGE
36Ø POKE VS,15:POKE AD,3Ø:POKE SR,2ØØ:POKE WF,17:F
    OR I=1 TO 17
37Ø H=INT(RND(Ø)*1Ø)+21:L=INT(RND(Ø)*45)+21Ø:POKE
    {SPACE}HB,H:POKE LB,L
38Ø FOR T=1 TO 8Ø:NEXT T:NEXTI:POKE VS,Ø:POKE HB,Ø
    :POKE LB,Ø
4ØØ RETURN
41Ø POKEZ,2:POKEVS,15:POKEAD,3Ø:POKESR,2ØØ:POKEWF,
    129:POKE HB,2:POKE LB,125
415 FOR I=1 TO 4ØØ:NEXT I:POKE VS,15:POKE HB,Ø:POK
    E LB,Ø
```

```
420 FORX=1144TO1823:IF PEEK(X)<>1THEN NEXTX
430 IFPEEK(X)=1THEN POKEX,3:NEXTX
440 J=Ø
445 FORI=1TO1Ø:GET C$:NEXTI
450 PRINT"{HOME}{BLU}{2Ø DOWN}{RVS}PLAY AGAIN? (Y/
    N)":POKE 646,14
465 GET C$:IF C$="" THEN 465
470 IFC$="Y"THEN 11Ø
490 POKE53272,21:POKE5328Ø,14:POKE53281,6:POKE 52,
    5Ø:POKE56,5Ø:PRINT"{CLR}SEE YA!"
500 DATA126,219,219,255,165,9Ø,9Ø,165,6Ø,66,165,12
    9,153,165,66,6Ø
510 DATA 17Ø,85,17Ø,85,126,219,255,189,6Ø,66,165,1
    29,165,153,66,6Ø
520 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
```

Snake Escape

Daryl Biberdorf 64 Translation by Patrick Parrish

You'll have to watch out for poisonous mushrooms as you race against the clock towards your goal in "Snake Escape."

In "Snake Escape," your goal is to move a snake out of a poisonous garden. There are approximately 150 poisonous plants on the screen after you enter your skill level. The snake appears in the upper-left corner after all poisonous plants have been placed. You then attempt to get the snake to the escape hole within the time limit you chose earlier.

The snake must reach the hole without hitting a poisonous plant, running into itself, or running out of time. If it reaches the escape hole safely, you will receive a bonus in addition to your score. The snake grows as it moves along; you receive one point for each body segment it adds while moving. If it runs into itself or a poisonous plant, a cross will appear in the center of the screen with your score and the number of remaining snakes. You may stop the snake if you wish by simply releasing all keys, but remember this costs you time.

Strategy

If you are running your snake near the left or right edges of the screen, remember that the 64 has horizontal screen wraparound. You may end up hitting a poisonous plant on the other side of the screen, so be careful! Occasionally, the snake will be cornered between plants and itself due to a miscalculation in maneuvering. Try to fill up all the spaces you can in the cornered-off area. You may lose a snake, but you will still receive a few extra points. Also, try to keep moving at all times. And watch where you're going.

The direction in which the snake moves is determined in lines 200 through 230. As written, keys I (up), J (left), K (right), and M (down) move the snake. If you aren't comfortable controlling the snake with these keys, you can easily change the program to accept other key commands.

For instance, suppose you want to use the Z key rather than the J key to move the snake left. Since location 197 reads the keyboard, you must first determine the number which is POKEd into this location when Z is pressed. Type the following line:

```
1 PRINT PEEK(197):FOR I=1 TO 400:NEXT I:GOTO 1
```

and then RUN the program. Next press the Z key, and the number in location 197 corresponding to the Z key (12) will print repeatedly on the screen. Try some other keys, noting their values, then hit the RUN/STOP key.

You are now ready to make the modification in line 200: substitute 12 for 34. RUN the program (after deleting line 1, of course); you can move the snake left with the Z key.

Snake Escape

```
5 GOTO100
10 POKE54296, 15 :POKE54277, 17 :POKE54278, 17
15 POKE 54276, 17 :POKE 54273, 28 :POKE54272, 49
20 POKE54276,0:POKE54273,0:POKE54272,0
30 RETURN
100 SO=0:SR=3
110 GOSUB30000:GOSUB29000
120 PRINT"{CLR}"
130 GOSUB28000:GOSUB8000:GOSUB9000:GOSUB28000
140 TI$="000000"
150 CL=INT(RND(1)*7)+1:IFCL=5ORCL=3THEN150
160 IFTI$=L$THENGOSUB7000:GOTO130
170 IFDH=0THENPOKEB,HC
180 POKEB,HC:POKECO,CL
190 K=PEEK(197)
200 IFK=34THENDR=-1:GOTO250:REM LEFT
210 IFK=37THENDR=1:GOTO250:REM RIGHT
220 IFK=33THENDR=-40:GOTO250:REM UP
230 IFK=36THENDR=40:GOTO250:REM DOWN
240 GOTO160
250 POKEB,BC:B=B+DR:CO=CO+DR:SO=SO+1
260 IFPEEK(B)=88THENDH=0:GOTO9500
270 IFPEEK(B)=160THENGOSUB5000:GOTO120
280 IFPEEK(B)=81THENGOTO9500
300 IFB<1024ORB>2023THENB=B-DR:CO=CO-DR
310 GOSUB10:GOTO150
4000 REM PRINT INSTRUCTIONS
4010 PRINT"{CLR}{DOWN}{BLU}{5 RIGHT}YOUR GOAL IS T
     O MOVE THE SNAKE OUT OF THE{2 SPACES}POISON P
     ATCH."
4020 PRINT"{DOWN}{GRN}{5 RIGHT}TRY TO AVOID ALL PO
     ISON ({BLK}X{CYN})."
```

```
4030 PRINT"{3 DOWN}{RED}CONTROLS:":PRINT"{PUR} J=
     {RVS}LEFT":PRINT"{GRN} K={RVS}RIGHT"
4040 PRINT"{CYN} I={RVS}UP":PRINT"{RED} M={RVS}DOW
     N"
4050 PRINT"{DOWN}{RED}POINT VALUES:"
4060 PRINT"{BLU}BODY SEGMENT={RVS}1{OFF} POINT"
4070 PRINT"{2 DOWN}{RED}YOU WILL RECEIVE A BONUS F
     OR ESCAPING."
4080 PRINT"{3 DOWN}{PUR}{RVS}{8 RIGHT}HIT A KEY TO
      START "
4090 GETA$:IFA$=""THEN4090
4100 RETURN
5000 VB=Ø:POKE53280,3:POKE53281,1
5010 IFS=1THENVB=20
5020 IFS=2THENVB=30
5030 IFS=3THENVB=40
5035 IFS=4THENVB=50
5040 BN=FNSC(VB)
5050 PRINT"{CLR}{6 DOWN}{8 RIGHT}{BLU}...YOU HAVE
     {SPACE}ESCAPED!!!"
5060 SO=SO+BN
5070 PRINT"{2 DOWN}{15 RIGHT}{RED}{RVS}BONUS{OFF}:
     {RVS}{BLU}"BN"{OFF}"
5080 PRINT"{2 DOWN}{15 RIGHT}{RVS}{PUR}SCORE{OFF}:
     {RVS}{GRN}"SO
5090 PRINT"{2 DOWN}{8 RIGHT}{BLU}"SR" {RED}SNAKES
     {SPACE}REMAINING"
5100 POKE54296, 15 :POKE54277, 83 :POKE54278, 50
5102 FORHI=33TO 57STEP2:LO=INT(RND(Ø)*5Ø)+18Ø
5103 POKE 54276,17:FORJ=1TO6Ø:NEXTJ:POKE 54273,HI:
     POKE54272,LO:NEXT
5106 FORT=1TO 2ØØ :NEXT:POKE54276,Ø:POKE54273,Ø:PO
     KE54272,Ø
5120 DH=2:RETURN
6000 PRINT"{CLR}{1Ø DOWN}{12 RIGHT}{BLU}VVVVVVVVVV
     VVV"
6003 PRINT"{12 RIGHT}{BLU}V{RVS}{CYN}{11 RIGHT}
     {OFF}{BLU}V"
6005 PRINT"{12 RIGHT}VVVVVVVVVVVVV"
6010 PRINT"{HOME}{11 DOWN}{13 RIGHT}{RVS}{BLK} GAM
     E"
6020 POKE54296, 15 :POKE54277, 53 :POKE54278, 69
6021 POKE 54276, 33 :POKE 54273, 3 :POKE54272, 244
6022 FORT=1TO 9ØØ :NEXT:POKE54276,Ø:POKE54273,Ø:PO
     KE54272,Ø
6025 POKE36874,15Ø:PRINT"{HOME}{11 DOWN}{18 RIGHT}
     {RVS}{BLK} OVER "
6026 POKE54296, 15 :POKE54277, 53 :POKE54278, 69
6027 POKE 54276, 33 :POKE 54273, 2 :POKE54272, 163
```

```
6028 FORT=1TO 9ØØ :NEXT:POKE54276,Ø:POKE54273,Ø:PO
     KE54272,Ø
6Ø4Ø PRINT"{3 DOWN}{12 RIGHT}{RED}PLAY AGAIN ?"
6Ø5Ø GETP$:IFP$=""THEN6Ø5Ø
6Ø6Ø IFP$="Y"THENSO=Ø:SR=3:LK=Ø:GOTO12Ø
6Ø7Ø IFP$<>"N"THEN6Ø5Ø
6Ø8Ø PRINT"{3 DOWN}{17 RIGHT}BYE!{HOME}":END
7ØØØ SR=SR-1:POKE5328Ø,3:POKE53281,1
7Ø1Ø PRINT"{CLR}{6 DOWN} {RED}WHEW! YOU HAVE JUST
     {SPACE}DIED OF EXAUSTION!"
7Ø2Ø PRINTSPC(14)"{4 DOWN}{GRN}Z{PUR}SCORE{OFF}:
     {RVS}{GRN}"SO
7Ø3Ø PRINTSPC(9)"{5 DOWN}{RED}"SR"{BLU}SNAKES REMA
     INING"
7Ø4Ø POKE54296, 1Ø :POKE54277, 31 :POKE54278, 17
7Ø42 POKE 54276, 33 :POKE 54273, 5 :POKE54272, 71
7Ø43 FORVØ=15TO5STEP-.5:POKE54296,VØ:FORT=1TO1ØØ:N
     EXT:NEXT
7Ø45 POKE54276,Ø:POKE54273,Ø:POKE54272,Ø:POKE54296
     ,Ø
7Ø5Ø FORT=1TO2ØØØ:NEXT
7Ø6Ø IFSR=ØTHEN6ØØØ
7Ø7Ø RETURN
8ØØØ POKE5328Ø,4:POKE53281,1:PRINT"{CLR}{3 DOWN}"S
     PC(42)"{RED}CHOOSE YOUR SKILL:"
8ØØ5 PRINT"{2 SPACES}{17 T}"
8Ø1Ø PRINTSPC(51)"{DOWN}{BLU}LEVEL 1=6Ø SECONDS"
8Ø2Ø PRINTSPC(51)"{RED}LEVEL 2=45 SECONDS"
8Ø3Ø PRINTSPC(51)"{GRN}LEVEL 3=3Ø SECONDS"
8Ø4Ø PRINTSPC(51)"{PUR}LEVEL 4=15 SECONDS"
8Ø45 PRINT"{3 DOWN}{7 RIGHT}{YEL}L{BLU}E{GRN}V
     {PUR}E{CYN}L {RED}?"
8Ø5Ø GETS$:IFS$=""THEN8Ø5Ø
8Ø6Ø S=VAL(S$)
8Ø7Ø IFS=1THENL$="ØØØ1ØØ":RETURN
8Ø8Ø IFS=2THENL$="ØØØØ45":RETURN
8Ø9Ø IFS=3THENL$="ØØØØ3Ø":RETURN
81ØØ IFS=4THENL$="ØØØØ15":RETURN
811Ø GOTO8Ø5Ø
9ØØØ POKE5328Ø,4:POKE53281,8:PRINT"{CLR}"
9Ø1Ø FORF=1TO15Ø:D=INT(RND(1)*966)+1Ø58
9Ø2Ø POKED,88:POKED+54272,1:FORJ=1 TO2Ø:NEXTJ:POKE
     D+54272,Ø:NEXTF
9Ø3Ø POKE2Ø23,16Ø:POKE2Ø22,16Ø:POKE1983,16Ø:POKE19
     82,16Ø
9Ø4Ø POKE56295,6:POKE56294,6:POKE56255,6:POKE56254
     ,6
9Ø5Ø POKE1943,32:POKE2Ø21,32
9Ø6Ø RETURN
```

```
9500 POKE54296, 15 :POKE54277, 53 :POKE54278, 69
9505 POKE 54276, 33 :POKE 54273, 5 :POKE54272, 71
9510 FORT=1TO 900 :NEXT:POKE54276,0:POKE54273,0:PO
     KE54272,0
9515 POKE54296, 15 :POKE54277, 53 :POKE54278, 69
9520 POKE 54276, 33 :POKE 54273, 3 :POKE54272, 244
9525 FORT=1TO 900 :NEXT:POKE54276,0:POKE54273,0:PO
     KE54272,0
9530 POKE54296, 15 :POKE54277, 53 :POKE54278, 69
9533 POKE 54276, 33 :POKE 54273, 2 :POKE54272, 163
9536 FORT=1TO 900 :NEXT:POKE54276,0:POKE54273,0:PO
     KE54272,0
9540 SR=SR-1
9550 PRINT"{HOME}{10 DOWN}"SPC(18)"{RVS}{WHT}
     {RIGHT} {RIGHT}"SPC(37)"RIP"SPC(37)"{RIGHT}
     {RIGHT}"SPC(37)"{RIGHT} {RIGHT}{OFF}"
9560 FORT=1TO1000:NEXTT
9570 POKE53280,3:POKE53281,1:PRINT"{CLR}{5 DOWN}"
9580 PRINTSPC(14)"{RED}TOO BAD!!"
9590 PRINT"{4 DOWN}{14 RIGHT}{RVS}{BLU}SCORE{OFF}:
     {RVS}{PUR}"SO"{OFF}"
9600 PRINTSPC(8)"{4 DOWN}{GRN}"SR"{BLU}SNAKES REMA
     INING"
9610 FORT=1TO2000:NEXTT:IFSR=0THEN6000
9620 GOTO120
10000 POKEV,15:POKES3,217:POKES3,217:POKEV,0:POKES
      3,0:RETURN
28000 BC=81:HC=87:B=1024:S3=36876:CO=55296:LK=0:RE
      TURN
29000 DEFFNA(L)=INT(RND(1)*L)+1064
29010 DEFFNSC(L)=INT(RND(1)*L)+5:RETURN
30000 POKE53280,3:POKE53281,1
30010 PRINT"{CLR}{8 DOWN}{11 RIGHT}{RVS}{RED}
      {17 SPACES}"
30020 PRINT"{11 RIGHT}{RVS}{RED} {GRN}{15 SPACES}
      {RED} "
30030 PRINT"{11 RIGHT}{RVS}{RED} {GRN} SNAKE ESCAP
      E! {RED} "
30040 PRINT"{11 RIGHT}{RVS}{RED} {GRN}{15 SPACES}
      {RED} "
30050 PRINT"{11 RIGHT}{RVS}{RED}{17 SPACES}"
30070 PRINT"{2 DOWN}{12 RIGHT}{BLU}INSTRUCTIONS ?"
30080 GETI$:IFI$=""THEN30080
30090 IFI$="Y"THENGOSUB4000:GOTO30120
30100 IFI$="N"THEN30120
30110 GOTO30080
30120 RETURN
```

2

The Viper

Dave and Casey Gardner 64 Version by Charles Brannon

"The Viper" is a fast-action game with 60 difficulty levels. A joystick is required.

The Viper is a fast, furious, *hungry* snake. It races about, devouring its favorite food — asterisks! And the more it eats, the bigger it gets. Since snakes have a hard time growing wider, the Viper simply gets longer. Since the Viper has such sharp, venomous teeth, it must not in its haste accidentally run into its own lengthening body. To make things especially interesting, the Viper must maneuver through a maze with electric walls. One false move means certain doom.

With a joystick you can experience the perils of the Viper. The program is easy to set up and play. Just follow the screen instructions. Maneuver the Viper with a joystick plugged into port one.

You can choose from various difficulty levels to control the Viper's speed. You also select one of three courses — no maze, the easy maze, or the hard maze. Your score is the number of those delicious asterisks eaten multiplied by the skill level you selected, so the harder the game, the more possible points. You get twice as many points for the easy maze, and five times as many for the hard maze.

A Word to Programmers

In order to get the game to run fast enough, the entire main loop of the program was written in machine language. The resulting speed was so fast that delay loops had to be inserted just to slow it down to a barely playable level. If you're brave enough, try level 20 — you'll never be able to play it. If anyone can score any points on level 20 with the hard maze, it will be truly miraculous.

Another feature is the word VIPER that moves about on the title screen. No, it's not high-resolution graphics, and it's not made of sprites, but rather from simple character graphics found on the keyboard. The movement works with programmable INSerts and DELetes. Again, look it over. You may be able to use the technique for animation in your next game.

The Viper

```
100 DT=60:DIM MA(DT),Q(100),I%(15)
110 I%(14)=-40:I%(13)=40:I%(11)=-1:I%(7)=1
120 I%(10)=-41:I%(6)=-39:I%(9)=39:I%(5)=41:JOY=563
    21
130 FORJ=1TODT:READMA(J):NEXT
140 PRINT"{WHT}{CLR}"CHR$(142):C=54272:SC=1024:POK
    E53281,2:POKE53280,8
150 MZ=0:P=0:DR=0
160 CURR=251:SPEED=49352:INDEX=SPEED+1:LNGTH=INDEX
    +1:RTN=LN+1
170 SID=54272:V=SID+24:S1=SID:S2=SID:S3=S2:A=2:N=2
    :MM=0:S4=SID+4
180 FORI=0TO24:POKESID+I,0:NEXT:POKESID+1,25:POKES
    ID+5,6:POKESID+6,0
190 POKESID+24,15
200 GOSUB410:POKESID+5,6:POKESPEED,19-SK
210 FORJ=1024TO1063:POKEJ+C,7:POKEJ,160:NEXT
220 FORJ=1064TO2024STEP40:POKEJ+C,7:POKEJ,160:NEXT
230 FORJ=2023TO1984STEP-1:POKEJ+C,7:POKEJ,160:NEXT
240 FORJ=1983TO1063STEP-40:POKEJ+C,7:POKEJ,160:NEX
    T
250 M=INT(RND(1)*1000)+SC
260 IFPEEK(M)<>32THEN250
270 POKEM,42:POKEM+C,1
280 S=INT(RND(1)*1000)+SC
290 IFPEEK(S)<>32THEN280
300 POKE S,90:POKES+C,16*RND(1):IF(PEEK(56321)AND1
    5)=15THEN300
310 S%=S/256:POKECURR,S-S%*256:POKECURR+1,S%:POKEI
    NDEX,0
320 POKELNGTH,N:SYS49152+5:REM MAIN LOOP GOTO 170
330 HIT=PEEK(RTN)
340 IFHIT<>160ANDHIT<>214THEN360
350 S=PEEK(CU)+256*PEEK(CU+1):POKES,42:POKES+C,7:G
    OTO770
360 IFHIT<>42THEN320
370 POKESID,0:POKESID+5,9:POKES4,128:POKES4,129:P=
    P+1:N=N+2:FORT=1TO50:NEXT
380 POKES4,128:POKESID,0:POKESID+5,6:POKESID+24,0:
    POKESID+24,15
390 GOSUB880:POKEM,42:POKEM+C,1:POKESID+24,0:POKES
    ID+24,15
400 GOTO320
410 IFTR=1THENPRINT"{CLR}":GOTO470
420 GOSUB950
430 PRINT"{2 DOWN}{3 SPACES}GET THE '*'S BUT":PRIN
    T"{3 SPACES}DON'T HIT ANYTHING ELSE"
```

```
440 PRINT"{2 DOWN}{3 SPACES}USE JOYSTICK IN CONTRO
    L PORT ONE."
450 FORJ=1TO45:POKESID,230:POKES4,33:FORT=1TO2:NEX
    T:POKES4,32:POKESID,Ø
460 POKESID+5,2
47Ø PRINT"{3 DOWN}"TAB(11)"ENTER SKILL LEVEL:"
48Ø PRINTTAB(1Ø)"⁅8⁆{RVS}{9 SPACES}1111111112":
    SK=1Ø
49Ø PRINT" {YEL}SLOW{WHT}{2 SPACES}<- ⁅8⁆{RVS}12
    3456789Ø123456789Ø{OFF}{WHT} ->{2 SPACES}⁅6⁆
    FAST"
5ØØ PRINTTAB(1Ø)"{RVS}{WHT} -{CYN} -{PUR} -{GRN} -
    {YEL} -⁅1⁆ -⁅6⁆ -⁅7⁆ -{BLU} -⁅3⁆ -":PR
    INT
51Ø PRINT"{UP}"TAB(1Ø+SK);"{WHT}↑{LEFT}";
52Ø J=15-(PEEK(56321)AND15):SK=SK+((JAND8)=8)*(SK<
    19)-((JAND4)=4)*(SK>Ø)
53Ø IF(PEEK(56321)AND16)=Ø THEN56Ø
54Ø IF TI<T THEN53Ø
55Ø T=TI+5:PRINT" ":GOTO51Ø
56Ø IFTR=1THENPRINT"{CLR}":GOTO61Ø
57Ø PRINT CHR$(14)"{CLR}{DOWN}YOU WILL GET 2 TIMES
    ":PRINT" AS MANY POINTS WITH"
58Ø PRINT" AN EASY MAZE.
59Ø PRINT"{2 DOWN} YOU WILL GET 5 TIMES":PRINT" AS
     MANY POINTS WITH"
6ØØ PRINT" A HARD MAZE.
61Ø PRINT CHR$(14)"{2 DOWN}⁅8⁆ PRESS {WHT}LEFT
    ⁅8⁆ FOR HARD MAZE"
62Ø PRINT"{DOWN} PRESS {WHT}RIGHT⁅8⁆ FOR EASY MA
    ZE"
63Ø PRINT"{DOWN} PRESS {WHT}JOYBUTTON⁅8⁆ FOR NO
    {SPACE}MAZE"
64Ø IFPEEK(56321)<>255 THEN64Ø
65Ø MZ=Ø:J=PEEK(56321):IF(JAND16)=ØTHENPRINT"{CLR}
    "CHR$(142);:RETURN
66Ø IF(JAND15)=15 THEN65Ø
67Ø PRINT"{CLR}"CHR$(142):IF(JAND4) THEN72Ø
68Ø I=-1:PRINT"{HOME}{RVS}HARD MAZE"
69Ø FORJ=1TODT:POKESC+8Ø+I*32Ø+MA(J)+C,3:POKESC+MA
    (J)+8Ø+I*32Ø,16Ø:NEXTJ
7ØØ I=I+1:IFI<2 THEN69Ø
71Ø MZ=1:RETURN
72Ø IF(JAND8)THEN57Ø
73Ø I=-1:PRINT"{HOME}{RVS}EASY MAZE"
74Ø FORJ=1TO32:POKESC+MA(J)+C+8Ø+32Ø*I,3:POKESC+MA
    (J)+8Ø+32Ø*I,16Ø:NEXT
75Ø I=I+1:IFI<2THEN74Ø
76Ø MZ=2:RETURN
```

```
770 POKESID,Ø:POKESID+5,15:POKES4,129:FORJ=15TO4ST
    EP-.1:POKESID+24,J:NEXT
780 POKESID+24,15:FORT=1TO5ØØ:NEXT:POKES4,128:FORT
    =1TO2ØØ:NEXT:POKESID+5,6
790 IFMZ=1THENP=P*5
800 IFMZ=2THENP=P*2
810 R=P*(SK+1)
820 PRINT"{CLR}{2 DOWN}{YEL} YOUR SCORE:"R
830 IFR>HSTHENHS=R
840 PRINT"{2 DOWN} {CYN}HIGH SCORE:"HS
850 PRINT:PRINT"{WHT}PRESS ⁅3⁆{RVS}JOYBUTTON
    {OFF} {WHT}TO PLAY AGAIN."
860 IF(PEEK(56321)AND16)THEN86Ø
870 GOTO14Ø
880 M=INT(RND(1)*1ØØØ)+SC:MM=Ø
890 IFPEEK(M)<>32THEN88Ø
900 RETURN
910 DATA 259,26Ø,336,337,338,341,342,343,376,383,4
    11,412,413,414,415,416
920 DATA 423,424,425,426,427,428,456,463,496,497,4
    98,5Ø1,5Ø2,5Ø3,579,58Ø
930 DATA 258,259,33Ø,331,332,333,334,345,346,347,3
    48,349,418,419,42Ø,421
940 DATA 49Ø,491,492,493,494,5Ø5,5Ø6,5Ø7,5Ø8,5Ø9,5
    78,581
950 PRINT"{CLR}{WHT} "CHR$(142);:FORI=2TO39:PRINT"
    *";:NEXT:PRINT:PRINT"{4 DOWN}"
960 PRINT" ";:FORI=2TO39:PRINT"*";:NEXT
970 PRINT"{HOME}{DOWN}⁅3 @⁆ ⁅@⁆ ⁅@⁆ ⁅3 @⁆
    {SPACE}⁅2 @⁆ ⁅3 @⁆"
980 PRINT"{2 SPACES}{RVS} £{OFF}£{RVS}£{OFF}£
    {RVS}£{OFF}£{RVS}£{OFF}£{RVS}£{OFF}£
    {RVS}£{OFF}£{RVS}£{OFF}£
990 PRINT" ⁅T⁆{RVS} {OFF}£{RVS}£{OFF}£{RVS}£
    {OFF}£⁅2 T⁆{RVS}£{OFF}£⁅T⁆{RVS}£{OFF}
    £⁅*⁆{RVS}⁅*⁆"
1000 PRINT" ⁅2 T⁆ ⁅T⁆ ⁅2 T⁆{2 SPACES}⁅2 T⁆
     ⁅2 T⁆ ⁅3 T⁆{3 SPACES}":IFZZ=1THEN1Ø7Ø
1010 IFPEEK(9ØØ)<>232THENGOSUB113Ø
1020 FOR CO=3 TO 7:POKE894,CO:SYS893
1030 FORI=1TO2Ø:PRINT"{HOME}{DOWN}"CHR$(148)"
     {DOWN}{LEFT}"CHR$(148)" {DOWN}{LEFT}"CHR$(148
     )" {DOWN}{LEFT}"CHR$(148)" {DOWN}{LEFT}"
1040 POKESID+1,CO*2+I:POKES4,33:POKES4,32:NEXT
1050 FORI=1TO2Ø:PRINT"{HOME}{DOWN} "CHR$(2Ø)"
     {DOWN} "CHR$(2Ø)"{DOWN} "CHR$(2Ø)"{DOWN} "CHR
     $(2Ø)"{DOWN} "
1060 POKESID+1,CO*2+2Ø-I:POKES4,33:POKES4,32:NEXT:
     NEXT
```

```
1070 FORI=1TO10:PRINT"{HOME}{DOWN}"CHR$(148)"
     {DOWN}{LEFT}"CHR$(148)" {DOWN}{LEFT}"CHR$(148
     )" {DOWN}{LEFT}"CHR$(148)" {DOWN}{LEFT}"
1080 NEXT
1090 POKESID+1,60
1100 FORJ=15TO1STEP-1:POKE894,J:POKESID,J*10:POKES
     4,33
1110 SYS893:POKES4,32:POKESID+24,J:NEXT:POKESID+1,
     15:POKESID+24,15
1120 ZZ=1:RETURN
1130 FORI=893TO905:READA:POKEI,A:NEXT
1140 PRINT"{HOME}{8 DOWN}{RVS}READY TO PLAY IN 5 S
     ECONDS..."
1150 DATA 169, 1, 162, 0, 157, 40, 216, 232
1160 DATA 224, 160, 208, 248, 96
1170 FORI=49152TO49350:READA:CK=CK+A:POKEI,A:NEXT
1180 PRINT"{HOME}{8 DOWN}{30 SPACES}"
1190 IF CK<>29203 THEN PRINT"ERROR IN DATA STATEME
     NTS!":POKE900,0:END
1200 RETURN
1210 DATA169,0,141,199,192,173,1,220
1220 DATA41,15,170,189,183,192,240,3
1230 DATA141,199,192,173,201,192,10,170
1240 DATA165,251,157,205,192,165,252,157
1250 DATA206,192,56,173,201,192,237,202
1260 DATA192,16,3,24,105,128,10,170
1270 DATA189,205,192,133,253,189,206,192
1280 DATA133,254,169,32,145,253,238,201
1290 DATA192,173,201,192,16,5,169,0
1300 DATA141,201,192,169,230,141,0,212
1310 DATA169,32,141,4,212,169,33,141
1320 DATA4,212,169,214,145,251,24,165
1330 DATA251,133,253,165,252,105,212,133
1340 DATA254,169,5,145,253,24,173,199
1350 DATA192,16,13,101,251,133,251,165
1360 DATA252,233,0,133,252,76,138,192
1370 DATA101,251,133,251,165,252,105,0
1380 DATA133,252,24,165,251,133,253,165
1390 DATA252,105,212,133,254,177,251,201
1400 DATA32,208,24,169,81,145,251,169
1410 DATA4,145,253,173,200,192,240,8
1420 DATA162,0,134,162,197,162,208,252
1430 DATA76,5,192,141,203,192,96,0
1440 DATA0,0,0,0,41,217,1,0
1450 DATA39,215,255,0,40,216,0,0
```

Chapter 3

Thinking Games

States & Capitals Tutor

Enoch L. Moser

"States & Capitals Tutor," in addition to being a useful tool for students who are learning the American states and capitals, also demonstrates the use of arrays in programs and the storage and retrieval of data on cassette. Both of these concepts are important to programmers, but nonprogrammers may use States & Capitals Tutor without delving into the working details.

"States & Capitals Tutor" asks a student the name of either a state or a capital, and keeps track of correct and incorrect responses. The program randomly decides whether to quiz the student on either states or capitals and also chooses the questions randomly.

Questions answered correctly are not repeated. However, the program *will* repeat questions that are missed. And like any good teacher, States & Capitals Tutor will help students who ask for it. Students who are stumped can simply type HELP. The program gives the correct answer and comes back to the troublesome question later. It also keeps track of how many times the student asks for help.

When all 50 states have been correctly matched with their capitals, and if the student has not asked for help or missed any questions, he or she is rewarded with a perfect score message.

A Two-Part Program

The program reads the states and capitals from a disk or tape file which is generated by Program 2, "File Maker."

To use these programs, follow this procedure:

1. Type in Program 1. Line 5 for tape users should be

   ```
   5 OPEN 1,1,0, "STATES"
   ```

   Line 5 for disk users should be

   ```
   5 OPEN 1,8,0, "STATES"
   ```

2. SAVE Program 1 to disk or tape. Tape users should leave the tape at its position after the SAVE.

3. Type in Program 2 (File Maker). Line 40 for tape users should be

   40 OPEN 1,1,1, "STATES"

   Line 40 for disk users should be

   40 OPEN 1,8,1, "STATES"
4. RUN the program. The states and capitals will be on the tape or disk under the filename STATES. (Tape users should leave the tape in its position.)
5. SAVE Program 2.
6. Rewind the tape.
7. LOAD Program 1; leave the PLAY button pressed and the tape in position when loading is complete.
8. RUN Program 1.

How the Programs Work

As mentioned, the File Maker program stores the states and capitals on tape or disk under the filename STATES. The main program, States & Capitals Tutor, reads this file and stores the data in ST$ (I,J), a *two-dimensional array* (more on this in a moment). When a right answer is given, the range of the random number generator (line 100) is decreased by one (line 205), and that state/capital is moved to the top part of the list (lines 180-200), out of the range of selection. Otherwise, the program is fairly straightforward.

The definitions of the variables are:

ST$ (49,1)	States and capitals array.
K	Number of elements moved to top of list.
R1%	State pointer.
R2%	State or capital selector.
AN$	Answer.
RT%	Number right.
WR%	Number wrong.
HE%	Number of helps.
I$	Temporary string for exchanging data.

Arrays

An *array* is simply an ordered set of data. It may have one or more *dimensions*. A one-dimensional array is merely a list whose data elements are numbered starting with 0. For example, a grocery list of 20 items, numbered 0 to 19, would be a one-dimensional array with 20 data elements.

To define an array, you must use a special type of variable called a *subscripted variable*. This takes the form AN(I), where AN is the Array Name and I is the number (subscript) of the desired element. In our grocery list example, if I=19, then AN(I) would be the last item on the list.

The array name may be any legal variable name, with $ (string variable) or % (integer variable) appended if appropriate. (This would indicate that the data contained in the array are strings or integers.)

Let's say you want a one-dimensional array with four elements. The four elements are integers (whole numbers): 21, 23, 25, and 27. The array would be represented by AN%(I). That is to say, AN%(0)=21, AN%(1)=23, AN%(2)=25, and AN%(3)=27.

A *two-dimensional array* is also an ordered list, but one whose elements are each an ordered list themselves. It's easier to understand if you picture it as a chart. For example, a two-dimensional array might look like this:

	I=0	I=1	I=2	I=3
J=0	21	23	25	27
J=1	43	45	47	49
J=2	51	53	58	59

A proper name for this array could be AN% and its elements identified as AN%(I,J). If I=0 and J=0, then AN%(I,J)=21. If I=3 and J=2, then AN%(I,J)=59. The advantage of arrays is that they let you store lots of numbers or other data without using lots of variables, and you can access any data element with a simple mathematical calculation. But be careful: arrays also consume big chunks of memory.

Arrays can become very complicated. It's easy to picture one- and two-dimensional arrays, but how about arrays of three or even four dimensions? Elements of three-and four-dimensional arrays are identified in the form AN%(I,J,K) and AN%(I,J,K,L), respectively.

Creating Arrays

Typically, arrays are created with nested FOR/NEXT loops, each containing a READ from a DATA statement or an INPUT from a storage device. Each FOR/NEXT level creates one ordered list. For example, the following program could be used to define the contents of the two-dimensional array shown above:

```
10 DIM AN%(3,2)
20 FOR I=0 TO 3
30 FOR J=0 TO 2
40 READ AN%(I,J)
50 NEXT J
60 NEXT I
70 DATA 21,43,51,23,45,53,25,47,58,27,49,59
```

The inner (or nested) FOR/NEXT loop (lines 30-50) creates the ordered list of elements in the J-dimension within each element of the I-dimension. Compare the above chart to the DATA statement in line 70 to see how the array is set up.

The DIMension statement (line 10) is required to tell the computer how much memory to set aside for the array. Note that dimension sizes in a DIMension statement are one less than the number of elements in the dimension. The numbers of dimensions and the number of elements in each dimension are limited only by the amount of memory available.

Remember that an array can hold other types of data besides numbers. States & Capitals Tutor uses a two-dimensional string array, ST$(I,J), to store the 50 states and 50 capitals. See lines 10-35 in Program 2.

Storing Data

Data can be added to a program by using DATA statements or keyboard inputs, or from data files stored on tape or disk. Tape or disk files work best when several programs must have access to the same data, or when a program needs several different data files, or when the amount of data you need to store exceeds memory capacity. Note that when arrays are filled from DATA statements, twice as much memory is required as when they are filled from tape or disk.

Storing and retrieving data is quite simple if you adhere to a few rules. First, before information can be written to or read from a file, a communications channel between the computer and recorder must be opened with the OPEN command. This tells the computer which file is involved and in which direction the information will flow (*input* from the recorder into the computer, or *output* from the computer to the recorder). If a write is indicated in the OPEN command, the computer will write a filename. If a read is indicated, the computer will search for the requested filename and then read the file.

Second, the file must be closed, after use, by the CLOSE command. This is especially important when creating a new file.

The third rule to watch when storing information on tape or disk is that variable types must be consistent. That is to say, data stored as numeric, integer, or string variables must be read back into variables of the same type. The variable names themselves are not stored, so they can be read back into entirely different variables, as long as you don't mismatch types.

Fourth, data is read back in the same order in which it was written. Therefore, the program must expect the data in exactly the same order in which it will be received.

Program 1. States & Capitals Tutor

```
5 OPEN 1,1,Ø,"STATES":REM FOR DISK OPEN 1,8,Ø,"STA
  TES"
1Ø DIM ST$(49,1)
15 FOR I=Ø TO 49
2Ø FOR J=Ø TO 1
25 INPUT#1,ST$(I,J)
3Ø NEXT J
35 NEXT I
4Ø CLOSE 1
45 K=Ø:RT%=Ø:WR%=Ø:HE%=Ø
48 PRINTCHR$(147)
5Ø PRINT"STATES TUTOR"
55 PRINT:PRINT"THIS PROGRAM TUTORS THE STUDENT IN
   {6 SPACES}STATES AND CAPITALS"
6Ø PRINT:PRINT"IF YOU DON'T KNOW AN{2 SPACES}ANSWE
   R,TYPE 'HELP'"
65 PRINT:PRINT"PRESS ANY KEY TO CONTINUE"
7Ø GET A$:IF A$=""THEN7Ø
1ØØ R1%=INT((5Ø-K)*RND(-RND(Ø)))
1Ø5 R2%=INT(2*RND(1))
11Ø PRINTCHR$(147)
115 IF R2%=Ø THEN 13Ø
12Ø PRINT"THE CAPITAL OF ":PRINTST$(R1%,Ø);" IS"
125 GOTO 14Ø
13Ø PRINTST$(R1%,1):PRINT:PRINT"IS THE CAPITAL OF
    {SPACE}WHAT STATE?"
14Ø INPUT AN$
145 IF AN$=ST$(R1%,R2%)THEN17Ø
15Ø IF AN$="HELP"THEN22Ø
155 GOTO 25Ø
17Ø RT%=RT%+1
175 PRINT"THAT'S RIGHT!"
18Ø FOR I=ØTO1
185 I$=ST$((49-K),I)
19Ø ST$((49-K),I)=ST$(R1%,I)
195 ST$(R1%,I)=I$
```

```
200 NEXTI
205 K=K+1
210 GOTO 300
220 HE%=HE%+1
225 PRINT:PRINT"THE ANSWER IS..."
230 PRINTSPC(5)ST$(R1%,R2%)
235 GOTO 300
250 WR%=WR%+1
255 PRINT:PRINT"SORRY.THE CORRECT ANSWER IS "
260 PRINT:PRINTSPC(5)ST$(R1%,R2%)
300 PRINT:PRINT:PRINT:PRINT
305 PRINT"YOUR SCORE IS:"
310 PRINTSPC(5)RT%;" RIGHT"
315 PRINTSPC(5)WR%;" WRONG"
320 PRINTSPC(5)HE%;" HELPS"
325 IF RT%=50THEN400
330 PRINT:PRINT"PRESS ANY KEY TO CONTINUE "
335 GET A$:IF A$=""THEN 335
340 GO TO 100
400 IF WR%+HE%=0THEN430
405 PRINT"THAT'S ALL. BUT NOT ALL YOUR ANSWERS"
406 PRINT"WERE CORRECT OR I HAD TO HELP YOU."
408 PRINT"PRESS ANY KEY TO START OVER"
410 GET A$:IF A$=""THEN410
415 GOTO45
430 PRINT:PRINT"YOU DID IT!!!!!"
435 PRINT"A PERFECT SCORE AND I DIDN'T HELP"
440 PRINT:PRINT"PRESS ANY KEY TO START OVER"
445 GET A$:IF A$=""THEN 445
450 GOTO 45
```

Program 2. File Maker (Data File)

```
10 DIM ST$(49,1)
15 FOR I=0 TO 49
20 FOR J=0 TO 1
25 READ ST$(I,J)
30 NEXTJ
35 NEXTI
40 OPEN 1,1,1,"STATES":REM FOR DISK OPEN 1,8,1,"ST
   ATES"
45 FOR I=0 TO 49
50 FOR J=0 TO 1
55 PRINT#1,ST$(I,J)
60 NEXT J
65 NEXT I
70 CLOSE 1
75 DATA ALABAMA,MONTGOMERY,ALASKA,JUNEAU,ARIZONA,P
   HOENIX,ARKANSAS,LITTLE ROCK
```

```
80 DATA CALIFORNIA,SACRAMENTO,COLORADO,DENVER,CONN
   ECTICUT,HARTFORD,DELAWARE,DOVER
85 DATA FLORIDA,TALLAHASSEE,GEORGIA,ATLANTA,HAWAII
   ,HONOLULU,IDAHO,BOISE
90 DATA ILLINOIS,SPRINGFIELD,INDIANA,INDIANAPOLIS,
   IOWA,DES MOINES,KANSAS,TOPEKA
95 DATA KENTUCKY,FRANKFORT,LOUISIANA,BATON ROUGE,M
   AINE,AUGUSTA,MARYLAND,ANNAPOLIS
100 DATA MASSACHUSETTS,BOSTON,MICHIGAN,LANSING,MIN
    NESOTA,SAINT PAUL,MISSISSIPPI,JACKSON
110 DATA MISSOURI,JEFFERSON CITY,MONTANA,HELENA,NE
    BRASKA,LINCOLN,NEVADA,CARSON CITY
115 DATA NEW HAMPSHIRE,CONCORD,NEW JERSEY,TRENTON,
    NEW MEXICO,SANTA FE,NEW YORK,ALBANY
120 DATA NORTH CAROLINA,RALEIGH,NORTH DAKOTA,BISMA
    RCK,OHIO,COLUMBUS
125 DATA OKLAHOMA,OKLAHOMA CITY,OREGON,SALEM,PENNS
    YLVANIA,HARRISBURG
130 DATA RHODE ISLAND,PROVIDENCE,SOUTH CAROLINA,CO
    LUMBIA,SOUTH DAKOTA,PIERRE
135 DATA TENNESSEE,NASHVILLE,TEXAS,AUSTIN,UTAH,SAL
    T LAKE CITY,VERMONT,MONTPELIER
140 DATA VIRGINIA,RICHMOND,WASHINGTON,OLYMPIA,WEST
     VIRGINIA,CHARLESTON,WISCONSIN,MADISON
145 DATA WYOMING,CHEYENNE
```

3

Mystery Spell

Doug Hapeman 64 Version by Eric Brandon

This spelling game features lively graphics and sprites. It's also a clever teaching aid for parents, teachers, and students in which spelling lessons can be reviewed and then practiced.

If you've ever played Hangman, you won't have any trouble learning "Mystery Spell." Although it's similar in concept, there's a twist. Instead of a gallows, you'll see flying blackbirds, and hear cheerful music.

When the game begins, a happy face appears in a little hut surrounded by trees and landscape. The letters of the alphabet appear near the bottom of the screen, and blank spaces representing the secret word appear near the top. When you select a letter, the bird moves to the selected letter if it's a correct choice. For each incorrect choice, a blackbird descends and lands on a perch. Too many blackbirds disallow any more guesses, and the word will be spelled correctly for you.

The program has 53 preselected words. You can change the words or add to the word list simply by creating your own DATA statements beginning at line 2780. The only restriction is that the last DATA entry must be an asterisk (*).

Animation

The most interesting feature of Mystery Spell is the animated bird. The bird flies around the top of the screen, swooping down to pick up letters and to sit on its perch, depending on whether your guesses are right or wrong.

As the bird moves around, it seems to flap its wings, creating an illusion of flight. This is achieved by rapidly displaying different poses. In films, this is done by passing many frames through a projector every second. To achieve the illusion of flapping wings, we too must create a few frames of a bird in motion.

Using a sprite editor program, we first drew the bird you see in Figure 1. Then, using that sprite, we designed two more birds, one with the wing up (Figure 2) and one with the wing down

(Figure 3). Using those shapes, we designed three more birds identical to the first three, but without legs. This gave us three frames for the bird carrying a letter, and three frames for the bird flying freely. We then set up the DATA statements in the program as if we were going to display six different sprites.

Immediately after the screen RAM are eight memory locations that tell the 64 where in memory to find the shapes of the eight sprites. Usually these locations are at 2040 to 2047 ($07F8 to $07FF). By rapidly POKEing 2040 with the pointer to each frame, the bird seems to flap its wings. To see how this is done, look at lines 2000-2060. This is the routine which flies the bird around the top of the screen until you press a key. Line 2050 steps through the frame numbers. The actual POKEing is done at the end of line 2000.

Another interesting feature of the game is that when you guess correctly, the bird swoops down to pick up a letter, and then carries it up to the word. How is that letter incorporated into the bird sprite?

In the character set ROM at 53248 ($D000), the shape of each character is contained in eight bytes. Each byte is one row, and each bit is a column within that row. Depending on whether the value of that bit is 0 or 1, the pixel will be clear or set inside the character. The sprite is 24 bits wide, which is as wide as three characters. This means that by putting character shape data into every third byte within a sprite, we can make character shapes inside sprites. This technique could be used in any program which moves letters or text around smoothly. To see how this is done, look at lines 2180-2260.

Lines 2180 and 2190 make the character ROM available to be PEEKed. They also turn off the keyboard. Lines 2200 to 2240 take the character data and put it in the sprites. Finally, lines 2250 and 2260 cover up the character ROM and reenable the keyboard.

Figure 1. Sprite-Created Bird

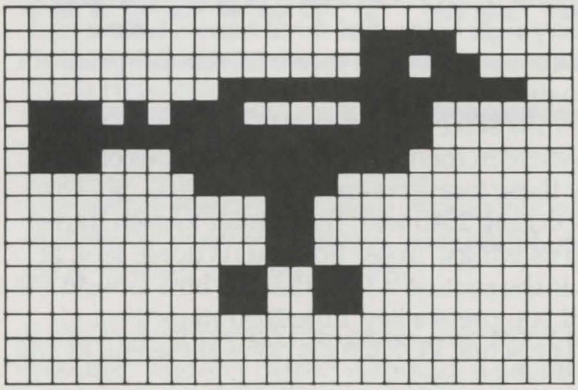


Figure 2. Bird with Wing Up

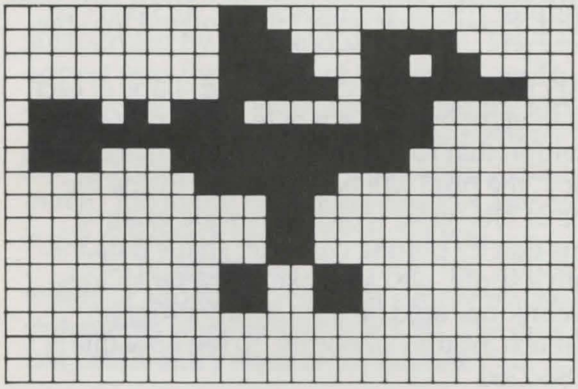
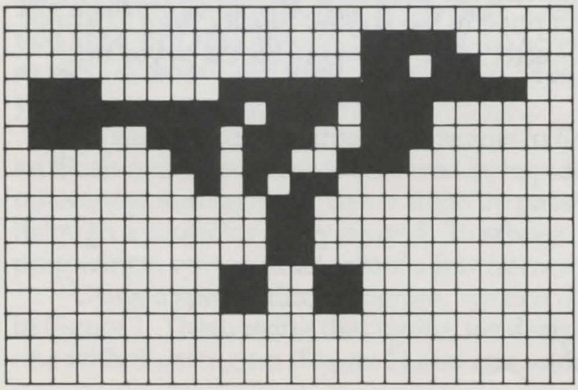


Figure 3. Bird with Wing Down



Mystery Spell

```
100 GOSUB 2660
110 X=RND(-TI)
120 DIM W(20),W$(500)
130 GOSUB 1190 :REM DRAW HOUSE
140 PRINT"{HOME}{BLU}PLEASE WAIT...
150 GOSUB 1380 :REM POKE IN SPRITES
160 GOSUB 1970 :REM GET WORDS
170 GOSUB 690{2 SPACES}:REM SET UP SPRITES
180 PRINT"{HOME}{14 SPACES}"
190 W$=W$(RND(1)*N+1)
200 GOSUB 650
210 L$=" ABCDEFGHIJKLMNOPQRSTUVWXYZ"
220 PRINT"{HOME}{17 DOWN}{8 RIGHT}";
230 FOR I=2 TO 14
240 PRINTMID$(L$,I,1)"{RIGHT}";
250 NEXT
260 PRINT:PRINT"{DOWN}{8 RIGHT}";
270 FOR I=15TO 27
280 PRINTMID$(L$,I,1)"{RIGHT}";
290 NEXT
300 PRINT"{HOME}{4 DOWN}"SPC(18-LEN(G$));
310 FOR I=1TO LEN(G$)
320 PRINTMID$(G$,I,1)"{RIGHT}";
330 NEXT
340 IF COUNT<>LEN(W$) THEN420
350 POKE 198,0
360 FOR DL=1TO100:NEXTDL:CL=CL+1:IFCL=3THENCL=1
370 PRINTMID$("{BLK}{CYN}",CL,1);
380 PRINT"{HOME}{14 SPACES}YOU WIN !!!!"
390 GETA$:IFA$=""THEN 360
400 GOTO 2610
410 GOSUB 2000
420 GETA$:IFA$<"A"ORA$>"Z"ANDA$<>"←"THEN410
430 IF A$="←"THEN 760
440 P=ASC(A$)-64
450 IF MID$(L$,P+1,1)<>" "THEN540
460 PRINT"{HOME}{4 DOWN}{8 SPACES}LETTER ALREADY C
    HOSEN{10 SPACES}"
470 FOR I=1 TO 800:NEXTI
480 PRINT"{HOME}{4 DOWN}{38 SPACES}"
490 PRINT"{HOME}{4 DOWN}"SPC(18-LEN(G$));
500 FOR I=1TO LEN(G$)
510 PRINTMID$(G$,I,1)"{RIGHT}";
520 NEXT
530 GOTO 420
540 L$=LEFT$(L$,P)+" "+MID$(L$,P+2)
550 RF=0 :REM FLAG FOR CORRECT GUESS
```

```
560 FOR I=1 TO LEN(W$)
570 IF MID$(W$,I,1)<>A$ THEN 610
580 G$=LEFT$(G$,I)+MID$(W$,I,1)+MID$(G$,I+2)
590 RF=RF+1
600 COUNT=COUNT+1
610 NEXT I
620 IF RF=0 THEN GOSUB 1030
630 IF RF THEN GOSUB 2070
640 GOTO 220
650 G$=" "
660 FOR I=1 TO LEN(W$):G$=G$+"-":W(I)=0:NEXT
670 RETURN
680 I=I+1:GOTO1980
690 REM SET UP SPRITES
700 V=53248
710 FOR I=0 TO 15:POKE V+I,0:NEXT
720 POKE V+21,255
730 FOR I=V+39 TO V+46:POKE I,0:NEXT
740 X=0:Y=60:S=251
750 RETURN
760 PRINT"{HOME}{BLU}ENTER YOUR GUESS: ";
770 POKE V+21,PEEK(V+21)AND254
780 FOR I=1 TO LEN(W$):PRINT"[@]";:NEXT
790 PRINT"{HOME}{18 RIGHT}";GU$;
800 IF LEN(GU$)<LEN(W$)THENPRINT"[+]";
810 IF LEN(GU$)<LEN(W$)-1 THEN FOR I=2 TO LEN(W$)-
    LEN(GU$):PRINT"[@]";
820 GET K$:IF K$=""THEN 820
830 IF K$=CHR$(20) AND LEN(GU$)>0 THEN GU$=LEFT$(G
    U$,LEN(GU$)-1):GOTO790
840 IF K$=CHR$(13) AND LEN(GU$)=LEN(W$) THEN 870
850 IF K$>="A" AND K$<="Z" AND LEN(GU$)<LEN(W$) TH
    EN GU$=GU$+K$
860 GOTO 790
870 IF GU$<>W$ THEN 930
880 PRINT"{HOME}{38 SPACES}"
890 PRINT"{HOME}{4 DOWN}"SPC(18-LEN(" "+W$));
900 FOR I=1TO LEN(" "+W$)
910 PRINTMID$(" "+W$,I,1)"{RIGHT}";
920 NEXT:GOTO350
930 PRINT"{HOME}{BLK}{13 SPACES}SORRY... YOU LOSE
    {5 SPACES}"
940 PRINT"{BLK}THE WORD WAS ..."
950 PRINT"{HOME}{4 DOWN}"SPC(18-LEN(" "+W$));
960 FOR I=1TO LEN(" "+W$)
970 PRINTMID$(" "+W$,I,1)"{RIGHT}";
980 FOR D=1 TO 200:NEXT
990 NEXT
1000 POKE 198,0
```

```
1Ø1Ø GETA$:IFA$=""THEN1Ø1Ø
1Ø2Ø GOTO 261Ø
1Ø3Ø DB=DB+1:S=S-3
1Ø4Ø DX=32*DB+16:DY=225
1Ø5Ø IF DB=8 THEN DB=Ø
1Ø6Ø POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKE V+1,Y:POKE2Ø4Ø,S
1Ø7Ø IF X=Ø THEN POKE V+21,PEEK(V+21)OR1
1Ø8Ø FLAG=Ø
1Ø9Ø IFABS(X-DX)>1THENX=X+3:FLAG=1:IFX>344THEN X=Ø
     :POKEV+21,PEEK(V+21)AND254
11ØØ IF Y<DY THEN Y=Y+2:FLAG=1
111Ø S=S+1:IFS=251THENS=248
112Ø IF FLAG THEN 1Ø6Ø
113Ø X=DX:Y=DY
114Ø POKEV+2*DB,XAND255:POKEV+16,PEEK(V+16)OR(2↑DB
     )*(-(X>255))
115Ø POKEV+2*DB+1,Y:POKE2Ø4Ø+DB,254
116Ø IF DB<>Ø THEN POKE V+21,PEEK(V+21)AND254
117Ø X=Ø:Y=6Ø:IF DB=Ø THEN 93Ø
118Ø RETURN
119Ø POKE 53281,3:POKE 5328Ø,4
12ØØ PRINT"{CYN}{CLR}
121Ø PRINT"{4 DOWN}
122Ø PRINT
123Ø PRINT"{5 SPACES}{GRN}{3 SPACES}{RVS}
     {2 SPACES}{OFF}{1Ø SPACES}{WHT}⁅D⁆{UP}{RVS}
     ⁅B⁆{OFF}{DOWN}{6 SPACES}{GRN}
124Ø PRINT"{6 SPACES}{RVS}⁅K⁆{4 SPACES}{OFF}
     ⁅J⁆{6 SPACES}{RVS}{YEL}£⁅*⁆{BLK}{OFF}
     ⁅2 G⁆{3 SPACES}{GRN} {RVS}⁅J⁆ ⁅L⁆{OFF}
125Ø PRINT"{6 SPACES}{RVS}⁅J⁆{4 SPACES}⁅L⁆
     {OFF}{5 SPACES}{RVS}{YEL}£{2 SPACES}⁅*⁆
     {OFF}{BLK}⁅G⁆{3 SPACES}{GRN} {RVS}
     {3 SPACES}{OFF}
126Ø PRINT"{6 SPACES}{RVS}⁅G⁆{4 SPACES}⁅N⁆
     {OFF}{4 SPACES}{RVS}{YEL}£{4 SPACES}⁅*⁆
     {OFF}{GRN}{3 SPACES}{RVS}⁅J⁆{3 SPACES}⁅L⁆
     {OFF}
127Ø PRINT"{6 SPACES}{RVS}{6 SPACES}{OFF}
     {4 SPACES}{RVS}{RED}{4 SPACES}⁅£⁆ {OFF}
     {GRN}{3 SPACES}{RVS}{5 SPACES}{OFF}
128Ø PRINT"{6 SPACES}⁅5⁆{2 SPACES}{RVS}
     {2 SPACES}{OFF}{6 SPACES}{RVS}{RED} ⁅£⁆
     {4 SPACES}{OFF}{2 SPACES}{GRN}{3 SPACES}{RVS}
     ⁅5⁆ {OFF}
129Ø PRINT"{RVS}⁅6⁆{8 SPACES}⁅5⁆{2 SPACES}
     ⁅6⁆{6 SPACES}{RED}{2 SPACES}⁅I⁆⁅F⁆
     ⁅£⁆⁅6⁆{5 SPACES}⁅5⁆ ⁅6⁆{12 SPACES}";
```

```
1300 PRINT"{8 SPACES}E5Ǝ{2 SPACES}E6Ǝ
     {6 SPACES}{RED}{2 SPACES}{OFF} {RVS}EKƎ
     {2 SPACES}E6Ǝ{5 SPACES}E5Ǝ E6Ǝ
     {12 SPACES}";
1310 PRINT"E6Ǝ{RVS}";
1320 FOR I=Ø TO 8:PRINT"{40 SPACES}";:NEXT
1330 FOR I=1 TO 8 : L=1Ø24+23*4Ø+I*4 :POKE L,114:P
     OKEL+54272,Ø:NEXT
1340 FOR I=Ø TO 39:POKE 1Ø24+24*4Ø+I,16Ø:POKE 5529
     6+24*4Ø+I,13:NEXT
1350 PRINT"{HOME}
1360 PRINT"{BLK}
1370 RETURN
1380 I=15872:IFPEEK(I+1)=96THENFORI=1TO64*6+2:READ
     A:NEXT:RETURN
1390 READ A:IF A=256 THEN 141Ø
1400 POKE I,A:I=I+1:GOTO 139Ø
1410 FOR I=Ø TO 63:POKE 254*64+I,PEEK(249*64+I):NE
     XT:RETURN
1420 DATA Ø,96,Ø,Ø,113,224,Ø
1430 DATA 121,176,Ø,125,252,117,193
1440 DATA 192,127,255,192,113,255,128
1450 DATA Ø,252,Ø,Ø,24,Ø,Ø
1460 DATA 24,Ø,Ø,1Ø2,Ø,Ø,1Ø2
1470 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1480 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1490 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1500 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1510 DATA Ø,Ø,Ø,Ø,Ø,1,224
1520 DATA Ø,1,176,Ø,127,252,117
1530 DATA 193,192,127,255,192,113,255
1540 DATA 128,Ø,252,Ø,Ø,24,Ø
1550 DATA Ø,24,Ø,Ø,1Ø2,Ø,Ø
1560 DATA 1Ø2,Ø,Ø,Ø,Ø,Ø,Ø
1570 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1580 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1590 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1600 DATA Ø,Ø,Ø,Ø,Ø,Ø,1
1610 DATA 224,Ø,1,176,112,127,252
1620 DATA 127,221,192,115,185,192,1
1630 DATA 179,128,Ø,172,Ø,Ø,24
1640 DATA Ø,Ø,24,Ø,Ø,1Ø2,Ø
1650 DATA Ø,1Ø2,Ø,Ø,Ø,Ø,Ø
1660 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1670 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1680 DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø
1690 DATA Ø,Ø,Ø,Ø,96,Ø,Ø
1700 DATA 113,224,Ø,121,176,Ø,125
1710 DATA 252,117,193,192,127,255,192
```

```
1720 DATA 113,255,128,0,252,0,0
1730 DATA 0,0,0,0,0,0,0
1740 DATA 0,0,0,0,0,0,0
1750 DATA 0,0,0,0,0,0,0
1760 DATA 0,0,0,0,0,0,0
1770 DATA 0,0,0,0,0,0,0
1780 DATA 0,0,0,0,0,0,0
1790 DATA 0,1,224,0,1,176,0
1800 DATA 127,252,117,193,192,127,255
1810 DATA 192,113,255,128,0,252,0
1820 DATA 0,0,0,0,0,0,0
1830 DATA 0,0,0,0,0,0,0
1840 DATA 0,0,0,0,0,0,0
1850 DATA 0,0,0,0,0,0,0
1860 DATA 0,0,0,0,0,0,0
1870 DATA 0,0,0,0,0,0,0
1880 DATA 0,0,1,224,0,1,176
1890 DATA 112,127,252,127,221,192,115
1900 DATA 185,192,1,179,128,0,172
1910 DATA 0,0,112,0,0,0,0
1920 DATA 0,0,0,0,0,0,0
1930 DATA 0,0,0,0,0,0,0
1940 DATA 0,0,0,0,0,0,0
1950 DATA 0,0,0,0,0,0,0
1960 DATA 0,0,0,0,0,0,0,256
1970 I=1
1980 READ W$(I):IFW$(I)="*"THENN=I-1:RETURN
1990 I=I+1:GOTO1980
2000 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKE V+1,Y:POKE2040,S
2010 IF X=0 THEN POKE V+21,PEEK(V+21)OR1
2020 X=X+3:IFX>344 THEN X=0:POKEV+21,PEEK(V+21)AND
     254
2030 Y=Y-1+RND(1)*2:IFY>100THENY=99
2040 IF Y<50 THEN Y=50
2050 S=S+1:IFS=254THENS=251
2060 RETURN
2070 DX=INT(P+13*(P>13))*16+24+40
2080 DY=173+INT(P/14)*24:IF S>250 THEN S=S-3
2090 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKEV+1,Y:POKE2040,S
2100 IF X=0 THEN POKE V+21,PEEK(V+21)OR1
2110 FLAG=0
2120 IFABS(X-DX)>2THENX=X+3:FLAG=1:IFX>344THENX=0:
     POKEV+21,PEEK(V+21)AND254
2130 IF Y<DY THEN Y=Y+2:FLAG=1
2140 S=S+1:IFS=251THENS=248
2150 IF FLAG THEN 2090
2160 X=DX:Y=DY
```

```
2170 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKEV+1,Y:POKE2040,249
2180 POKE 56334,PEEK(56334)AND254
2190 POKE 1,PEEK(1)AND251
2200 FOR I=0 TO 7
2210 B=PEEK(53248+8*P+I)
2220 FOR J=248 TO 250
2230 POKE J*64+40+I*3,B
2240 NEXT J,I
2250 POKE 1,PEEK(1)OR4
2260 POKE 56334,PEEK(56334)OR1
2270 PRINT"{HOME}{17 DOWN}{8 RIGHT}";
2280 FOR I=2 TO 14
2290 PRINTMID$(L$,I,1)"{RIGHT}";
2300 NEXT
2310 PRINT:PRINT"{DOWN}{8 RIGHT}";
2320 FOR I=15TO 27
2330 PRINTMID$(L$,I,1)"{RIGHT}";
2340 NEXT
2350 DX=160-8*LEN(G$):DY=69
2360 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKEV+1,Y:POKE2040,S
2370 IF X=0 THEN POKE V+21,PEEK(V+21)OR1
2380 FLAG=0
2390 IFABS(X-DX)>2THENX=X+3:FLAG=1:IFX>344THEN X=0
     :POKEV+21,PEEK(V+21)AND254
2400 IF Y>DY THEN Y=Y-2:FLAG=1
2410 S=S+1:IFS=251THENS=248
2420 IF FLAG THEN 2360
2430 X=DX:Y=DY
2440 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKEV+1,Y:POKE2040,249
2450 PRINT"{HOME}{4 DOWN}"SPC(18-LEN(G$));
2460 FOR I=1TO LEN(G$)
2470 IF MID$(G$,I,1)=A$ THEN PRINT A$;:RF=RF-1:IFR
     F=0 THEN GOSUB 2560
2480 IF MID$(G$,I,1)<>A$ THEN PRINT"{RIGHT}";
2490 PRINT"{RIGHT}";
2500 IF RF=0 THEN I=100:GOTO2540
2510 FOR J=0 TO 15:X=X+1:S=S+1:IFS=251THENS=248
2520 POKEV,XAND255:POKEV+16,PEEK(V+16)AND254OR-(X>
     255):POKE2040,S
2530 NEXT J
2540 NEXT I
2550 RETURN
2560 FOR K=0 TO 7
2570 FOR J=248 TO 250
2580 POKE J*64+40+K*3,0
2590 NEXT J,K
```

```
2600 RETURN
2610 PRINT"{CLR}{7 DOWN}{BLK}DO YOU WISH TO PLAY A
     GAIN (Y/N) ?"
2615 POKE V+21,PEEK(V+21)AND254
2620 PRINT"{10 DOWN}YOU MISSED THIS MANY :"
2630 GETA$:IFA$<>"N"AND A$<>"Y"THEN2630
2640 IF A$="Y"THENPOKE V+21,0:RUN110
2650 END
2660 POKE 53281,0:POKE 53280,0
2670 PRINT"{CLR}{YEL}{13 SPACES}INSTRUCTIONS
2680 PRINT"{2 DOWN}{WHT}{4 SPACES}CHOOSE LETTERS T
     O GUESS THE WORD.
2690 PRINT"{DOWN}IF YOU CHOOSE A WRONG LETTER, THE
      BIRD
2700 PRINT"{DOWN}WILL LAND ON ITS PERCH.
2710 PRINT"{DOWN}{4 SPACES}WHEN ALL THE PERCHES AR
     E FULL, OR
2720 PRINT"{DOWN}YOU GUESSED THE WORD, THE GAME IS
      OVER
2730 PRINT"{2 DOWN}{4 SPACES}YOU CAN HIT THE "CHR$
     (34)"←"CHR$(34)" KEY ANYTIME TO
2740 PRINT"{DOWN}GUESS THE WORD. IF YOU GET IT WRO
     NG,{DOWN}{4 SPACES}YOU LOSE.
2750 PRINT"{3 DOWN}{9 RIGHT}{YEL}HIT A KEY TO BEGI
     N"
2760 GETA$:IFA$=""THEN2760
2770 RETURN
2780 DATA HAPPY,BRIDGE,FAMILY,CHILDREN
2790 DATA WINDOW,TRAIN,DWARF,BIRDS
2800 DATA SUPERMAN,CONCERT,PEOPLE,MAGIC
2810 DATA SPACE,SCIENCE,PLANETS,GALAXY,STARS
2820 DATA ROOMS,TEACHER,CHALK,BLACKBOARD
2830 DATA SCREEN,COMPUTER,KEYBOARD,PROGRAM
2840 DATA SPELLING,WORDS,COLORS,LETTERS
2850 DATA MARKET,STREETS,SQUARE,TRIANGLE
2860 DATA MOVIE,SPACESHIP,LASER,AIRPLANE,BOAT
2870 DATA STICK,ROCK,PAPER,WIN,PLACE,SHOW
2880 DATA CHANNEL,EXECUTIVE,MONEY,SHIRT
2890 DATA QUIET,LOUD,BILLBOARD,YACHT,MOTORCYCLE,*
```

3

Oil Tycoon

Gordon F. Wheat 64 Translation by Chris Metcalf

You are P. J. Uing and you are about to make big money in the petroleum business, but drilling for oil is not as easy as it sounds. There are obstacles you must overcome in order to make a profit. There are shale formations that grind away your pipe. You can blast through them, but your dynamite is limited. Pockets of natural gas sometimes collect where you have previously pumped out the oil. Hit one of these and your oil rig goes up with a bang. There are also "devils" that live in the oil. They take a dim view of your draining their caverns. But you won't give up—because you are the Oil Tycoon.

I designed "Oil Tycoon" to be as much fun for parents as it will be for children. Since the game is not based on reaction time but rather on strategy, it helps even the score for the arcade drop-outs. Your strategy will slowly build, and before long you will be rolling in cash or attaining high scores, however you wish to look at it.

Difficulty Levels

The screen will display the high scores attained for each of the eight difficulty levels. The program will return to this screen after each game. Your score and the difficulty level of the game you have just completed are displayed at the top of the screen.

At the bottom of the screen you will see "DIFF . LEVEL 12345678." Choose the difficulty level by moving the joystick left and right and pressing the fire button when the number of the difficulty level you want is blinking. Level one is primarily for small children. I would recommend that seasoned gamers begin with level two. The higher the difficulty level, the more difficult the game becomes. The various conditions for the eight difficulty levels are shown in the table.

Difficulty Levels

Level	Sticks of Dynamite per Oil Rig	Pieces of Shale	Invisible Shale
1	3	20	No
2	2	20	No
3	3	30	No
4	2	30	No
5	4	20	Yes
6	3	20	Yes
7	4	30	Yes
8	3	30	Yes

Playing Oil Tycoon

After you choose the level, the oil field is drawn on the screen. It will be different for each game; you should never see the same screen twice. For each game, you receive five oil rigs, each of which has 20 lengths of pipe and a number of sticks of dynamite, depending on the difficulty level you choose.

In the upper-left corner of the screen are the oil rigs you have remaining. In the upper-right corner is your score. Between these are the sticks of dynamite you have remaining for the oil rig now in play. The second line displays the unused lengths of pipe for the oil rig now in play. As you drill, this pipe will be used one length at a time and will be replaced as you withdraw your drill. The lower portion of the screen is the playing field. Yellow squares are dirt, black squares are oil, and the irregular squares are shale.

Move the joystick left and right to position your oil rig over the column you want to drill through. To drill, pull the joystick down. To withdraw the drill, push the joystick up. You cannot move the oil rig while there is drilling pipe in the ground. You cannot bore through shale, through devils, or off the bottom of the screen. If you try, your drill will be ground up, and you will lose that length of pipe for the oil rig in play. This becomes very important in difficulty levels above four, for the shale is invisible and looks like dirt. At these levels, it is very easy to lose most of your drilling pipe before you realize that you are trying to drill through shale. Also try to avoid drilling through empty spaces from which you have previously pumped oil. Natural gas can collect in these empty spaces and may cause an explosion when you try to drill through them again.

Controlling the fire button takes some getting used to, because it does three things. As you bore, if the end of the drilling pipe is in oil or an empty space, pressing the fire button causes your oil rig to start pumping. If the end of the pipe is in dirt, pressing fire drops a stick of dynamite down the pipe. If you are not drilling, or if you have fully withdrawn the pipe, pressing fire replaces your current oil rig with one of your remaining rigs. Be careful—it is easy to lose valuable rigs. Replacing your oil rig with a new one is useful mainly when you have used up your allotted dynamite for the rig in play, or if you do not have enough pipe remaining to reach pools of oil near the bottom of the screen.

Use your dynamite to blow up shale, devils, or dirt. When you drop dynamite down the pipe, it will continue to fall until it hits one of these three obstacles. This means that if there is oil or empty space directly below the tip of the drill, the dynamite will fall out of the bottom of the pipe and through this space until it hits shale, a devil, or dirt.

Pumping Oil

When you pump, all of the oil in adjacent spaces to the sides and above the level of the drill bit will be pumped out. In other words, all squares of oil connected to the one you are pumping will also be pumped out only if they lie directly *above* or *to the sides* of the oil being pumped. Any squares of oil *below* those which are being pumped out will remain where they are.

If you uncover a devil while pumping, it will blow up your oil rig. If you try to pump a pool of oil which is at or below the level of an uncovered devil, and which is directly connected to the devil's space, it will also blow up your rig.

The deeper the oil, the more it is worth when you pump it out. An extra oil rig is awarded for each $100,000 you acquire. In addition, if you pump out all the oil on the screen and then retract your pipe, you will be awarded an extra oil rig and a new screen is drawn.

Oil Tycoon

```
100 PRINT"{CLR}⁅7⁆":IFPEEK(14336)=2ANDPEEK(14805
    )=24THEN195
105 POKE53280,6:POKE53281,6:POKE53270,8
110 PRINTTAB(14)"INSTRUCTIONS"SPC(28)"⁅12 T⁆":PR
    INT"{DOWN}JOYSTICK:"
115 PRINT"{DOWN} RIGHT AND LEFT = MOVE RIG"
120 PRINT" DOWN = DRILL":PRINT" UP = RETRACT PIPE"
```

```
125 PRINT"{2 DOWN}{2 SPACES}WHEN YOU PUSH THE FIRE
     BUTTON AND THE
130 PRINT"PIPE IS DOWN IN OIL OR IN SPACE, THE"
135 PRINT"PUMP IS TRIGGERED.{2 SPACES}IF THE PIPE
    {SPACE}IS DOWN"
140 PRINT"IN DIRT, DYNAMITE IS DROPPED.
145 PRINT"{2 DOWN}WATCH OUT FOR SHALE AND GAS IN E
    MPTY","SPACES AND DEVILS IN OIL.
150 PRINT"{2 DOWN}PLEASE {CYN}WAIT{7} FOR FURTHE
    R INSTRUCTIONS."
155 POKE52,56:POKE56,56:CLR:AD=14336
160 FORA=ADTOAD+207:READB:POKEA,B:NEXT:POKE56334,P
    EEK(56334)AND254:POKE1,51
165 FORA=AD+256TOAD+471:POKEA,PEEK(38912+A):NEXT:P
    OKE1,55
170 POKE56334,PEEK(56334)OR1
175 PRINT"{UP}{2 SPACES}PRESS ANY KEY WHEN READY T
    O BEGIN. "
180 IFPEEK(197)=64ANDPEEK(653)=ØANDPEEK(5632Ø)=127
    THEN18Ø
185 :
190 :
195 PRINT"{CLR}":POKE5328Ø,6:POKE54296,15:DIMA%(4Ø
    ):W=1184:JS=5632Ø
200 POKE53282,6:POKE53283,Ø:POKE5327Ø,24:POKE54291
    ,Ø:POKE54292,24Ø
205 FORI=ØTO2:POKE54276+I*7,8:NEXT:POKE53281,3
210 POKE54284,Ø:POKE54285,24Ø:POKE54277,Ø:POKE5427
    8,24Ø:IFZ>B%(T)THENB%(T)=Z
215 POKE53272,21:PRINT"{CLR}{DOWN}{RED}",T,"
    {2 SPACES}$"MID$(STR$(Z*1ØØ),2)".ØØ{BLU}"
220 PRINTTAB(8)"{DOWN} LEVEL{6 SPACES}HIGH SCORE
    {DOWN}"
225 FORA=1TO8:PRINT,A,"{2 SPACES}$"MID$(STR$(B%(A)
    *1ØØ),2)".ØØ":PRINT:NEXT
230 PRINT"{DOWN} DIFFICULTY LEVEL? 12345678{GRN}":
    T=1
235 POKE56194+T,Ø:T1=T:T=T+(PEEK(JS)AND4)/4-(PEEK(
    JS)AND8)/8:T=(7ANDT-1)+1
240 IFT<>T1THENPOKE56194+T1,6
245 POKE56194+T,1:L=3:IFT/2=INT(T/2)THENL=2
250 S=2Ø:IFT=3ORT=4ORT>6THENS=3Ø
255 N=24:IFT>4THENN=25:L=L+1
260 GETA$:IF(PEEK(5632Ø)AND16)=16ANDA$<>CHR$(13)TH
    EN235
265 POKE53272,31:PRINT"{CLR}":POKE5328Ø,9:POKE5328
    1,1:M=4:Z=Ø:K=Ø:GOSUB59Ø
270 POKEW+X,14:X=2Ø:P=2Ø:Y=L:R=1:GOSUB7Ø5:GOSUB645
    :POKE198,Ø
```

```
275 :
280 :
285 REM MAIN LOOP OF PROGRAM
290 A=PEEK(JS):IF(AAND4)=ØANDR=1THENPOKEW+X,14:X=X
    +(X>Ø)
295 IF(AAND8)=ØANDR=1THENPOKEW+X,14:X=X-(X<39)
300 POKEW+X,2:IF(AAND2)=ØANDP>ØTHEN34Ø
305 IF(AAND1)=ØANDR>1THEN4ØØ
310 IF(AAND16)=ØTHEN435
315 GETA$:IFR=1ANDA$=" "THEN375
320 GOTO29Ø
325 :
330 :
335 REM DRILLING AND GAS EXPLOSIONS
340 A=R*4Ø+W+X:C=PEEK(A):P=P-1:GOSUB675
345 IFC=NORC=3ORA>2Ø23THENFORA=1TO3:GOSUB73Ø:NEXT:
    GOTO29Ø
350 IFRND(1)>.Ø6ORC<>14THENFORB=1TO3:POKEA,C+B:GOS
    UB73Ø:NEXT:R=R+1:GOTO29Ø
355 FORB=1TO2:POKEA,C+B:GOSUB73Ø:NEXT:GOSUB735:B=Ø
360 R=R-1:POKE54296,4:IFR<1THENPOKE54296,15:POKEW+
    X,23:GOSUB715:GOTO375
365 POKE54273,B:POKE54276,129:A=R*4Ø+W+X:C=PEEK(A)
    :PK=PEEK(A+54272):POKEA,C+1
370 POKEA+54272,15:FORD=ØTO2ØØ:NEXT:POKEA,C-3:POKE
    A+54272,PK:B=B+1Ø:GOTO36Ø
375 POKEW+X,14:X=2Ø:M=M-1:P=2Ø:Y=L:R=1:GOSUB7Ø5:IF
    M<ØTHEN2Ø5
380 GOSUB645:GOTO29Ø
385 :
390 :
395 REM DRILLING UP
400 R=R-1:B=R*4Ø+W+X:C=PEEK(B):FORA=1TO3:POKEB,C-A
    :GOSUB73Ø:NEXT
405 P=P+1:GOSUB675:IFR<>1THEN29Ø
410 FORA=W+8ØTO2Ø23:IFPEEK(A)=9THEN29Ø
415 NEXT:M=M+1:FORC=1TO3:GOSUB7Ø5:NEXT:GOSUB59Ø:GO
    SUB645:GOTO29Ø
420 :
425 :
430 REM DYNAMITE, GUSHERS, DEVILS
435 J=Z:Q=R-1:FORA=ØTO21:A%(A)=Ø:NEXT:B=Q*4Ø+W+X:A
    =PEEK(B):IFA<>7THEN48Ø
440 A=W+X:B=4Ø:IFY<1THEN29Ø
445 A=A+4Ø:C=PEEK(A):POKEA,C+1:IFC=14ORC=9THENPOKE
    A,C+4
450 POKE54273,B:POKE54276,33:FORD=ØTO2ØØ:NEXT:D=PE
    EK(A+4Ø)
455 IFD<>4ANDD<>NANDD<>3ANDA<1984THENB=B-2:POKEA,C
    :GOTO445
```

```
460 POKE54276,8:GOSUB735:POKEA+54312,15
465 IFC<>14ANDC<>9THENR=R-1
470 Y=Y-1:GOSUB665:GOTO290
475 :
480 A%(X)=1:POKE54273,40:POKE54276,129:POKE54296,4
    :V=W+X-40
485 IFA=12THENPOKEB,17:Z=Z+Q:POKEV,0
490 E=0:F=38:D=1:G=1:I=1:GOSUB530:POKEV,1:E=39:F=1
    :D=-1:G=D:I=D:GOSUB530
495 E=0:F=39:D=1:G=-40:I=0:GOSUB530:POKEV,0:IFC<>6
    THEN515
500 Z=J:POKEB,3:POKEB+54272,2:POKEV,14:FORA=0TO40:
    POKE54280,88:POKE54283,17
505 POKE53283,14:POKE54296,6:FORB=1TO5:NEXT
510 POKE54296,0:POKE53283,0:FORB=1TO5:NEXTB,A:POKE
    54283,0:B=0:GOTO360
515 IFHTHENQ=Q-1:GOTO490
520 POKEV,14:POKE54276,8:POKE54296,15:POKE54283,2:
    GOSUB705:GOSUB645:GOTO290
525 :
530 IFC=6THENRETURN
535 H=0:FORA=ETOFSTEPD:IFA%(A)=0THEN570
540 B=Q*40+W+A+G:C=PEEK(B)
545 IFC=9ORC=12THENPOKEB,C+5:H=1:Z=Z+Q+ABS(I)-1:A%
    (A+I)=1:GOTO565
550 IFC=14ORC=17THENA%(A+I)=1:H=1:GOTO570
555 IFC=3THENC=6:RETURN
560 A%(A+I)=0:GOTO570
565 IFRND(1)<.02ANDC<>12THENC=6:RETURN
570 NEXT:RETURN
575 :
580 :
585 REM INITIALIZE THE DISPLAY
590 PRINT"{HOME}{5 DOWN}⁅8⁆";:FORA=1TO99:PRINT"D
    DDDDDDD";:NEXT:PRINT"DDDDDDD";
595 POKE2023,4:POKE56295,15:B=400:C=1264:FORA=1TO2
    :FORD=1TO40
600 E=INT(RND(0)*B/2)*2+C:IFPEEK(E)=9ORPEEK(E+1)=9
    THEN600
605 POKEE,9:POKEE+1,9:NEXT:B=360:C=1665:NEXT:FORA=
    1TOS
610 B=INT(RND(1)*340)*2+1264:C=PEEK(B):IFC=9ORC=NT
    HEN610
615 POKEB,N:POKEB+54272,10:NEXT:FORA=0TO199:POKE55
    296+A,0:NEXT:FORA=0TO39
620 POKE55376+A,3:POKE1104+A,20:NEXT
625 FORA=0TO3:POKE55337+A,2:NEXT:RETURN
630 :
635 :
```

```
640 REM UPDATE SCREEN INFORMATION
645 PRINT"{HOME}{GRN}"SPC(23)"$"MID$(STR$(Z*100),2
    )".00"
650 A=INT(Z/1000):IFA=K+1THENK=K+1:GOSUB705:GOSUB7
    05:M=M+1
655 IFM<1THENPOKE1024,14:POKE55296,14:GOTO665
660 FORA=1024TO1023+M:POKEA,2:POKEA+54272,0:NEXT:P
    OKEA,14:POKEA+54272,0
665 IFY=0THENPOKE1031,14:POKE55303,0:GOTO675
670 FORA=1031TO1030+Y:POKEA,19:POKEA+54272,0:NEXT:
    POKEA,14:POKEA+54272,0
675 IFP<1THENPOKE1064,14:POKE55336,0:RETURN
680 FORB=1064TO1063+P:POKEB,17:POKEB+54272,0:NEXT:
    POKEB,14:POKEB+54272,0
685 RETURN
690 :
695 :
700 REM MUSIC AND OTHER SUBROUTINES
705 POKE54276,17:FORA=15TO0STEP-1:POKE54296,A:POKE
    54273,86:FORB=1TO25:NEXTB,A
710 POKE54276,8:POKE54296,15:RETURN
715 POKE54276,8:POKE54276,129:POKE54273,91:FORD=15
    TO0STEP-1:POKE54296,D
720 POKE53281,1:POKE53280,2:FORE=1TO70:NEXT:POKE53
    280,6:NEXT
725 POKE54276,8:POKE54296,15:POKE53280,9:RETURN
730 POKE54287,20:POKE54290,8:POKE54290,129:POKE542
    90,128:RETURN
735 POKEA,21:POKEA+40,22:GOSUB715:POKEA,14:POKEA+4
    0,14:RETURN
740 :
745 :
750 REM CHARACTER DATA
755 DATA2,138,164,73,74,52,20,8,64,81,37,146,82,44
    ,40,16,24,24,36
760 DATA60,90,102,231,153,20,42,42,20,62,73,20,20,
    136,34,136,34,136,34,136,34
765 DATA148,22,148,34,136,34,136,34,148,22,148,22,
    148,34,136,34,148,22,148
770 DATA22,148,22,148,22,136,62,188,62,188,62,188,
    22,170,170,170,170,170
775 DATA170,170,170,150,150,150,170,170,170,170,17
    0,150,150,150,150,150,170,170
780 DATA170,150,150,150,150,150,150,150,150,150,19
    0,190,190,190,190,190,150,0,0
785 DATA0,0,0,0,0,0,20,20,20,0,0,0,0,0,20,20,20,20
    ,20,0,0,0,20,20,20,20,20,20
790 DATA20,20,20,60,60,60,60,60,60,20,0,60,60,60,6
    0,60,60,0,0,0,0,255,255,0,0,0
```

```
795 DATA218,118,181,153,110,93,197,65,65,82,150,85
    ,121,181,150,173,2,106,129,20
800 DATA64,162,129,2,169,128,141,19,145,169,0,133,
    136,34,136,34,136,34,136,34
```

3

Mosaic Puzzle

Bruce Jordan 64 Translation by Chris Metcalf

This adaptation of an old favorite will challenge your reasoning powers.

"Mosaic Puzzle" is a computer version of those sliding-squares puzzles that used to drive people nuts before the advent of Rubik's Cube. The object of the game is to arrange the 15 squares into some predetermined order by sliding them around in their frame. The first few moves are easy, but as the game progresses, it gets a lot more complicated. You'll find yourself rearranging everything just to get the last few squares in place.

The game has a timer for up to 23 hours, 59 minutes, 59 seconds, and a chicken switch. It also automatically checks for the winning order and allows you to go back to the puzzle the way you left it or reset it to the beginning arrangement.

When you start the game, you're asked if you wish to set a time limit. If you answer Y for yes, enter the time limit in one line with no spaces or punctuation between the values. For example, for a 1-hour, 23-minute limit, enter 012300.

Next, enter the goal order. This will be the order that you will try to match to win the game. When this is done, the upper half of the screen will clear, and the puzzle will appear.

Either the RETURN key or the fire button allows you to pause momentarily before resuming the game, restarting the program, or stopping play entirely. Breaking off and resuming has no effect on the time clock (displayed at the top of the screen along with the time limit).

As an aid to the user, various keys for up, down, right, and left can be selected at the beginning of the game. A joystick can also be used, as long as it is plugged into control port two. The time limit is an option in this version; if no time limit is selected, the screen will display elapsed time and TIME LIMIT: NONE.

If you succeed in getting the squares in the goal order, the message YOU WIN! appears on the screen, accompanied by a short tune and the elapsed time. If the time runs out before you are finished, you'll hear an unpleasant sound.

Mosaic Puzzle

```
100 POKE53280,14:POKE53281,6:POKE55,176:POKE56,29:
    CLR:POKE54276,8:POKE54283,8
110 POKE54277,0:POKE54278,255:POKE54284,0:POKE5428
    5,255:POKE54296,15
120 S=1355:SC=S+54272:DIMA$(16)
130 PRINT"{CLR}":G=1632:X=0:DX=1:P=55904:S1=54276:
    S2=54283:AD=1232:R=14
140 PRINT"{CLR}{DOWN}"TAB(11)" MOSAIC PUZZLE"TAB(5
    0)"[17 Y]{DOWN}"
150 :
160 :
170 REM FIND TIME LIMIT, MOVE KEYS
180 PRINT"[7] DO YOU WANT A TIME LIMIT? ";:GOSUB
    270
190 IFIN$<>"Y"THEN240
200 H=1:INPUT"{HOME}{6 DOWN} HOURS MINS SECS (6 DI
    GITS)";T$:IFLEN(T$)<>6THEN200
210 IFLEFT$(T$,2)>"23"ORLEFT$(T$,2)<"0"THEN200
220 IFMID$(T$,3,2)>"59"ORMID$(T$,3,2)<"0"THEN200
230 IFRIGHT$(T$,2)>"59"ORRIGHT$(T$,2)<"0"THEN200
240 PRINT"{DOWN} KEY FOR UP: ";:GOSUB270:U$=IN$:PR
    INT"{DOWN} FOR DOWN: ";:GOSUB270:D$=IN$
250 PRINT"{DOWN} FOR LEFT: ";:GOSUB270:L$=IN$:PRIN
    T"{DOWN} FOR RIGHT: ";:GOSUB270:R$=IN$
260 GOTO310
270 PRINT"[+]";:WAIT198,255:GETIN$:PRINT"{LEFT}"
    ;:POKE216,1:PRINTIN$:RETURN
280 :
290 :
300 REM FIND GOAL ORDER
310 PRINT"{CLR}"TAB(43)"ENTER GOAL SETUP"
320 PRINT"{DOWN}{3 SPACES}1 2 3 4 5 6 7 8 9"SPC(23
    )"A B C D E F {RVS}SPACE"
330 PRINTTAB(5)"{DOWN}IN ANY ORDER":PRINTTAB(248)"
    GOAL
340 FORK=0TO3:POKE1592+K,100:POKE1792+K,99:POKE558
    64+K,R:POKE56064+K,R
350 POKE1631+K*40,103:POKE1636+K*40,1?01:POKE55903+
    K*40,R:POKE55908+K*40,R:NEXT
360 FORI=1TO16:POKEG+X,63:POKEP+X,1
370 WAIT198,255:GETA$(I):FORL=I-1TO0STEP-1:IFA$(I)
    =A$(L)THEN370
380 NEXT:IFA$(I)=" "THENFORK=0TO4:POKE55471+K,15:N
    EXT:B2=32:GOTO420
390 IF(A$(I)<"1"ORA$(I)>"F")OR(A$(I)>"9"ANDA$(I)<"
    A")THEN370
400 B=VAL(A$(I)):B2=B+48:IFBTHENPOKE55417+2*B,15:G
    OTO420
```

```
410 B=ASC(A$(I))-64:B2=B:POKE55457+2*B,15
420 POKEG+X,B2:X=X+DX:IFX=4THENG=G+40:P=P+40:X=0
430 NEXT
440 :
450 :
460 REM SET UP WORK AREA
470 PRINT"{HOME}":FORI=0TO64:PRINT"{4 SPACES}";:NE
    XT:PRINT"{HOME}"TAB(127)"PUZZLE"
480 FORK=0TO3:POKE1192+K,100:POKE55464+K,R:POKE139
    2+K,99:POKE55664+K,R
490 POKE1231+K*40,103:POKE55503+K*40,R:POKE1236+K*
    40,101:POKE55508+K*40,R:NEXT
500 READA,B,C:IFA>=0THENPOKEAD+A,B:POKE55504+A,C:G
    OTO500
510 FORI=1TO500:NEXT:POKES1-3,80:POKES1,33:PRINT"
    {HOME}"TAB(28)"{10 DOWN}{RED}{WHT}!GO!⁅7⁆"
520 FORT=1TO300:NEXT:PRINT"{HOME}"TAB(28)"
    {10 DOWN}{4 SPACES}":POKES1,8:TI$="000000"
530 PRINT"{HOME}"TAB(25)"LIMIT:{CYN}";:IFT$=""THEN
    PRINT"NONE":GOTO580
540 PRINTLEFT$(T$,2)":"MID$(T$,3,2)":"RIGHT$(T$,2)
    "⁅7⁆"
550 :
560 :
570 REM LOOP MAIN CONTROL
580 PRINT"{HOME}TIME ELAPSED:{WHT}"LEFT$(TI$,2)":"
    MID$(TI$,3,2)":"RIGHT$(TI$,2)"⁅7⁆"
590 IFH=1ANDT$<=TI$THEN750
600 GETB$:J=31-PEEK(56320)AND31:IFB$=""ANDJ=0THEN5
    80
610 IFB$=CHR$(13)ORJ=16THENWN=0:GOTO780
620 IFB$=D$OR(JAND2)THENDR=-40:CK=100:GOTO660
630 IFB$=L$OR(JAND4)THENDR=1:CK=101:GOTO660
640 IFB$=R$OR(JAND8)THENDR=-1:CK=103:GOTO660
650 DR=40:CK=99:IFB$<>U$AND(JAND1)=0THEN580
660 IFPEEK(S+DR)=CKTHEN580
670 POKES,PEEK(S+DR):POKESC,PEEK(SC+DR):POKES+DR,3
    2:S=S+DR:SC=SC+DR
680 FORM=0TO120STEP40:FORN=0TO3:W=PEEK(AD+M+N)AND1
    27:IFW<>PEEK(1632+M+N)THEN580
690 NEXT:NEXT:PRINT"{HOME}"TAB(24)"{5 DOWN}{CYN}
    {RVS}YOU WIN!⁅7⁆":POKES1-3,0:POKES1,33:WN=1
700 READN1,N2,D:IFN1=-1THENPOKES1,8:GOTO780
710 POKES1-4,N1:POKES1-3,N2:FORT=1TOD:NEXT:GOTO700
720 :
730 :
740 REM END OF GAME
750 PRINT"{HOME}"TAB(23)"{5 DOWN}{WHT}{RVS}!YOU LO
    SE!⁅7⁆":POKES1-3,10:POKES1,17:WN=1
```

```
760 POKES2-3,60:POKES2,129:FORT=1TO300:NEXT:POKES2
    ,8:POKES1,8
770 :
780 TM$=TI$:PRINT"{HOME}"TAB(21)"{9 DOWN}(1) RESET
790 PRINTTAB(21)"{DOWN}(2) QUIT":IFWN=0THENPRINTTA
    B(21)"{DOWN}(3) AS YOU LEFT IT"
800 GETV$:IFV$<"1"ORV$>"3"THEN800
810 IFV$="1"THENRUN
820 IFV$="2"THENEND
830 IFWNTHEN800
840 PRINT"{HOME}{8 DOWN}":FORI=1TO6:PRINTTAB(21)"
    {18 SPACES}":NEXT
850 TI$=TM$:GOTO580
860 :
870 :
880 REM SETUP AND MUSIC DATA
890 DATA0,49,1,1,178,3,2,51,1,3,180,3
900 DATA40,53,1,41,182,3,42,55,1,43
910 DATA184,3,80,57,1,81,129,3,82,2,1
920 DATA83,131,3,120,4,1,121,133,3,122
930 DATA6,1,123,32,3,-1,-1,-1
940 DATA 96,22,150,0,0,50,96,22,75,0,0,50,96,22,75
    ,49,28,175,96,22,115,49,28
950 DATA175,135,33,250,0,0,0,-1,-1,-1
```

Chapter 4

Dexterity

Blockhead

Matt Giwer 64 Version by Gregg Peele

Here is a challenging game for the whole family. See how many balloons the blockhead can pop in the allotted time. Requires game paddles.

"Blockhead" is a colorful game similar to some of the early arcade games. It is simple to play, and will especially appeal to young children, who will like the clever use of sound and color in the game. The program makes good use of the Commodore 64's graphic capabilities, for it utilizes the eight available sprites and even includes a machine language routine. This interrupt-driven routine provides optimal motion in the game, as well as monitors the position of the sprites.

Once you have the program typed in, SAVEd, and LOADed, you can see that the machine language routine still operates, even if the BASIC part of the program does not. LOAD and RUN Blockhead, then press RUN-STOP. This breaks the BASIC program, but the blockhead can still be moved with the paddle control.

Blockhead uses the collision register to detect when one sprite touches another. Since the collision register is changed only temporarily when sprites collide, the contents representing the collision must be saved until an event occurs which may again make the sprite collide. The register is then cleared, and the sprite is ready for collision. Collision detection between the blockhead and balloons is handled through BASIC.

The game is played with a set of paddles, which must be plugged into Control Port 1. Since Blockhead is a one-player game, only one paddle will work. The paddle moves the blockhead's home base from side to side, with the blockhead standing on it. You use the fire button on the paddle to make the blockhead leap.

The original version of this game is written to be used with Atari-style paddles. If you have Commodore paddles, you must change lines 1070 and 1080 to read as follows:

```
1070 DATA 216,24,173,164,194,105,28,141
1080 DATA 161,194,56,173,164,194,233,217
```

This alteration leaves a slight glitch in the paddle movement around the seam but provides for optimal range for movement around the screen.

Playing the Game

This game works using a timer. The object of the game is to pop the balloons as they float across the sky. The more balloons you pop in the time limit of two minutes, the more points you'll receive. Not only must you pop the balloons, but you must also catch the blockhead before he falls below his home base. If you miss catching him, points are deducted until you bring him to the surface by pressing the fire button. He'll then leap back into the air.

For each balloon that you pop, you receive 10 points. Each time you drop the blockhead, your score is reduced by 15 points.

When you LOAD and RUN the program for Blockhead, a tune plays and the screen sets up. This takes a few moments, so be patient. Finally, the blockhead appears, and the balloons begin to float across the sky. At first, they are close to the ground and easy to pop. Simply press the fire button and the blockhead leaps into the air. If he touches a balloon, it disappears, and you'll hear a soft popping sound. You've just received ten points. The balloons will continue to float at this level until all six of them are popped by the blockhead.

As soon as the first level of balloons has been popped, the tune plays again, then another level, slightly higher, appears from the left side of the screen. There are six levels of balloons altogether. If you pop all the balloons, 36 in all, the game stops, even if there is time remaining. At this point, you're asked if you want to play another game.

Of course, popping the balloons is only half the fun. You also have to catch the blockhead as he drops to the ground. If you miss him with the paddle-controlled base, he will vanish. To make him reappear, you need to press the fire button to make him leap back up.

Going for the High Score

After playing Blockhead a few times, you'll notice some things that can increase your score, or reduce the time it takes you to pop all the balloons.

If you time the blockhead's leap, you can pop two balloons at once. This must be precise. The blockhead has his hands outstretched, and if both come in contact with a balloon at the same time, the balloon on either side will pop. Sometimes this works, and other times it doesn't.

You can also receive points if the blockhead comes very close to a balloon. The balloon won't pop, but you'll hear the popping sound, and another ten points will be added to your score. Just as with trying to pop two balloons at once, this will not work all the time.

If you keep the blockhead's home base stationary, most of the time he will fall back to it. Not always, so you have to keep your eye on him.

Remember that the blockhead is not able to pop a balloon on the way down, only on the way up.

Blockhead

```
100 POKE49152,Ø
110 DIM HA(12),HB(12),HC(12),LA(12),LB(12),LC(12)
120 FORQ=1TO11:READHA(Q),LA(Q),HB(Q),LB(Q),HC(Q),L
    C(Q):NEXT
130 S=54272:FORE=STOS+28:POKEE,Ø:NEXT
140 POKE54296,15 :POKE54277,56 :POKE54278,212
150 POKE54284,56 :POKE54286,212
160 POKE54291,56 :POKE54292,212
170 POKE S+4,17:POKES+16,17:POKES+18,17
180 FORD=1TO11
190 POKES+1,HA(D):POKES,LA(D):POKES+8,HB(D)
200 POKES+9,LB(D):POKES+15,HC(D):POKES+14,LC(D)
210 FORT=1TO1ØØ:NEXT
220 IFHC(D) =7THENFORT=1TO1ØØ:NEXT
230 NEXT
240 FORT=1TO 45Ø :NEXT:FORE=STOS+28:POKEE,Ø:NEXT
250 IFPEEK(49152)=173ANDTH=1THENRETURN
260 DATA33,135,21,31,8,97,31,165,21,31,8,225,29,22
    3,22,96,9,1Ø4
270 DATA 28 ,49,22,96,9,247,26,156,21,31,1Ø,143
280 DATA28,49,21,31,9,247,29,223,22,96,9,1Ø4,31,16
    5,22,96,8,225
290 DATA33,135,21,31,8,97,25,3Ø,22,96,7,233,33,135
    ,21,31,8,97
300 GOTO33Ø
310 S=54272
320 POKES+24,15:POKE54276,65:POKE54275,1Ø:POKE5427
    4,1Ø:POKES+24,Ø:RETURN
330 POKE53281,7:HI=134:GOSUB93Ø
340 DATA1,255,Ø,7,255,192,15,239,224,31,1,24Ø,63,1
    Ø9,248,63,111,248,63,1,248,63
```

```
350 DATA237,248,63,109,248,31,1,240,31,239,240,15,
    239,224,15,255,224,7,255,192,3
360 DATA255,128,1,255,0,0,254,0,0,124,0,0,56,0,0,1
    6,0,0,56,0
370 V=53248
380 FOR J=960TO1022:READ WQ:POKE J,WQ:NEXT
390 POKEV+21,0
400 POKEV+41,6:POKEV+42,0:POKEV+43,1:POKEV+44,2:PO
    KEV+4,70
410 POKE53264,0
420 POKEV+45,4:POKEV+46,8
430 FORT=2042TO2047:POKET,15:NEXT:POKEV+21,255
440 IFPEEK(V+2)<50AND(PEEK(V+16)AND2)=0THENPOKEV+2
    ,254
450 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,255,255,255,255,255
460 DATA255,255,255,255,255,255,255,255,255,255,25
    5,255,255,255,255,255,255,255
470 DATA255,255,255,255,255,255,255,255,255,255
480 V=53248
490 FORI=832TO894:READJ:POKEI,J:NEXT
500 FORK=834+64TO892+66:READL:POKEK,L:NEXT:POKE204
    1,14:POKEV+40,6
510 POKE2040,13:POKEV+39,2:POKEV,150:POKEV+1,200
520 IFPEEK(49152)<>173THENGOSUB1050
530 POKEV+3,191
540 IFHI<70THENHI=59
550 TH=1:GOSUB130
560 POKEV+2,PEEK(V):POKEV+21,255
570 FORG=V+5TO V+15STEP2:POKEG,HI:NEXT
580 SYS49658
590 DATA0
600 DATA0,0,0,0,0,0,3,255,240,3,63,48,3,51,48,3,24
    3,240,3,63,48,3,204,240,3,243
610 DATA240,3,255,240,0,127,128,127,243,255,127,25
    5,255,255,255,255,128,115
620 DATA128,0,127,128,0,127,128,0,251,192,1,241,22
    4,3,224,240,7,192,120
630 IF(PEEK(56321)AND4)<>0THEN790
640 X2=0:POKE49829,0
650 FORT=(PEEK(V+3))TO50STEP-4:POKEV+3,T
660 IFPEEK(V+30)>3THENPOKEV+21,(PEEK(V+21)ANDNOT(P
    EEK(V+30))):SC=SC+10:GOSUB310
670 POKE(V+21),(PEEK(V+21)OR3)
680 NEXT:GOTO700
690 GOTO790
700 POKE49829,0
710 FORJ=(PEEK(V+3))TO255STEP20:POKEV+3,J:IFPEEK(4
    9829)=3THENX2=1:GOTO790
```

```
720 PI=INT(RND(Ø)*2Ø)-1Ø:IF(PEEK(5325Ø)+PI)<6ØAND(
    PEEK(53264)AND2)=ØTHENPI=Ø
73Ø IF(PEEK(V+2)+PI)<5ØAND(PEEK(V+16)AND2)=ØORPEEK
    (V+2)>254THENPI=Ø
74Ø IF(PEEK(53264)AND2)<>ØAND(PEEK(5325Ø)+PI)>2ØTH
    ENPI=Ø
75Ø IF PEEK(5325Ø)+PI<245AND PEEK(5325Ø)+PI>1ØTHEN
    POKE5325Ø,PEEK(5325Ø)+PI
76Ø IFPEEK(V+3)<2Ø1THEN78Ø
77Ø PRINT"{HOME}{3 DOWN}{7 RIGHT}{BLK}OOPS!":SC=SC
    -5:FORT=1TO1ØØ:NEXT:PRINT"{HOME}{7 RIGHT}
    {3 DOWN}{5 SPACES}"
78Ø NEXT
79Ø IF PEEK(V+21)=3THEN:HI=HI-15:POKEV+3,19Ø:GOTO5
    3Ø
8ØØ IFX2=1ANDPEEK(V+3)>18ØTHENPOKEV+3,19Ø
81Ø P=INT(RND(Ø)*2Ø)-1Ø:IFPEEK(5325Ø)+P<15THENP=Ø
82Ø PRINT"{HOME}{15 RIGHT}{BLK}SCORE";"{5 SPACES}"
    ;
83Ø PRINT"{HOME}{15 RIGHT}{BLK}SCORE";SC
84Ø IFVAL(TI$)>59ØØTHENTI$="ØØØØØØ"
85Ø IFTI$>="ØØØ2ØØ"THEN87Ø
86Ø PRINT"{HOME}{DOWN}{3 RIGHT}TIME ";RIGHT$(TI$,4
    );"{HOME}{DOWN}{3 RIGHT}TIME ";:GOTO63Ø
87Ø PRINT"{HOME}{15 RIGHT}{8 DOWN}GAME OVER":POKE1
    98,Ø
88Ø PRINT"{HOME}{DOWN}{3 RIGHT}TIME ";RIGHT$(TI$,4
    );"{HOME}{DOWN}{3 RIGHT}TIME ";
89Ø PRINT"{HOME}{1Ø RIGHT}{1Ø DOWN}PLAY AGAIN? Y O
    R N "
9ØØ IFPEEK(197)=25THENCLR:RESTORE:GOTO11Ø
91Ø IFPEEK(197)=39THENSYS2Ø48
92Ø GOTO89Ø
93Ø PRINT"{CLR}";:FORBO=1Ø24TO1984STEP4Ø:POKEBO,22
    4:POKEBO+39,224
94Ø POKEBO+54272,2:POKEBO+54311,2
95Ø POKEBO+1,224:POKEBO+38,224
96Ø POKEBO+1+54272,4:POKEBO+5431Ø,4
97Ø POKEBO+2,224:POKEBO+37,224
98Ø POKEBO+2+54272,15:POKEBO+543Ø9,15
99Ø NEXT
1ØØØ FORFL=1864TO2Ø23:POKEFL,224:POKEFL+54272,8:NE
     XT
1Ø1Ø TI$="235952"
1Ø2Ø FORTE=1Ø25TO1Ø62:POKETE,224:POKETE+54272,3:NE
     XT
1Ø3Ø POKE5328Ø,1
1Ø4Ø RETURN
```

```
1050 POKEV+21,0:FORV1=49152TO49673:READJ2:POKEV1,J
     2:CK=CK+J2:NEXT
1051 IF CK<>65960 THEN PRINT "DATA ERROR IN LINES
     {SPACE}1060-1710":STOP
1052 RETURN
1060 DATA 173, 25, 212, 73, 255, 141, 164, 194
1070 DATA 216, 24, 173, 164, 194, 105, 40, 141
1080 DATA 161, 194, 56, 173, 164, 194, 233, 215
1090 DATA 141, 162, 194, 173, 164, 194, 201, 216
1100 DATA 176, 17, 173, 161, 194, 141, 163, 194
1110 DATA 173, 16, 208, 41, 254, 141, 16, 208
1120 DATA 76, 65, 192, 173, 16, 208, 9, 1
1130 DATA 141, 16, 208, 173, 162, 194, 141, 163
1140 DATA 194, 173, 163, 194, 141, 0, 208, 173
1150 DATA 30, 208, 141, 160, 194, 240, 3, 141
1160 DATA 165, 194, 173, 160, 194, 41, 1, 240
1170 DATA 23, 169, 190, 173, 163, 194, 141, 2
1180 DATA 208, 173, 16, 208, 41, 1, 141, 6
1190 DATA 202, 10, 13, 6, 202, 141, 16, 208
1200 DATA 173, 16, 202, 56, 233, 210, 141, 17
1210 DATA 202, 173, 16, 202, 24, 105, 45, 141
1220 DATA 18, 202, 173, 16, 202, 201, 210, 176
1230 DATA 17, 173, 16, 208, 41, 251, 141, 16
1240 DATA 208, 173, 18, 202, 141, 4, 208, 76
1250 DATA 168, 192, 173, 16, 208, 9, 4, 141
1260 DATA 16, 208, 173, 17, 202, 141, 4, 208
1270 DATA 173, 19, 202, 56, 233, 210, 141, 20
1280 DATA 202, 173, 19, 202, 24, 105, 45, 141
1290 DATA 21, 202, 173, 19, 202, 201, 210, 176
1300 DATA 17, 173, 16, 208, 41, 247, 141, 16
1310 DATA 208, 173, 21, 202, 141, 6, 208, 76
1320 DATA 224, 192, 173, 16, 208, 9, 8, 141
1330 DATA 16, 208, 173, 20, 202, 141, 6, 208
1340 DATA 173, 22, 202, 56, 233, 210, 141, 23
1350 DATA 202, 173, 22, 202, 24, 105, 45, 141
1360 DATA 24, 202, 173, 22, 202, 201, 210, 176
1370 DATA 17, 173, 16, 208, 41, 239, 141, 16
1380 DATA 208, 173, 24, 202, 141, 8, 208, 76
1390 DATA 24, 193, 173, 16, 208, 9, 16, 141
1400 DATA 16, 208, 173, 23, 202, 141, 8, 208
1410 DATA 173, 25, 202, 56, 233, 210, 141, 26
1420 DATA 202, 173, 25, 202, 24, 105, 45, 141
1430 DATA 27, 202, 173, 25, 202, 201, 210, 176
1440 DATA 17, 173, 16, 208, 41, 223, 141, 16
1450 DATA 208, 173, 27, 202, 141, 10, 208, 76
1460 DATA 80, 193, 173, 16, 208, 9, 32, 141
1470 DATA 16, 208, 173, 26, 202, 141, 10, 208
1480 DATA 173, 28, 202, 56, 233, 210, 141, 29
1490 DATA 202, 173, 28, 202, 24, 105, 45, 141
```

```
1500 DATA 30, 202, 173, 28, 202, 201, 210, 176
1510 DATA 17, 173, 16, 208, 41, 191, 141, 16
1520 DATA 208, 173, 30, 202, 141, 12, 208, 76
1530 DATA 136, 193, 173, 16, 208, 9, 64, 141
1540 DATA 16, 208, 173, 29, 202, 141, 12, 208
1550 DATA 173, 31, 202, 56, 233, 210, 141, 32
1560 DATA 202, 173, 31, 202, 24, 105, 45, 141
1570 DATA 33, 202, 173, 31, 202, 201, 210, 176
1580 DATA 17, 173, 16, 208, 41, 127, 141, 16
1590 DATA 208, 173, 33, 202, 141, 14, 208, 76
1600 DATA 192, 193, 173, 16, 208, 9, 128, 141
1610 DATA 16, 208, 173, 32, 202, 141, 14, 208
1620 DATA 238, 16, 202, 238, 16, 202, 24, 173
1630 DATA 16, 202, 105, 43, 141, 19, 202, 173
1640 DATA 19, 202, 105, 43, 141, 22, 202, 173
1650 DATA 22, 202, 105, 43, 141, 25, 202, 173
1660 DATA 25, 202, 105, 43, 141, 28, 202, 173
1670 DATA 28, 202, 105, 43, 141, 31, 202, 173
1680 DATA 30, 208, 240, 3, 141, 160, 194, 76
1690 DATA 49, 234, 120, 169, 0, 141, 20, 3
1700 DATA 169, 192, 141, 21, 3, 88, 96, 0
1710 DATA 255, 255, 0, 0, 255, 255, 0, 0
```

4

Diamond Drop

Matt Giwer 64 Version by Eric Brandon

Catch the falling diamonds—if you can. This fast-action game is easy to play.

"Diamond Drop" is a game that requires good judgment and quick reflexes. It's fast and easy to play. To insure fast action, it is written predominantly in machine language. BASIC is used only to print instructions, set up the display, select the skill level, and initiate the drop.

The game display starts with six rows of objects at the top of the screen and a stack of six catching trays at the bottom. As the objects begin to drop, you must use the L and ; keys to maneuver the trays and catch the objects. To make play more challenging, one tray disappears whenever the last ball drops from a row. Thus, you have only one tray with which to catch objects from the last row. When all the objects have dropped, you start again with six rows of objects and six trays. Play continues until a total of five objects hit the ground.

Since the DATA statements comprise the machine language program for the game, it is essential that they be typed correctly. Be sure to SAVE a copy of the program before you attempt to RUN it, since an error in typing may cause your computer to lock up, forcing you to turn the power off to recover. If Diamond Drop fails to RUN properly, the problem will most likely be a mistyped number somewhere in the DATA statements, so check carefully.

Diamond Drop

```
5 POKE 53280,12:POKE53281,0
7 IF PEEK(49152)<>120THENGOSUB49000
9 SYS 49745
10 PRINT"{CLR}{WHT}"TAB(13)"DIAMOND DROP"
20 PRINT"{5 DOWN}{YEL}{5 SPACES}CATCH THE DIAMONDS
   BEFORE THEY
30 PRINT"{DOWN}{5 SPACES}TOUCH THE GROUND. YOU HAV
   E FIVE
40 PRINT"{DOWN}{5 SPACES}CHANCES.
```

```
45 PRINT"{2 DOWN}{WHT}{13 SPACES}L - MOVE LEFT
46 PRINT"{13 SPACES}; - MOVE RIGHT{YEL}"
50 PRINT"{5 DOWN}[6]{9 SPACES}{RVS}HIT ANY KEY T
   O BEGIN"
60 GETA$:IFA$=""THEN60
65 GOSUB 1000
70 PRINT"{CLR}{WHT}SCORE 00000{4 SPACES}CHANCES: Q
   QQQ "
71 SPEED = 53241
72 PADDLES=12*4096+4095
73 FLAG=12*4096+4094 : POKE FLAG,0
74 WIDTH = 12*4096+15*256+15*16+11
75 POKE PADDLES,6 : POKE WIDTH,W : POKE SPEED,10-S
78 ROW(6)=81:ROW(5)=81:ROW(4)=207:ROW(3)=207:ROW(2
   )=90:ROW(1)=90
80 PRINT" {YEL}{RVS}";:FORI=1TO38:PRINT"Z";:NEXT:P
   RINT"{OFF} ";
85 PRINT" {YEL}{RVS}";:FORI=1TO38:PRINT"Z";:NEXT:P
   RINT"{OFF} ";
90 PRINT" {CYN}{RVS}";:FORI=1TO38:PRINT"P";:NEXT:P
   RINT"{OFF} ";
95 PRINT" {CYN}{RVS}";:FORI=1TO38:PRINT"P";:NEXT:P
   RINT"{OFF} ";
100 PRINT" {OFF}[7]";:FORI=1TO38:PRINT"W";:NEXT:
    PRINT" ";
102 PRINT" {OFF}[7]";:FORI=1TO38:PRINT"W";:NEXT:
    PRINT" ";
105 PRINT"{WHT}";
109 REM 40 SPACES IN NEXT LINE
110 FORI=1TO17:PRINT"{40 SPACES}";:NEXT
120 PRINT"{HOME}";
130 FOR I=1984 TO 2023 : POKE I,248:POKE I+54272,1
    0:NEXT
140 IF PEEK(789)<>12*16THENSYS 12*4096
150 FOR ROW = 6 TO 1STEP-1:FOR CHAR=1 TO 38
155 FOR K=1 TO 600-CHAR*10+(6-ROW)*20-50*(9-PEEK(S
    PEED)):NEXT
157 IF PEEK(FLAG) THEN 2000
160 P=RND(1)*38+1
170 IF PEEK(1024+ROW*40+P)=32THEN160
180 POKE 1024+ROW*40+P,ROW(ROW)
190 NEXTCHAR
191 SYS 49745
192 FORQ=1TO2:POKE54296,05 :POKE54277,5:POKE54278,
    218
193 POKE 54273,150 :POKE54272,139:POKE54276,17
194 FORT=1TO50:NEXT:POKE54276,16:FORT=1TO10:NEXT
195 NEXTQ
197 IF ROW >1 THENSYS 49691
```

```
200 NEXTROW
201 FOR K=1 TO 300:NEXTK
205 POKE PADDLE,6
206 IF PEEK(SPEED)=2 AND PEEK(WIDTH)>1 THEN POKE W
    IDTH,PEEK(WIDTH)-1
207 IF PEEK(SPEED)>2 THEN POKE SPEED,PEEK(SPEED)-1
210 PRINT"{HOME}{DOWN}";
220 GOTO 80
999 END
1000 PRINT"{CLR}{7 SPACES}DIFFICULTY{4 SPACES}
     {5 DOWN}"
1010 INPUT"{WHT}SPEED (1-9){YEL}{3 RIGHT}5{3 LEFT}
     ";S
1015 IF S>9 OR S<1 THEN 1010
1020 INPUT"{3 DOWN}{WHT}WIDTH OF PADDLES (1-9)
     {YEL}{3 RIGHT}4{3 LEFT}";W
1030 IF W>9 OR W<1 THEN 1020
1040 RETURN
2000 PRINT"{HOME}{10 DOWN}{2 SPACES}{YEL}GAME OVER
      - HIT SPACE TO CONTINUE"
2010 POKE 198,0
2020 GETA$:IFA$<>" "THEN2020
2030 RUN 65
49000 PRINT"{WHT}{CLR}{2 DOWN}LOADING MACHINE LANG
      UAGE...{3 DOWN}":TI$="000000"
49005 I=49152
49007 PRINT"READY IN"STR$(31-VAL(TI$))" SECONDS
      {UP}"
49010 READ A:CK=CK+A:IF A=256 THEN 49030
49020 POKE I,A:I=I+1:GOTO 49007
49030 IFCK<>89323 THEN PRINT "ERROR IN LINES 49152
       TO 49840":STOP
49040 RETURN
49152 DATA 120,169,192,141,21,3,169
49160 DATA 29,141,20,3,88,169,18
49168 DATA 141,253,207,169,0,141,250
49176 DATA 207,141,247,207,141,248,207
49184 DATA 96,173,255,207,141,252,207
49192 DATA 172,253,207,169,32,153,151
49200 DATA 7,200,169,160,174,251,207
49208 DATA 153,151,7,200,202,208,249
49216 DATA 169,32,153,151,7,206,252
49224 DATA 207,208,3,76,3,193,172
49232 DATA 253,207,169,32,153,71,7
49240 DATA 200,169,160,174,251,207,153
49248 DATA 71,7,200,202,208,249,169
49256 DATA 32,153,71,7,200,206,252
49264 DATA 207,208,3,76,3,193,172
49272 DATA 253,207,169,32,153,247,6
```

```
49280 DATA 200,169,160,174,251,207,153
49288 DATA 247,6,200,202,208,249,169
49296 DATA 32,153,247,6,200,206,252
49304 DATA 207,240,123,172,253,207,169
49312 DATA 32,153,167,6,200,169,160
49320 DATA 174,251,207,153,167,6,200
49328 DATA 202,208,249,169,32,153,167
49336 DATA 6,200,206,252,207,240,91
49344 DATA 172,253,207,169,32,153,87
49352 DATA 6,200,169,160,174,251,207
49360 DATA 153,87,6,200,202,208,249
49368 DATA 169,32,153,87,6,200,206
49376 DATA 252,207,240,59,172,253,207
49384 DATA 169,32,153,7,6,200,169
49392 DATA 160,174,251,207,153,7,6
49400 DATA 200,202,208,249,169,32,153
49408 DATA 7,6,200,206,252,207,240
49416 DATA 27,172,253,207,169,32,153
49424 DATA 183,5,200,169,160,174,251
49432 DATA 207,153,183,5,200,202,208
49440 DATA 249,169,32,153,183,5,200
49448 DATA 165,197,201,42,208,13,173
49456 DATA 253,207,201,1,240,24,206
49464 DATA 253,207,76,40,193,201,50
49472 DATA 208,14,173,253,207,24,109
49480 DATA 251,207,201,39,240,3,238
49488 DATA 253,207,238,250,207,173,250
49496 DATA 207,205,249,207,240,3,76
49504 DATA 49,234,169,0,141,250,207
49512 DATA 169,112,133,251,169,7,133
49520 DATA 252,160,0,185,152,7,41
49528 DATA 127,201,32,208,74,200,192
49536 DATA 39,208,242,160,0,177,251
49544 DATA 201,81,240,37,201,207,240
49552 DATA 33,201,90,240,29,200,192
49560 DATA 40,208,237,56,165,251,233
49568 DATA 40,133,251,176,2,198,252
49576 DATA 166,251,208,220,166,252,224
49584 DATA 4,208,214,76,49,234,170
49592 DATA 152,24,105,40,168,138,145
49600 DATA 251,152,56,233,40,168,169
49608 DATA 32,145,251,32,251,193,76
49616 DATA 99,193,169,32,153,152,7
49624 DATA 32,81,194,169,15,141,24
49632 DATA 212,169,17,141,5,212,169
49640 DATA 213,141,6,212,169,2,141
49648 DATA 3,212,169,100,141,2,212
49656 DATA 169,5,141,1,212,169,135
49664 DATA 141,0,212,169,65,141,4
```

```
49672 DATA 212,160,0,162,0,142,32
49680 DATA 208,232,208,250,200,208,247
49688 DATA 169,12,141,32,208,169,64
49696 DATA 141,4,212,160,39,185,0
49704 DATA 4,201,81,240,11,136,208
49712 DATA 246,169,1,141,254,207,76
49720 DATA 49,234,169,32,153,0,4
49728 DATA 76,49,234,152,72,160,10
49736 DATA 185,0,4,201,57,208,9
49744 DATA 169,48,153,0,4,136,76
49752 DATA 255,193,185,0,4,24,105
49760 DATA 1,153,0,4,104,168,96
49768 DATA 174,255,207,202,142,255,207
49776 DATA 232,169,152,133,251,169,7
49784 DATA 133,252,56,165,251,233,80
49792 DATA 133,251,176,2,198,252,202
49800 DATA 208,242,160,0,177,251,201
49808 DATA 160,240,4,200,76,59,194
49816 DATA 174,251,207,169,32,145,251
49824 DATA 200,202,208,250,96,160,0
49832 DATA 152,153,0,212,200,192,9
49840 DATA 208,248,96,256
```

Laser Bounce

Frank L. Broadnax

Don't let the ball get by you. The longer you can chip away at the bricks, the higher your score.

"Laser Bounce" is a game of movement and trajectory similar to some of the earlier videogames. Using only the character set provided with the Commodore 64, it displays a spaceship, the laser balls which rebound from the ship, and the walls of energy you are trying to break through.

Played with a joystick plugged into Control Port 2, the game begins with a simple title screen and a short musical introduction. At that point you're asked if you want to read the instructions before the game. If this is your first game, you would press Y, and the instructions appear. Once you've played the game, however, you can press N and go directly to the screen setup.

The screen sets up quickly, with the present and high scores displayed at the top, your spaceship in the middle, and the six colored energy walls below. The number of spaceships remaining is indicated by the small circles near the top-right-hand corner of the display.

As soon as the screen is completed, the game begins. Your spaceship fires its laser, and the ball appears. The ball will travel in one of four directions to start the game. It will move up and to the right, up and to the left, down and to the right, or down and to the left. Be especially watchful for the ball to move up, toward your spaceship, for you won't have much time to intercept it.

Intercepting the laser ball makes it rebound and move toward the energy walls or the side of the screen. It will bounce off both, but you'll receive points only if it hits the wall and eliminates a brick. Ten points are awarded for each gap created.

Although it doesn't matter which part of the ship the ball touches, it's best to use its center. Sometimes you may think you're in the right position, but the ball misses one wing of the ship and gets by you. Unlike other games of trajectory, the ball will not bounce at a different angle depending on where it strikes the ship. No matter where the ball touches the spaceship, it will simply rebound.

The ship moves rather slowly, so it's a good idea to keep track of the ball, especially when it gets trapped in the wall and is busy eliminating bricks. You should be able to tell when the ball will escape from the wall and head back toward you. Anticipating it is important: if your spaceship is out of position, it will be hard to recover in time to intercept. Because the spaceship moves three columns at a time, its movement is sometimes jerky, and can make it seem like the spaceship is changing position faster than it actually is.

The laser ball is also hard to keep track of at times. Because it is drawn and erased each time it moves, it blinks off and on. However, when it erases bricks from the energy wall, it seems to disappear for a moment. If it is eliminating bricks rapidly, the best way to keep track of it is to watch the pattern of erasing bricks. Plotting where it will return toward your ship, you can move to that position.

If you miss intercepting the ball, and it gets by you, your spaceship will reappear in the middle of the screen, fire its laser, and another round begins. You have a total of five spaceships during a game, the number remaining indicated by the display.

If you erase all five energy walls, the game isn't over. Another five walls are drawn when you reach 4800 points, the total you should have after eliminating all the bricks. Each time all five walls are erased, another five appear to take their place. You receive no additional spaceships, however.

As the game ends, a message appears asking if you want to play another game. Pressing Y sets up another screen after you've indicated whether you need to read the instructions again. The score will return to 0, but the previous high score remains as long as the computer is left on. The high score only prints once a ball is missed. You can quit playing simply by entering N when the prompt appears at the end of a game.

Laser Bounce Variations

It's easy to create several variations of this game simply by altering a few of the program lines.

An interesting variation can be created by changing line 400. Instead of the value DY = −DY, insert DX = −DX. This will make the laser ball wind its way down through the energy walls, reappearing and moving toward the spaceship only after it's erased its way free.

Another change can be made in lines 460 and 470. Insert GOTO 310 instead of GOTO 320. After a ball is missed, the energy screens will be redrawn, in effect making you start over. Your score will not return to 0, however.

Changing the value of DX in line 335 will also create another variation of Laser Bounce. DX = 2 will alter the angle at which the ball rebounds. This can make the ball difficult to intercept, especially as the game begins and the ball moves up and to one side. You'll have to be fast to intercept it before it gets by you.

Programmer's Notes

It may be useful to outline some of the major subroutines of this game program so you can see how it all fits together.

Lines	**Function**
5-170	Set up the title screen and send the program to the subroutine which plays the opening music.
180-220	Begin the setup of the game instructions, and send the program to the subroutine at 35000, which contains the rest of the game description.
230-335	Set variables and the screen.
330	Ball movement loop begins. This is the main loop of the program.
335	Ball starts from the end of the laser.
336-337	Alter the direction of the ball each time it appears.
400	Check to see if the ball touches a brick in the energy wall.
420-450	Check to see if the ball is in contact with the spaceship.
10100	Subroutine to create the firing laser using only standard graphic characters.
12000-12130	POKE in the graphic character and colors to create the energy walls. The reversed space with screen code value of 160 was used to make the walls.
15000-15160	Create the spaceship using the graphic characters with screen code values of 73, 81, and 83, and two characters with the value of 67.
20000-20080	Subroutine to move the spaceship. The ship moves three columns at a time by erasing its previous position and POKEing in the new location. The value is read from the joystick (PEEK 56320). This subroutine also keeps the spaceship on the screen.
25000-25020	Scoring subroutine which starts in line 400, then moves to this section of the program. The score is printed to the screen, changing by 10 each time a brick is erased. Lines 25011 to 25019 redraw the bricks once the screen is cleared, depending on the score displayed.

27000-27040	Sound subroutine for the effect as the ball hits and erases the bricks. Values are POKEd into sound memory locations for attack (A), waveform (W), high sound register (HF), and low sound register (LO). The sound variables are set in line 260, at the opening of the program.
30000-30070	Subroutine to handle a missed ball and the high score. The spare spaceships are controlled here as well. Line 30030 increases PL by 1 each time a ball is missed. If PL exceeds 1098, then the game ends; otherwise, line 30070 POKEs a value of 102 in location PL, erasing one spare spaceship.
35000-35100	Remainder of the screen and game instructions.
40000-40240	Set values and the DATA statements for the music which plays at the opening of the game
45000-45030	Subroutine which asks the player whether another game is wanted.

Laser Bounce

```
5 PRINT"{CLR}"
1Ø PRINT
2Ø PRINT
3Ø PRINT
4Ø PRINT
5Ø PRINTSPC(8)"***********************"
6Ø PRINTSPC(8)"*{21 SPACES}*"
7Ø PRINTSPC(8)"*{4 SPACES}LASER{2 SPACES}BOUNCE
   {4 SPACES}*"
8Ø PRINTSPC(8)"*{21 SPACES}*"
16Ø PRINTSPC(8)"***********************"
17Ø GOSUB4ØØ1Ø
18Ø PRINTTAB(128)"DO YOU WISH INSTRUCTIONS"
19Ø PRINTTAB(96)"Y OR N"
2ØØ GETA$:IFA$=""THEN2ØØ
21Ø IFA$="Y"THENPRINT"{CLR}":GOSUB35Ø1Ø
22Ø IFA$<>"Y"THEN23Ø
23Ø PRINT"{CLR}":POKE5328Ø,11:POKE53281,Ø
24Ø P1=1Ø94:SC=Ø:CO=54272
25Ø FORR=54272TO54296:POKER,Ø:NEXT
26Ø L=54296:W=54276:A=54277:HF=54273:LF=54272
27Ø POKEL,15
3Ø2 FORU1=1Ø24TO1Ø63:POKEU1,16Ø:POKEU1+CO,11:NEXT
3Ø4 FORU2=1Ø64TO11Ø3:POKEU2,1Ø2:POKEU2+CO,11:NEXT
3Ø6 FORU3=1Ø95TO1Ø98:POKEU3,87:POKEU3+CO,1:NEXT
3Ø7 PRINTTAB(6)"{UP}{WHT}SCORE="
3Ø8 PRINTSPC(23)"{3 UP}{WHT}HI="
31Ø GOSUB12Ø1Ø
```

```
320 GOSUB15Ø1Ø
325 C=1161:V=1162:B=1163:N=1164:M=1165
33Ø REM BALL
335 X=19:Y=9:DX=1:DY=1
336 IFRND(1)<.5THENDY=-DY
337 IFRND(1)<.5THENDX=-DX
34Ø POKE1Ø24+X+4Ø*Y,81:POKE55296+X+4Ø*Y,1
37Ø POKE1Ø24+X+4Ø*Y,32
38Ø X=X+DX:IFX=ØORX=39THENDX=-DX
39Ø Y=Y+DY:IFY=24THENDY=-DY
395 BL=1Ø24+X+4Ø*Y:C1=16Ø
4ØØ IFPEEK(BL)=C1THENDY=-DY:SC=SC+1Ø:GOSUB25Ø1Ø:GO
    SUB27Ø1Ø
42Ø IFPEEK(BL)=67THENDY=-DY:GOTO39Ø
43Ø IFPEEK(BL)=81THENDY=-DY:GOTO39Ø
44Ø IFPEEK(BL)=85THENDY=-DY:GOTO39Ø
45Ø IFPEEK(BL)=73THENDY=-DY:GOTO39Ø
46Ø IFPEEK(BL)=1Ø2THENGOSUB3ØØ1Ø:GOTO32Ø
47Ø IFPEEK(BL)=87THENGOSUB3ØØ1Ø:GOTO32Ø
48Ø GOSUB2ØØ2Ø:GOTO34Ø
1ØØØØ REM LASER DELAY
1Ø1ØØ FORT=1TO1ØØ:NEXT:RETURN
12ØØØ REM DRAW BRICKS
12Ø1Ø FORQ1=15Ø4TO1583:POKEQ1,16Ø:POKEQ1+CO,7:NEXT
12Ø3Ø FORQ2=1584TO1663:POKEQ2,16Ø:POKEQ2+CO,6:NEXT
12Ø5Ø FORQ3=1664TO1743:POKEQ3,16Ø:POKEQ3+CO,8:NEXT
12Ø7Ø FORQ4=1744TO1823:POKEQ4,16Ø:POKEQ4+CO,5:NEXT
12Ø9Ø FORQ5=1824TO19Ø3:POKEQ5,16Ø:POKEQ5+CO,2:NEXT
1211Ø FORQ6=19Ø4TO1983:POKEQ6,16Ø:POKEQ6+CO,4:NEXT
1213Ø RETURN
15ØØØ REM LASER SHIP & LASER FIRE
15Ø1Ø FORZ=1144TO1183:POKEZ,32:NEXT
15Ø2Ø POKE1161,85:POKE1162,67:POKE1163,81:POKE1164
      ,67:POKE1165,73
15Ø3Ø FORZ1=55416TO55455:POKEZ1,1:NEXT
15Ø4Ø POKE12Ø3,66:POKE55475,2:GOSUB1Ø1ØØ
15Ø5Ø POKE1243,66:POKE55515,2:GOSUB1Ø1ØØ
15Ø6Ø POKE1283,66:POKE55555,2:GOSUB1Ø1ØØ
15Ø7Ø POKE1323,66:POKE55595,2:GOSUB1Ø1ØØ
15Ø8Ø POKE1363,66:POKE55635,2:GOSUB1Ø1ØØ
15Ø9Ø POKE14Ø3,81:POKE55675,1:GOSUB1Ø1ØØ
151ØØ POKE12Ø3,32:GOSUB1Ø1ØØ
1511Ø POKE1243,32:GOSUB1Ø1ØØ
1512Ø POKE1283,32:GOSUB1Ø1ØØ
1513Ø POKE1323,32:GOSUB1Ø1ØØ
1514Ø POKE1363,32:GOSUB1Ø1ØØ
1515Ø POKE14Ø3,32:GOSUB1Ø1ØØ
1516Ø RETURN
2ØØØØ REM SHIP MOVEMENT
```

```
20020 IFPEEK(56320)=119THENPOKEC,32:POKEV,32:POKEB
      ,32:M=M+3:N=N+3:B=B+3:V=V+3:C=C+3
20030 IFPEEK(1183)=73THENM=1183:N=1182:B=1181:V=11
      80:C=1179
20040 POKEM,73:POKEN,67:POKEB,81:POKEV,67:POKEC,85
20050 IFPEEK(56320)=123THENPOKEM,32:POKEN,32:POKEB
      ,32:C=C-3:V=V-3:B=B-3:N=N-3:M=M-3
20060 IFPEEK(1144)=67THENC=1143:V=1144:B=1145:N=11
      46:M=1147
20070 POKEC,85:POKEV,67:POKEB,81:POKEN,67:POKEM,73
20080 RETURN
25000 REM PRINT SCORE
25010 PRINTTAB(12)"{UP}{WHT}"SC
25011 IFSC=4800THENGOSUB12010
25012 IFSC=9590THENGOSUB12010
25013 IFSC=14380THENGOSUB12010
25014 IFSC=19170THENGOSUB12010
25015 IFSC=23960THENGOSUB12010
25016 IFSC=28750THENGOSUB12010
25017 IFSC=33540THENGOSUB12010
25018 IFSC=38330THENGOSUB12010
25019 IFSC=43120THENGOSUB12010
25020 RETURN
27000 REM SOUND
27010 POKEA,9:POKEW,17:POKEHF,67:POKELF,15
27030 POKEW,0
27040 RETURN
30000 REM MISSED BALL & HI SCORE
30010 IFSC>HITHENHI=SC
30020 PRINTSPC(26)"{3 UP}{WHT}"HI
30030 P1=P1+1:IFP1>1098THENPRINTTAB(254)"{WHT}GAME
      {3 SPACES}OVER":GOTO45000
30070 POKEP1,102:POKEP1+CO,11:RETURN
35000 REM INSTRUCTIONS
35010 PRINTTAB(88)"WELCOME TO LASER BOUNCE"
35020 PRINTTAB(40)"THE OBJECT OF LASER BOUNCE IS T
      O REFLECT"
35030 PRINT"THE BALL BACK TO THE BRICKS WITH YOUR"
35040 PRINTTAB(40)"SPACE SHIP."
35050 PRINTTAB(40)"TO MOVE YOUR SHIP USE A JOY STI
      CK"
35060 PRINTTAB(40)"PLUGGED INTO CONTROL PORT # 2."
35070 PRINTTAB(126)"PRESS SPACE BAR TO PROCEED"
35080 GETP$:IFP$=""THEN35080
35090 IFP$<>CHR$(32)THEN35080
35100 IFP$=CHR$(32)THENRETURN
40000 REM SONG AT BEGINING
40010 SO=54272
40020 FORL=SQTOSO+24:POKEL,0
```

```
40030 POKESO+5,9:POKESO+6,40
40040 POKESO+24,15
40050 READHF,LF,DR
40060 IFHF<0THENRETURN
40070 POKESO+1,HF:POKESO,LF
40080 POKESO+4,33
40090 FORT=1TODR:NEXT
40100 POKESO+4,32:FORT=1TO50:NEXT
40110 GOTO40050
40120 DATA14,24,250,11,48,125,12,143,125,14,24,125
40130 DATA11,48,125,12,143,125,14,24,125,15,210,25
      0
40140 DATA12,143,125,14,24,125,15,210,125,12,143,1
      25
40150 DATA14,24,125,15,210,125,16,195,250,18,209,2
      50
40160 DATA14,24,125,15,210,125,11,48,125,12,143,12
      5
40170 DATA14,24,250,12,143,125,11,48,125,16,195,25
      0
40180 DATA16,195,250,14,24,250,11,48,125,12,143,12
      5
40190 DATA14,24,125,11,48,125,12,143,125,14,24,125
40200 DATA15,210,250,12,143,125,14,24,125,15,210,1
      25
40210 DATA12,143,125,14,24,125,15,210,125,16,195,2
      50
40220 DATA18,209,250,14,24,125,15,210,125,11,48,12
      5
40230 DATA12,143,125,14,24,125,16,195,125,14,24,12
      5
40240 DATA12,143,125,11,48,500,-1,-1,-1
45000 PRINTTAB(44)"{WHT}DO YOU WISH ANOTHER GAME Y
       OR N"
45010 GETA$:IFA$=""THEN45010
45020 IFA$="Y"THENPRINT"{CLR}":GOTO180
45030 IFA$="N"THENPRINT"{CLR}":END
```

Chapter 5

Arcade-Style Games

5

The Hawkmen of Dindrin

Esteban V. Aguilar, Jr. 64 Version by Charles Brannon

Fly down through the dangerous skies of the planet Dindrin to collect stones. Retrieve enough of them and win the game, but beware of the floaters and lizards. Several special techniques are used in this game, including animation, multicolor sprites, and sound effects, each of which is explained in the article.

There's a strange planet named Dindrin where multicolor floaters and a giant sky skimmer drift through the daytime skies. On the surface of the planet, vicious land hunters come up from the ground and set polished golden stones in the sun. It's a form of worship too obscure, too alien to describe.

Suddenly a strange-looking hawk-like creature dives down and snatches a stone. You are the hawkman. Your objective is to pick up the golden stones.

Several special programming tricks went into this game. When you have the game running, watch the screen carefully. A patrol snake sweeps across the bottom of the screen. Airborne floaters pop up all over the screen. The hawkman's wings flap. The luminous stones at the bottom of the screen are protected by menacing lizards whose tongues wiggle venomously at you.

To play the game, use a joystick plugged into the first port. Maneuvering is accomplished by pulling left on the joystick to go backward. Whenever you want to dive or fly upward, you must pull down or up (respectively) on the stick. One thing to keep in mind when ascending or descending is that you will move diagonally rather than straight up or down.

The joystick response will be strange and difficult to master, but predictable. Once in a while, an obstacle such as a floater will get in your way; press the fire button to safely bump into the obstacle (and get points for it).

There are a couple of things to consider before playing the game. As time passes, you will lose energy. If your energy runs out, you will lose a life. Second, when you're flying, don't run into anything or you'll lose one of your lives. When all your lives are lost, the game is over.

How It's Done

Multicolored characters are used for the stones and the lizards. The patrol snake is a multicolored sprite.

The animation (wing flapping, tongue wiggling) is done by switching between two custom character sets. Every object to be animated has two alternate views. The same image is copied into both character sets for shapes that should not move, such as the stones or the score line.

A machine language routine is used for smooth, even horizontal motion for the patrol snake. Instead of being called when needed by BASIC, the machine language routine runs continuously in the background. The machine language routine also flips the character set.

Interrupting the Commodore 64

We used the hardware interrupt request (IRQ). To place a machine language routine so that it automatically executes every 1/60 second, you change the IRQ vector at $0314 (it normally points to the ROM interrupt routines) to point to your machine language routine. After your routine executes, it exits with a JMP to the normal ROM routine.

The setup is a little tricky. While you're storing the new IRQ value, you have to use SEI (SEt Interrupt disable bit) to prevent any interrupts from happening. If you don't, an interrupt *could* occur after you had stored the first byte of the vector value but before you changed the second. The interrupt would then vector through a "half-baked" value, and end up in limbo.

After you've changed the IRQ vector, you clear the interrupt disable bit (CLI) and return with RTS to BASIC. The machine language routine will then be running continuously in the background, flipping the character set and moving the sprite.

Multicolor

Multicolor graphics are important for good arcade effects. A few years ago, graphic objects (such as a tank or plane) were always a single color. But increasing realism has been a feature of arcade graphics, and multicolored objects are an important aspect of this realism.

Normally, when you define a custom character set, you create eight rows of pixels (picture elements, dots). Each row is eight dots (or bits) wide. With multicolor, each row is divided up into four two-bit pairs. Each pair of bits can hold a number from 0-3: 00, 01, 10, 11. You use a different number for each color. This reduces the resolution to four multicolor pixels per row, so the lizards and stones are composed of two characters each. You also have to tell the VIC-II chip that you are using multicolor. Do this with:

POKE 53270, PEEK (53270) OR 16

Disable multicolor with:

POKE 53270, PEEK (53270) AND 239

Here is a sample multicolor shape:

```
rrrr   r=red (arbitrary colors)
rbbb   b=blue
rbgg   g=green
rbgg
```

Let's say the binary codes for red, green, and blue are (respectively) 01, 10, and 11. Substituting gives:

```
01 01 01 01   01010101
01 10 10 10   01101010
01 10 11 11   01101111
01 10 11 11   01101111
```

You can change the colors according to this key:

00 Background #0 color register - 53281
01 Background #1 color register - 53282
10 Background #2 color register - 53283
11 Color in lower 3 bits in color memory.

That last line needs explaining. You know that to get variously colored characters, you POKE a number from 0-15 into the corresponding color memory location. However, colors 8-15 (accessed by the Commodore key) are really multicolors. Multicolor characters always are displayed with a color from 8-15. You won't get the eight alternate colors (such as gray), but the normal color on the key (15=yellow). Just add eight to the normal color number. So, a bit value of 11 will take on the value in color memory. The other colors will come from the color registers (00 is transparent).

Multicolored sprites are similar. Instead of the normal 24-bit resolution, the bits are grouped into 12-bit pairs. The colors come from:

00 – Transparent, screen color
01 – Sprite multicolor register #0 53285
10 – Normal sprite color register
11 – Sprite multicolor register #1 53286

You tell the VIC-II chip that you are using a multicolored sprite by:

POKE 53276, PEEK (53276) OR (2 ↑ X)

X is the sprite number, from 0 to 7. You can mix multicolored and regular sprites on the same screen. But all multicolored sprites will share the same two multicolor registers.

Simple SID Chip Sound

The "thrumming" noise is made by playing a low-pitched tone through the SID using the variable pulse wave and a fairly long (one-second) decay. Another sound effect (I can't really describe it) is made with white noise and a medium decay. The high byte of the pitch is changed as the note is played. There is also another sound effect created by the sawtooth waveform affecting the low byte of the pitch.

Hawkmen of Dindrin

```
100 REM HAWKMEN OF DINDRIN
110 REM COMMODORE 64 VERSION
120 POKE52,48:POKE56,48:CLR:GOSUB500:EN=500:GOTO16
    0
130 PRINT"{HOME}{RVS}{RED}";TAB(9)"{LEFT}";EN;"
    {BLU}";TAB(26-LEN(STR$(SC)));SC;
140 IF EN<=0THEN410
150 RETURN
160 IF(PEEK(56321)AND15)<>15THENJS=PEEK(56321)AND1
    5
170 IFRND(1)>.9THENQ=LL*RND(1)+(15*RND(1)+2)*LL:PO
    KET+Q,FOOL:POKEC+Q,6*RND(1)+2
180 IFRND(1)<.7THEN200
190 Q=920+INT(20*RND(1))*2:Z=33-2*(RND(1)>.7):POKE
    T+Q,Z:POKET+Q+1,Z+1
200 IFPEEK(V+31)THEN410
210 Q=PX+LL*PY:POKET+Q,PC:POKEC+Q,6:EN=EN-1-9*(1-(
    PEEK(56321)AND16)/16)
215 PRINT"{HOME}{RVS}{RED}"TAB(9);"{LEFT}";-EN*(EN
    >0);"{LEFT} ";:IFEN<=0THEN410
```

```
220 NX=PX+1+2*(JS=11):NY=PY+(NX<Ø)-(NX>39):NX=-NX*
    (NX<4Ø)-4Ø*(NX<Ø)
23Ø NY=NY-(JS=13)+(JS=14):IFNY<2ORNY>23THENJS=27-J
    S:NY=PY
24Ø WHATSIT=PEEK(T+NX+LL*NY)
25Ø IF NY>22 THEN 3ØØ
26Ø IFWHATSIT=32THENPOKET+PX+LL*PY,32:PX=NX:PY=NY:
    GOTO16Ø
27Ø IFPEEK(56321)AND16THEN41Ø
28Ø POKET+PX+LL*PY,32:POKES+24,15:POKES+5,9:POKES+
    6,Ø:POKES+1,1Ø
281 FORI=ØTO1Ø:POKES,I*2Ø:POKES+4,32:POKES+4,33:NE
    XT:POKES+24,Ø
29Ø WHATSIT=32:SC=SC+1Ø:EN=EN-5Ø:GOSUB13Ø:GOTO25Ø
3ØØ JS=27-JS:IFWHATSIT<33ORWHATSIT>34THEN33Ø
3Ø5 Q=(NXAND254)+LL*NY:POKET+Q,32:POKET+Q+1,32:EN=
    EN+5Ø
31Ø GOTO32Ø
32Ø POKET+PX+LL*PY,32:PX=NX:SC=SC+5Ø:GOSUB13Ø:GOTO
    16Ø
33Ø IFWH=32THEN16Ø
34Ø REM GRAB'EM AND EAT 'EM UP!
35Ø POKET+PX+LL*PY,32:Q=LL*NY+(NXAND254):POKET+Q,3
    7:POKET+Q+1,38:POKET+Q-LL,42
36Ø POKET+Q-LL+1,36:POKEC+Q-LL,13:POKEC+Q-LL+1,13
37Ø POKES+24,15:POKES+1,Ø:POKES,255:POKES+3,8:POKE
    S+2,Ø:POKES+5,12:POKES+6,Ø
375 POKES+4,64:POKES+4,65:FORW=1TO15ØØ:NEXT:POKES+
    4,64:FORL=STOS+24:POKEL,Ø:NEXT
38Ø POKE T+Q,33:POKET+Q+1,34:POKET+Q-LL,32:POKET+Q
    -LL+1,32
39Ø GOTO 43Ø
4ØØ REM PLAYER MEETS HIS DEMISE
41Ø POKES+24,15:POKES+5,9:POKES+6,Ø:POKES,2ØØ
42Ø FORI=ØTO9ØSTEP6:Q=PX+LL*PY:POKET+Q,44+I/3Ø:POK
    EC+Q,8*RND(1)
425 POKE5328Ø,16*RND(1):POKES+1,I:POKES+4,128:POKE
    S+4,129:NEXT
427 FORL=STOS+24:POKEL,Ø:NEXT
43Ø POKE5328Ø,Ø:IFLI<3THENPOKET+35+LI*2,32
44Ø POKET+PX+LL*PY,32:Z=PEEK(V+31):LI=LI+1:IFLI<4T
    HENEN=5ØØ:GOSUB72Ø:GOTO16Ø
45Ø SYS52992:REM TURN OFF ML
46Ø PRINT"{HOME}{3 DOWN}{RVS}";TAB(15);"{BLK}G
    {RED}A{CYN}M{PUR}E{RIGHT}{GRN}O{BLU}V{YEL}E
    {RED}R{BLU}"
47Ø PRINTTAB(7)"{DOWN}{RVS}PRESS {RED}FIRE{BLU} TO
     PLAY AGAIN"
48Ø IF(PEEK(56321)AND16)THEN48Ø
```

```
490 RUN
500 REM INITIALIZATION
510 POKE53280,0:POKE53281,1
515 T=1024:C=55296:S=54272:LL=40
520 CHSET=12288:IFPEEK(CHSET+264)=2 THEN 570
530 PRINT"{CLR}":C$="{BLK}{RED}{CYN}{PUR}{GRN}
    {YEL}{BLU}":FORI=1TO7:PRINT"{HOME}{DOWN}";MID$
    (C$,I,1);:GOSUB2000:NEXT
550 PRINTTAB(10)"{3 DOWN}{2 RIGHT}{BLK}READY IN
    {RED}22{BLK} SECONDS";
560 GOSUB750:GOSUB 840
570 PRINT"{CLR}";:FOOL=41
575 FORL=STOS+24:POKEL,0:NEXT
580 PC=43:POKE53282,10:POKE53283,2
590 POKE 53272,(PEEK(53272)AND240)OR12:REM ENABLE
    {SPACE}NEW CHARACTER SET
600 POKE 53270,PEEK(53270)OR16 :REM SET MULTICOLOR
     MODE
610 PRINT"{HOME}{RED}{RVS}{2 SPACES}ENERGY 500
    {2 SPACES}{BLU}{2 SPACES}SCORE{4 SPACES}0
    {GRN}{2 SPACES}LIVES {OFF}{PUR}+ + +"
630 FORI=0TO39STEP2:Q=24*LL+I:POKET+Q,39:POKET+Q+1
    ,40:POKEC+Q,7:POKEC+Q+1,7:NEXT
640 FORI=0TO39STEP2:Q=23*LL+I:POKET+Q,33:POKET+Q+1
    ,34:POKEC+Q,13:POKEC+Q+1,13:NEXT
650 Q=10+23*LL:POKET+Q,35:POKET+Q+1,36
660 V=53248:REM START OF VIC-II CHIP REGISTERS
670 POKEV,220:POKEV+1,194:POKEV+21,1:POKEV+39,7:PO
    KE2040,13
680 POKEV+23,1:POKEV+29,1:POKE53285,3:POKE53286,4:
    POKE53276,PEEK(53276)OR1
681 FORI=0TO63:POKE832+I,0:NEXT:RESTORE
685 FORI=0TO18:READA:POKE832+8+I,A:NEXT
690 DATA192,0,3,240,0,15,124,85,95,255,0,12,8,0,3,
    0,0,0,240
700 FORI=1TO5:Q=40*RND(1)+(10*RND(1)+3)*LL:POKET+Q
    ,FOOL:POKEC+Q,6*RND(1)+2:NEXT
710 SYS52992:REM START ML ROUTINE
720 PX=5:PY=5:PC=43:POKET+PX+LL*PY,PC:POKEC+PX+LL*
    PY,6
730 IF(PEEK(56321)AND15)=15THEN730
740 RETURN
750 RESTORE:FORI=0TO18:READA:NEXT:FORI=0TO96:READA
    :POKE52992+I,A:NEXT:RETURN
760 DATA 120,173,21,3,201,234,208,19
770 DATA 169,39,141,20,3,169,207,141
780 DATA 21,3,169,0,133,251,133,252
790 DATA 76,37,207,169,49,141,20,3
802 DATA 169,234,141,21,3,88,96,165
```

```
804 DATA 251,141,Ø,2Ø8,173,16,2Ø8,41
8Ø6 DATA 254,5,252,141,16,2Ø8,24,165
8Ø8 DATA 251,1Ø5,4,133,251,165,252,1Ø5
81Ø DATA Ø,133,252,24Ø,12,165,251,2Ø1
812 DATA 91,144,6,169,Ø,133,251,133
814 DATA 252,165,162,74,144,8,173,24
816 DATA 2Ø8,73,2,141,24,2Ø8,76,49
818 DATA 234
84Ø POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
841 FORI=ØTO511:POKE13312+I,PEEK(54272+I):POKE1536
    Ø+I,PEEK(54272+I):NEXT
842 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
86Ø READA:IFA=-1THENRETURN
87Ø FORJ=ØTO7:READB:POKECHSET+A*8+J,B:NEXTJ:GOTO86Ø
88Ø DATA 32,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
89Ø DATA 33,2,9,9,9,9,9,2,Ø
9ØØ DATA 34,16Ø,88,88,88,88,88,16Ø,Ø
91Ø DATA 35,12,3,16,196,195,63,3,3
92Ø DATA 36,Ø,192,252,236,252,24Ø,192,192
93Ø DATA 37,3,35,131,139,139,171,35,3
94Ø DATA 38,192,192,224,232,2Ø2,194,194,2ØØ
95Ø DATA 39,64,8Ø,84,85,85,85,85,85
96Ø DATA 4Ø,1,5,21,85,85,85,85,85
97Ø DATA 41,Ø,1Ø2,219,36,126,137,66,6Ø
98Ø DATA 42,Ø,15,Ø,51,63,15,15,3
99Ø DATA{2 SPACES}288,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
1ØØØ DATA 289,2,9,9,9,9,9,2,Ø
1Ø1Ø DATA 29Ø,16Ø,88,88,88,88,88,16Ø,Ø
1Ø2Ø DATA 291,12,3,Ø,192,195,63,3,3
1Ø3Ø DATA 292,Ø,192,252,2Ø4,252,24Ø,192,192
1Ø4Ø DATA 293,3,3,35,171,139,139,131,35
1Ø5Ø DATA 294,192,2ØØ,194,194,2Ø2,232,224,192
1Ø6Ø DATA 295,64,8Ø,84,85,85,85,85,85
1Ø7Ø DATA 296,1,5,21,85,85,85,85,85
1Ø8Ø DATA 297,129,1Ø2,9Ø,36,126,82,36,24
1Ø9Ø DATA 298,Ø,15,Ø,48,63,3,15,15
11ØØ DATA 43,153,219,231,255,9Ø,24,36,66
111Ø DATA 299,24,9Ø,231,255,219,153,36,66
112Ø DATA 44,217,219,231,75,2,24,36,66
113Ø DATA 45,216,225,235,69,7,2,4Ø,66
114Ø DATA 46,192,192,145,3,67,1,72,13Ø
115Ø DATA 47,192,128,8,1,1,Ø,16,128
116Ø DATA 3ØØ,217,219,247,99,22,24,36,68
117Ø DATA 3Ø1,216,225,227,71,23,13Ø,32,66
118Ø DATA 3Ø2,192,2ØØ,129,3,131,1,64,13Ø
119Ø DATA 3Ø3,192,144,Ø,1,1,Ø,8,128
12ØØ DATA -1
```

```
2ØØØ PRINT" {RVS} {2 RIGHT} {2 RIGHT}{2 SPACES}
    {2 RIGHT} {3 RIGHT} {RIGHT} {2 RIGHT} {RIGHT}
     {3 RIGHT} {RIGHT}{3 SPACES}{RIGHT} {3 RIGHT}
     "
2Ø1Ø PRINT" {RVS} {2 RIGHT} {RIGHT} {2 RIGHT}
    {RIGHT} {3 RIGHT} {RIGHT} {RIGHT} {2 RIGHT}
    {2 SPACES}{RIGHT}{2 SPACES}{RIGHT} {3 RIGHT}
    {2 SPACES}{2 RIGHT} "
2Ø2Ø PRINT" {RVS}{4 SPACES}{RIGHT}{4 SPACES}
    {RIGHT} {RIGHT} {RIGHT} {RIGHT}{2 SPACES}
    {3 RIGHT} {RIGHT} {RIGHT} {RIGHT}{2 SPACES}
    {2 RIGHT} {RIGHT} {RIGHT} "
2Ø3Ø PRINT" {RVS} {2 RIGHT} {RIGHT} {2 RIGHT}
    {RIGHT} {RIGHT} {RIGHT} {RIGHT} {RIGHT}
    {2 RIGHT} {3 RIGHT} {RIGHT} {3 RIGHT}
    {2 RIGHT}{2 SPACES}"
2Ø4Ø PRINT" {RVS} {2 RIGHT} {RIGHT} {2 RIGHT}
    {2 RIGHT} {RIGHT} {2 RIGHT} {2 RIGHT} {RIGHT}
     {3 RIGHT} {RIGHT}{3 SPACES}{RIGHT} {3 RIGHT}
     {3 DOWN}"
2Ø6Ø PRINTSPC(15);"{RVS}£{2 SPACES}Ɛ*Ʒ{2 RIGHT}
    {3 SPACES}"
2Ø7Ø PRINTSPC(15);"{RVS} {2 RIGHT} {2 RIGHT} "
2Ø8Ø PRINTSPC(15);"{RVS} {2 RIGHT} {2 RIGHT}
    {3 SPACES}"
2Ø9Ø PRINTSPC(15);"{RVS} {2 RIGHT} {2 RIGHT} "
21ØØ PRINTSPC(15);"Ɛ*Ʒ{RVS}{2 SPACES}{OFF}£
    {2 SPACES}{RVS} {2 DOWN}"
211Ø PRINT"{3 SPACES}{RVS}{3 SPACES}{2 RIGHT}
    {3 SPACES}{RIGHT} {3 RIGHT} {RIGHT}{3 SPACES}
    {2 RIGHT}{3 SPACES}{2 RIGHT}{3 SPACES}{RIGHT}
     {3 RIGHT} "
212Ø PRINT"{3 SPACES}{RVS} {2 RIGHT} {2 RIGHT}
    {2 RIGHT}{2 SPACES}{2 RIGHT} {RIGHT}
    {2 RIGHT} {RIGHT} {2 RIGHT} {2 RIGHT}
    {2 RIGHT}{2 SPACES}{2 RIGHT} "
213Ø PRINT"{3 SPACES}{RVS} {2 RIGHT} {2 RIGHT}
    {2 RIGHT} {RIGHT} {RIGHT} {RIGHT} {2 RIGHT}
    {RIGHT}{3 SPACES}{3 RIGHT} {2 RIGHT} {RIGHT}
    {SPACE}{RIGHT} "
214Ø PRINT"{3 SPACES}{RVS} {2 RIGHT} {2 RIGHT}
    {2 RIGHT} {2 RIGHT}{2 SPACES}{RIGHT}
    {2 RIGHT} {RIGHT} {2 RIGHT} {2 RIGHT}
    {2 RIGHT} {2 RIGHT}{2 SPACES}"
215Ø PRINT"{3 SPACES}{RVS}{3 SPACES}{2 RIGHT}
    {3 SPACES}{RIGHT} {3 RIGHT} {RIGHT}{3 SPACES}
    {2 RIGHT} {2 RIGHT} {RIGHT}{3 SPACES}{RIGHT}
    {SPACE}{3 RIGHT} "
216Ø RETURN
```

Minefield

Sean Igo 64 Translation by Gregg Peele

Your job is to get your trucks in quickly, defuse the bombs (especially the flashing ones which are about to go off), and get out as fast as you can. This game has four skill levels.

In this game, you drive a truck around to gather and defuse time bombs before they explode—all the while avoiding mines and bomb craters.

Playing the Game

You find yourself in the center of a small minefield with several bombs, represented by circles, and a generous number of mines, shown as X's. Your truck is a diamond. To defuse the bombs, just run over them with the truck.

When the bombs first appear, they are innocent-looking little circles. After a short time—the rate varies from bomb to bomb—they turn reverse-field. This means *watch* it. Soon they begin to blink, and you have only a few blinks to defuse them before they explode. Any mines (or heroic defusing teams) caught in the explosion will be instantly lost. Bombs caught in the explosion will explode, whether they were ready to or not.

Your truck can move in only four directions. It can wrap around all four edges of the screen. Don't run it into the mines or the craters (*) left by the bombs or your truck will be destroyed. Once you begin moving, your truck cannot stop until it is blown up or until the current minefield is cleared of bombs.

Skill Levels and Scoring

"Minefield" has four skill levels. Skill levels differ only in the number of trucks you get. Level 0, the easiest, has four trucks. Level 1 has three. Level 2 has two, and level 3 has one.

Scoring: 10 points for a normal bomb
20 points for a reverse-field bomb
30 points for a blinking bomb
−10 points at the end of an explosion for every bomb that went off. This is incentive to defuse more than one or two bombs in the later explosions.

Minefield

```
30 REM MINEFIELD FOR C-64
45 POKE53280,0:POKE53281,0
50 GOSUB 1130
60 REM ---INITIALIZE VARIABLES---
70 DIM BT(37),B3(37),B4(37),BP(37),BS(37),XM(4),YM
   (4),BC(25)
80 DEF FNY(X)=INT((X-1024)/40)
90 DEF FNX(X)=(X-40*FNY(X))-1024
100 DEF FNS(X)=1024+PX+40*PY
110 DEF FNP(X)=1307+INT(34*RND(1))+40*INT(15*RND(1
   ))
120 DEF FNN(X)=PEEK(FNS(X))
130 FORJ=1 TO 4:READ XM(J),YM(J):NEXT
140 DATA 0,-1,0,1,-1,0,1,0
150 SC=0:BT=1680:NB=4:NW=0:D=54272
160 PRINT"{CLR}";:POKE 53272,21
170 PRINT"{RVS}{WHT}MINE****- SCORE: 0"
180 PRINT"{RVS}{WHT}********-{RIGHT}HI SCORE:";HS
190 PRINT"{RVS}{WHT}***FIELD-{RIGHT}WAVE: 1"
200 PRINT"{RVS}{WHT}{8 SPACES}-{RIGHT}";:IF NL<>1
   {SPACE}THEN FORJ=1 TO NL-1:PRINT"Z";:NEXT
210 FORJ=1024 TO 1183:IFPEEK(J)=32 THEN POKE J,160
   :POKEJ+D,1
220 NEXT
230 XP$="{RED}U-I{DOWN}{4 LEFT}UU-II{DOWN}{6 LEFT}
   UUU-III{DOWN}{7 LEFT}*******{DOWN}
   {7 LEFT}JJJ-KKK"
235 XP$=XP$+"{RED}{DOWN}{6 LEFT}JJ-KK{DOWN}
   {4 LEFT}J-K"
240 S$="{HOME}{24 DOWN}"
250 Q$="{WHT}{40 RIGHT}"
260 XR$="{WHT}{3 SPACES}{DOWN}{4 LEFT}{5 SPACES}
   {DOWN}{6 LEFT}{7 SPACES}{DOWN}{7 LEFT}
   {3 SPACES}*{3 SPACES}{DOWN}{7 LEFT}{7 SPACES}"
265 XR$=XR$+"{DOWN}{6 LEFT}{5 SPACES}{DOWN}
   {4 LEFT}{3 SPACES}"
270 REM ---SET UP NEXT WAVE---
280 BG=0:NW=NW+1:IF NW>11 THEN 310
290 NB=NB+1.5:IF NW=1 THEN 330
300 IF NW<6 THEN BT=BT-180
310 PRINT"{HOME}{2 DOWN}{RVS}";TAB(15);NW
320 POKE FNS(1),32:FORJ=1 TO NB:POKEBP(J),32:NEXT
325 FORJ=1 TO 25:POKE BC(J),32:NEXT
330 BN=INT(NB):FORJ=1 TO NB:BS(J)=1:NEXT
340 FORJ=1 TO NB
350 BT(J)=(.4+INT(61*RND(1))/100)*BT
360 B3(J)=BT(J)+.5*BT(J):B4(J)=B3(J)+.25*BT(J)
370 NEXT
```

```
380 PX=19:PY=15:POKE FNS(1),90:POKEFNS(1)+D,1
390 FORJ=1 TO NB
400 BP(J)=FNP(1):IF PEEK(BP(J))<>32 THEN 400
410 POKE BP(J),87:POKEBP(J)+D,8:NEXT:NN=0
415 FORJ=1 TO 25
416 BC(J)=FNP(1):IF PEEK(BC(J))<>32 THEN 416
417 IF PEEK(BC(J)+1)=87 THEN 416
419 POKEBC(J),86:POKEBC(J)+D,5:NEXT
420 GET R$:IF R$<>"" THEN 420
430 DR=0:TX=TI
440 REM ---GET COMMANDS---
450 R=(15-(PEEK(56321)AND15))*2
460 IFR<>0THENDR=LOG(R)/LOG(2){41 SPACES}
470 IFR=0THEN490
480 REM ---MOVE TRUCK---
490 IF DR=0 THEN 600
500 POKE FNS(1),32:PX=PX+XM(DR):PY=PY+YM(DR)
510 IF PX<0 THEN PX=39
520 IF PX>39 THEN PX=0
530 IF PY<4 THEN PY=24
540 IF PY>24 THEN PY=4
550 X=FNN(1)
560 IF X=32 THEN POKE FNS(1),90:POKEFNS(1)+D,1:GOT
    O 600
570 IF X=42 OR X=86 THEN 960
580 GOTO 890
590 REM ---UPDATE BOMBS---
600 NN=NN+1:IF NN>INT(NB)THEN NN=1
610 IF BS(NN)=0 THEN 600
620 TG=TI-TX
630 IF TG>B4(NN) THEN N1=NN:GOTO 720
640 IF BS(NN)>2 THEN 690
650 IF TG>BT(NN) THEN BS(NN)=2
660 IF TG>B3(NN) THEN BS(NN)=3
670 IF BS(NN)=1 THEN 450
680 IF BS(NN)=2 THEN POKE BP(NN),215:POKEBP(NN)+D,
    1:GOTO 450
690 IF BS(NN)=3 THEN POKE BP(NN),87:POKEBP(NN)+D,1
    :BS(NN)=4:GOTO 450
700 IF BS(NN)=4 THEN POKE BP(NN),215:POKEBP(NN)+D,
    1:BS(NN)=3:GOTO 450
710 REM ---BOMB EXPLODES---
720 TQ=TI:PD=0
725 X$="{OFF}"+LEFT$(S$,FNY(BP(N1))-2)+LEFT$(Q$,FN
    X(BP(N1))-1)
730 BS(N1)=0:N2=0:PRINTX$;XP$;
740 FORJ=1 TO NB:X=PEEK(BP(J)):IF BS(J)=0 THEN 760
750 IF X<>87 AND X<>215 AND X<>218 THEN N2=J
760 NEXT:IF FNN(1)<>90 AND FNN(1)<>218 THEN PD=1
```

```
770 PRINTX$;XR$;:GR=129{4 SPACES}:GOSUB2000
780 FORJ=1TONB:IF PEEK(BP(J))=32 AND BS(J)<>0 THEN
     POKE BP(J),87-128*(BS(J)>1)
790 NEXT:BN=BN-1
800 IF PD=1 THEN 960
810 IF BN=0 THEN 840
820 IF N2=0 THEN TX=TX+(TI-TQ):GOTO 450
830 N1=N2:GOTO 725
840 PRINT"{HOME}{2 DOWN}{RVS}";TAB(20);
850 FORJ=1 TO 20:PRINT"{RVS}COMPLETED{9 LEFT}";:FO
    RK=1 TO 100:NEXT
860 PRINT"{RVS}{9 SPACES}{9 LEFT}";:FORK=1 TO 100:
    NEXT:NEXT
870 SC=SC-10*(INT(NB)-BG):IF SC<0 THEN SC=0
880 PRINT"{4 LEFT}{3 UP}{10 SPACES}{10 LEFT}";SC:G
    OTO 280
885 REM ---BOMB GATHERED---
890 BG=BG+1:TQ=TI:POKE FNS(1),218
895 FORJ=1 TO NB:IF PEEK(BP(J))=218 THEN AJ=BS(J):
    BS(J)=0
900 NEXT
910 IF AJ=4 THEN AJ=3
920 SC=SC+10*AJ:PRINT"{HOME}{RVS}";TAB(16);SC
930 GR=33:GOSUB2000:BN=BN-1:IF BN=0 THEN 840
940 TX=TX+(TI-TQ):GOTO 450
950 REM ---PLAYER DESTROYED---
960 GR=129:GOSUB2000
961 TQ=TI:FORJ=1 TO 20:POKE FNS(1),42:FORK=1 TO 25
    :NEXT:POKE FNS(1),170
970 FORK=1 TO 25:NEXT:NEXT:POKE FNS(1),32:NL=NL-1
    {19 SPACES}
980 POKE 1153+NL,160:DR=0:PX=19:PY=15
990 IF NL=0 THEN 1045
1000 IF BN=0 THEN 840
1010 GET R$:IF R$<>"" THEN 1010
1020 FORJ=1TONB:IF PEEK(BP(J))=32 AND BS(J)<>0 THE
     N POKE BP(J),87-128*(BS(J)>1)
1030 NEXT
1040 POKE FNS(1),90:TX=TX+(TI-TQ):GOTO 450
1045 IF SC>HS THEN HS=SC:PRINT"{HOME}{DOWN}{RVS}";
     TAB(19);HS
1050 FORJ=1 TO 1500:NEXT:PRINT"{HOME}{WHT}{2 DOWN}
     {RVS}";TAB(20);"GAME OVER{DOWN}{WHT}{9 LEFT}P
     LAY AGAIN?";
1060 PRINT"(Y/N){4 LEFT}";
1080 PRINT"{RVS}Y/{OFF}N{3 LEFT}";
1081 FORJ=1 TO 99:NEXT
1082 PRINT"{OFF}Y{RVS}/N{3 LEFT}";
1083 FORJ=1 TO 99:NEXT
```

```
1084 GET R$:IF R$="Y" THEN 111Ø
1Ø9Ø IF R$<>"N" THEN 1Ø8Ø
11ØØ PRINT"{CLR}{WHT}LATER ON!":END
111Ø GOSUB 113Ø:GOTO 15Ø
112Ø REM ---INSTRUCTIONS---
113Ø PRINT"{CLR}{RVS}{WHT}M{SHIFT-SPACE}I
     {SHIFT-SPACE}N{SHIFT-SPACE}E{SHIFT-SPACE}F
     {SHIFT-SPACE}I{SHIFT-SPACE}E{SHIFT-SPACE}L
     {SHIFT-SPACE}D":POKE 53272,23
114Ø PRINT"{WHT}DO YOU NEED INSTRUCTIONS (Y/N)"
115Ø GET R$:IF R$="N" THEN 141Ø
116Ø IF R$<>"Y" THEN 115Ø
118Ø PRINT"{CLR}{WHT}{DOWN}THE OBJECT OF THIS GAME
      IS TO PICK UP"
119Ø PRINT"{WHT}AS MANY BOMBS AS YOU CAN BEFORE TH
     EY"
12ØØ PRINT"{WHT}EXPLODE. TO PICK UP A BOMB, JUST R
     UN"
121Ø PRINT"{WHT}OVER IT WITH YOUR TRUCK."
122Ø PRINT"{WHT}BOMBS WILL EXPLODE AFTER A SHORT T
     IME."
123Ø PRINT"{WHT}IF A BOMB TURNS REVERSE-FIELD, BE
     {SPACE}CARE-"
124Ø PRINT"{WHT}FUL WITH IT. IF IT STARTS TO BLINK
     , IT"
125Ø PRINT"{WHT}WILL VERY SHORTLY EXPLODE-WATCH OU
     T!!"
126Ø PRINT"{WHT}BOMBS WILL CHAIN-REACT; ONE BOMB C
     AUGHT"
127Ø PRINT"{WHT}IN ANOTHER'S EXPLOSION WILL ALSO B
     LOW"
128Ø PRINT"{WHT}UP. IF YOU ARE CAUGHT IN A BOMB'S
     {SPACE}"
129Ø PRINT"{WHT}EXPLOSION, YOU WILL BE BLOWN UP."
13ØØ PRINT"{WHT}ALSO, DO NOT RUN INTO BOMB CRATERS
      (*)"
131Ø PRINT"{WHT}OR MINES (X) OR YOU'LL BE TOTALLED
     ."
132Ø PRINT"{WHT}THE CONTOLS ARE: 1 TO GO UP"
133Ø PRINT"{WHT}{17 SPACES}CTRL TO GO LEFT"
134Ø PRINT"{WHT}{17 SPACES}2 TO GO RIGHT"
135Ø PRINT"{WHT}{17 SPACES}← TO GO DOWN"
1355 PRINT"{WHT}OR YOU CAN USE A JOYSTICK IN PORT
     {SPACE}1."
136Ø PRINT"{WHT}YOUR TRUCK CANNOT STOP ONCE YOU BE
     GIN"
137Ø PRINT"{WHT}MOVING. IT CAN WRAP-AROUND BOTH TH
     E"
138Ø PRINT"{WHT}THE TOP AND SIDES OF THE SCREEN."
```

```
1390 PRINT"{DOWN}{WHT}P{WHT}RESS RETURN TO CONTINU
     E";
1400 GET R$:IF R$<>CHR$(13) THEN 1400
1410 PRINT"{CLR}S{WHT}ELECT SKILL SETTING (0-3)"
1420 GET R$:IF R$<"0" OR R$>"3" THEN 1420
1430 NL=4-VAL(R$):RETURN
1900 END
2000 REM SOUND OF EXPLOSION
2010 QW=54272
2020 FORS=QWTOQW+24:POKES,0:NEXT
2025 POKEQW+24,47
2030 POKEQW+5,64+7 :POKEQW+6,240
2050 POKEQW+4,GR :POKEQW+1,36:POKEQW,85
2060 FORT=1TO250:NEXT
2070 FORT=15TO0STEP-1 :POKEQW+24,INT(T):NEXT
2080 RETURN
```

Cylon Zap

Mark Dudley 64 Translation by Gregg Peele

Quick reflexes are what you'll need for this fast-action game.

"Cylon Zap" is an arcade-style game. A space station in the center of the screen, which you must defend at all costs, is attacked continually by Cylon ships. You must shoot them before they dive (kamikaze style) into the space station.

To defend against the Cylons, you have two weapons. First, the joystick is moved up, down, right, or left to fire lasers in any of these four directions. Second, the fire button detonates a smart bomb, which immediately clears the screen of all visible attackers. Smart bombs should be used sparingly, for only three are available at the beginning of play.

The score and the number of remaining bombs are continually updated at the upper-left corner of the screen. When the score reaches 30, the flank attackers begin to increase speed. When your score reaches 50, the attackers from the top and bottom increase their speed. If your score exceeds 60, you win bonus smart bombs.

If your point total is a high score since the program was first loaded, you enter your initials with the joystick. Moving the stick right or left lets you step through the alphabet forward or backward. When you find the correct letter, select it with the fire button. Be sure not to hold the fire button down too long when selecting your initials, or you may inadvertently choose the wrong letters.

Cylon Zap

```
100 POKE52,48:POKE56,48:CLR
125 DATA28,149,100,25,30,100,33,135,100,37,162,50,
    50,60,50
130 DATA42,62,100,37,162,50,50,60,50,42,62,100,33,
    135,100
140 DATA28,49,100,25,30,100
145 FORX=1TO36:READRT:NEXT
150 PRINT"{CLR}":POKE53281,0:POKE53280,0:PRINTCHR$
    (14)
```

```
160 GOSUB590
170 PRINT"{3 DOWN}{11 SPACES}{RVS}LOADING
    {SHIFT-SPACE}CHARACTERS"
180 POKE56334,(PEEK(56334)AND254):POKE1,PEEK(1)AND
    251
190 FORA=0TO2047:POKE(A+12288),PEEK(A+53248):NEXT
200 FORA=12552TO12672
210 READD
220 IFD<>-1THENPOKEA,D:NEXT
230 FORA=12288TO14335:READD:IFD<>-1THENPOKEA,PEEK(
    A):NEXT
240 FORA=12504TO12527:READD:POKEA,D:NEXT
250 POKE1,55
260 POKE56334,PEEK(56334)OR1
270 GOSUB750:PRINT"{UP}{10 SPACES}INSTRUCTIONS
    {OFF} {RVS}Y{OFF} OR {RVS}N{OFF}"
280 GETA$:IFA$=""THENPOKE56079,INT(RND(1)*7+1):POK
    E56084,INT(RND(1)*7+1):GOTO280
290 IFA$="Y"THENPOKE53272,(PEEK(53272)AND240)+12:G
    OSUB380
300 GOTO1000
310 DATA24,24,60,126,24,24,126,255,1,19,51,255,255
    ,51,19,1,128
315 DATA200,204,255,255,204,200
320 DATA128,255,126,24,24,126,60,24,24,24,24,60,24
    ,60,126,219,195
325 DATA3,7,44,254,254,44,7,3
330 DATA192,224,52,127,127,52,224,192,195,219,126,
    60,24,60
335 DATA24,24,16,8,16,8,16,8,16,8
340 DATA145,74,44,113,142,52,82,137,0,0,0,170,85,0
    ,0,0,-1
350 DATA0,0,0,119,68,116,20,119,0,0,0,119,85,87,86
    ,117,0,0,0,112,64,96,64,112
360 DATA0,0,0,206,170,206,170,202,0,0,0,238,136,23
    6,40,238,0,0,0
365 DATA224,128,224,32,224,-1
370 DATA0,0,0,206,170,202,170,206,0,0,0,139,218,17
    1,138,139,0,0,0
375 DATA56,160,56,136,56
380 PRINT"{CLR}{RED}WELCOME TO CYLON ZAP"
390 PRINT"YOU HAVE A BASE NAMED ALPHA"{10 SPACES}:
    PRINT
400 PRINT"{CYN}YOUR MISSION IS TO{2 SPACES}PROTECT
     THE":PRINT"NUCLEAR REACTOR"
410 PRINT"{PUR}FROM THE KAMIKAZE STAR ":PRINT" FIG
    HTERS"
420 PRINT"{DOWN}{GRN}YOU HAVE 4 LASERS{2 SPACES}CO
    NTROLLED BY THE{4 SPACES}JOYSTICK"
```

```
430 PRINT"{BLU}YOU ALSO HAVE SMART BOMBS LAUNCHED
    {SPACE}BY THE FIRE BUTTON"
440 PRINT"{DOWN}{YEL}ALL YOU DO IS POINT THE GUN A
    ND THE{6 SPACES}LASER FIRES AUTOMATICALLY"
450 GOSUB500
460 PRINT"{CLR}{PUR}{DOWN}THE FIGHTERS WILL FLY FA
    STER THE MORE{3 SPACES}OF THEM YOU DESTROY "
470 PRINT"{DOWN}{YEL}BONUS BASE AND BOMB AT 60 POI
    NTS"
480 PRINT"{BLU}{DOWN}{9 SPACES}{RVS}GOOD LUCK":GOS
    UB500:RETURN
490 GOTO65535
500 A$="{RVS}"
510 FORL=1TO1000
520 PRINT"{HOME}"
530 PRINTTAB(2)A$;"{CYN}{20 DOWN}HIT RETURN TO CON
    T"
540 GETR$:IFR$=CHR$(13)THENRETURN
550 FORI=1TO333:NEXT
560 IFA$="{RVS}"THENA$="{OFF}":GOTO580
570 IFA$="{OFF}"THENA$="{RVS}":GOTO580
580 NEXTL
590 A$="{RED}*** *{3 SPACES}* *{4 SPACES}***
    {2 SPACES}*{2 SPACES}*":X=LEN(A$):Z$="{DOWN}":
    GOSUB710
600 A$="*{4 SPACES}* *{2 SPACES}*{4 SPACES}* *
    {2 SPACES}** *":X=LEN(A$):Z$="{2 DOWN}":GOSUB7
    10
610 A$="*{5 SPACES}*{3 SPACES}*{4 SPACES}* *
    {2 SPACES}* **":X=LEN(A$):Z$="{3 DOWN}":GOSUB7
    10
611 A$="*{5 SPACES}*{3 SPACES}*{4 SPACES}* *
    {2 SPACES}* **":X=LEN(A$):Z$="{4 DOWN}":GOSUB7
    10
620 A$="***{3 SPACES}*{3 SPACES}***{2 SPACES}***
    {2 SPACES}*{2 SPACES}* ":X=LEN(A$):Z$="
    {5 DOWN}":GOSUB710
630 A$="{YEL}{2 SPACES}***{2 SPACES}***{2 SPACES}*
    **{2 SPACES}* *{2 SPACES}":X=LEN(A$):Z$="
    {8 DOWN}":GOSUB710
640 A$="{4 SPACES}*{2 SPACES}* *{2 SPACES}* *
    {2 SPACES}* *{3 SPACES}":X=LEN(A$):Z$="
    {9 DOWN}":GOSUB710
650 A$="{3 SPACES}*{3 SPACES}***{2 SPACES}***
    {2 SPACES}* *{3 SPACES}":X=LEN(A$):Z$="
    {10 DOWN}":GOSUB710
660 A$="{2 SPACES}*{4 SPACES}* *{2 SPACES}*
    {10 SPACES}":X=LEN(A$):Z$="{11 DOWN}":GOSUB710
```

```
670 A$="{2 SPACES}***{2 SPACES}* *{2 SPACES}*
    {4 SPACES}* *{2 SPACES}":X=LEN(A$):Z$="
    {12 DOWN}":GOSUB710
680 PRINT:PRINT
700 GOTO170
710 S=54272
711 POKE54296,15 :POKE54277,18:POKE54278,240
712 POKE 54276,33
720 FORI=1TOLEN(A$):POKE54273,I+40
721 PRINT"{HOME}{DOWN}{8 RIGHT}"Z$;SPC(X)LEFT$(A$,
    I):POKE54272,(I*2)+180
730 X=X-1:NEXT:FORG=15TO0STEP-1:POKE54296,G:NEXT:P
    OKES+4,16
735 FORE=STOS+28:POKEE,0:NEXT:RETURN
750 FORA=49152TO49453
760 READD
770 POKEA,D
780 NEXT
790 RETURN
800 DATA169,12,141,33,208,169,147,32,210,255,162,8
    ,160,16,32,240,255,169,18,32
810 DATA210,255,169
820 DATA169,32,210,255,169,127,32,210,255,169,146,
    32,210,255,169,32,32,210
825 DATA 255,169,18,32
830 DATA210,255,169,169,32,210,255,169,127,32,210,
    255,24,162,9,160,15,32
835 DATA 240,255,169,169
840 DATA32,210,255,169,160,162,5,32,210,255,202,22
    4,0,208,248,169,127
845 DATA 32,210,255,24
850 DATA162,10,160,15,32,240,255,169,146,32,210,25
    5,169,127,32,210,255
855 DATA 169,18,32,210,255
860 DATA169,160,162,5,32,210,255,202,224,0,208,248
    ,169,146,32,210,255
865 DATA 169,169,32,210,255
870 DATA24,162,11,160,15,32,240,255,169,32,32,210,
    255,169,18,32,210
875 DATA255,169,160,162,5,32
880 DATA210,255,202,224,0,208,248,169,146,32,210,2
    55,169,32,32,210,255,24
885 DATA 24,162,11,160,7
890 DATA32,240,255,169,18,32,210,255,24,162,12,160
    ,15,32,240,255,169,169
895 DATA 32,210,255,169
900 DATA160,162,5,32,210,255,202,224,0,208,248,169
    ,127,32,210,255,24
905 DATA 162,13,160,15,32,240
```

```
910 DATA255,169,146,32,210,255,169,127,32,210,255,
    169,18,32,210,255
915 DATA 169,160,162,5,32,210
920 DATA255,202,224,0,208,248,169,146,32,210,255,1
    69,169,32,210,255,24
925 DATA 169,146,32,210
930 DATA255,24,162,14,160,16,32,240,255,169,127,32
    ,210,255,169,169,32
935 DATA 210,255,169,32,32
940 DATA210,255,169,127,32,210,255,169,169,32,210,
    255,24,96
1000 RESTORE:CLR
1060 DEFFNA(A)=INT(RND(1)*X+A):TT=1482
1070 POKE53272,(PEEK(53272)AND240)+12
1080 N1=1042:N2=1922:N3=1464:N4=1502:V1=36876
1090 CS=53281:C=54272:W1=30:W2=20:W3=10:W4=5:W5=1
1100 A1$="D..":A2$="U..":A3$="D..":A4$="C..":A5$="
    O.."
1110 POKECS,1:PRINT"{CLR}":GOTO2190
1120 BASE=3:S1=1:S2=1:S3=1:S4=1:BOM=3:SC=0
1130 POKECS,12:X=15:Y=1:I=40
1140 PRINT"{CLR}{WHT}":POKECS,8
1150 GOSUB1450
1160 PRINT"{HOME}{WHT}SCORE"SC:PRINT"{HOME}{DOWN}B
    ASES"BA:PRINT"{WHT}BOMBS"BO
1170 J0=15-(PEEK(56321)AND15)
1180 G=42:FB=(PEEK(56321)AND16)
1190 POKETT,102
1200 POKETT+C,INT(RND(1)*7+1)
1210 IFJ0=1 THEN1510
1220 IFJ0=2 THEN1570
1230 IFJ0=4 THEN1630
1240 IFJ0=8 THEN1690
1250 IFFB=0ANDBOM>0THEN2590
1260 A1=FNA(1)
1270 A2=FNA(2)
1280 A3=FNA(3)
1290 A4=FNA(4)
1300 IFA1=1ANDS1<>0THENS1=0: GOSUB2680
1310 IFA2=2ANDS2<>0THENS2=0: GOSUB2680
1320 IFA3=3ANDS3<>0THENS3=0: GOSUB2680
1330 IFA4=4ANDS4<>0THENS4=0: GOSUB2680
1340 IFS1=0ANDPEEK(N1+40)<>102THENN1=N1+I:POKEN1+C
    ,4:POKEN1,40:POKEN1-I,32
1350 IFPEEK(N1+40)=102THENGOSUB2050
1360 IFS2=0ANDPEEK(N2-40)<>102THENN2=N2-I:POKEN2+C
    ,3:POKEN2,37:POKEN2+I,32
1370 IFPEEK(N2-40)=102THENGOSUB2050
1380 IFS3=0ANDPEEK(N3+1)<>102THENN3=N3+Y:POKEN3+C,
    5:POKEN3,39:POKEN3-Y,32
```

```
1390 IFPEEK(N3+1)=102THENGOSUB2050
1400 IFS4=0ANDPEEK(N4-1)<>102THENN4=N4-Y:POKEN4+C,
     6:POKEN4,38:POKEN4+Y,32
1410 IFPEEK(N4-1)=102THENGOSUB2050
1420 IFBASE=0THENGOTO2130
1430 IFSC>50THENX=4
1440 GOTO1160
1450 PRINT"{RED}":SYS49152:POKECS,11
1460 POKE1362+C,1:POKE1362,33:POKE1602+C,1:POKE160
     2,36:POKE1479+C,1:POKE1479,34
1470 POKE1485+C,1:POKE1485,35
1480 POKETT-1,102:POKETT+1,102:POKETT-40,102:POKET
     T+40,102
1490 POKETT-1+C,1:POKETT+1+C,1:POKETT-40+C,1:POKET
     T+40+C,1
1500 RETURN
1510 POKE54296,15:POKE54273,33:POKE54272,133:POKE5
     4277,50:POKE54278,120
1520 POKE54276,129
1530 FORF=1362TO1042STEP-40
1540 IFPEEK(F-40)<>40THENPOKEF+C,1:POKEF,41:FORT=1
     TO5:NEXT:POKEF,32:NEXT
1550 IFPEEK(F-40)=40THENPOKEN1+C,2:POKEN1,42:GOSUB
     1830:POKEN1,32:N1=1042:S1=1
1560 POKE54296,0:POKE1362,33:GOTO1260
1570 POKE54296,15:POKE54273,33:POKE54272,133:POKE5
     4277,50:POKE54278,120
1580 POKE54276,129
1590 FORF=1602TO1944STEP40
1600 IFPEEK(F+40)<>37THENPOKEF+C,1:POKEF,41:FORT=1
     TO5:NEXT:POKEF,32:NEXT
1610 IFPEEK(F+40)=37THENPOKE2+C,2:POKEN2,42:GOSUB1
     830:POKEN2,32:N2=1922:S2=1-40
1620 POKE54296,0:POKE1602,36:GOTO1260
1630 POKE54296,15:POKE54273,33:POKE54272,133:POKE5
     4277,50:POKE54278,120
1640 POKE54276,129
1650 FORF=1479TO1464STEP-1
1660 IFPEEK(F-1)<>39THENPOKEF+C,1:POKEF,43:FORT=1T
     O5:NEXT:POKEF,32:NEXT
1670 IFPEEK(F-1)=39THENPOKEN3+C,2:POKEN3,42:GOSUB1
     830:POKEN3,32:N3=1464:S3=1
1680 POKE54296,0:POKE1479,34:GOTO1260
1690 POKE54296,15:POKE54273,33:POKE54272,133:POKE5
     4277,50:POKE54278,120
1700 POKE54276,129
1710 FORF=1485TO1502
1720 IFPEEK(F+1)<>38THENPOKEF+C,1:POKEF,43:FORT=1T
     O5:NEXT:POKEF,32:NEXT
```

```
1730 IFPEEK(F+1)=38THENPOKEN4+C,2:POKEN4,42:GOSUB1
     830:POKEN4,32:N4=1502:S4=1
1740 POKE54296,0:POKE1485,35:GOTO1260
1745 FORS0=54272TO54272+28:POKES0,0:NEXT
1750 POKE54296,15:POKE54277,53:POKE54278,69:POKE54
     276,33
1770 RESTORE:FORGB=1TO12:READHA,LA,DU:POKE54273,HA
     :POKE54272,LA
1780 FORT=1TODU:NEXTT
1790 NEXTGB:FORS0=54272TO54272+28:POKES0,0:NEXT
1800 RETURN
1810 DATA217,200,213,200,223,200,227,100,234,100,2
     30,200
1820 DATA227,100,234,100,230,200,223,200,227,200,2
     17,200,213,300,-1
1830 POKE54296,15:POKE54277,53:POKE54278,67:POKE54
     276,129
1840 POKE54272,200:POKE54273,33
1850 FORL=15TO0STEP-1
1860 POKE54296,L
1870 NEXT:POKE54276,0
1880 SC=SC+1
1890 IFSC=30THENX=INT(X/2):Y=2
1900 IFSC=50THENX=4:I=80:BOM=BOM+1
1910 IFSC=60ORSC=110ORSC=150THENGOTO1930
1920 RETURN
1930 PRINT"{CLR}{10 DOWN}{10 SPACES}BONUS";
1940 PRINT" BASE - BOMB":L=0
1950 POKE54296,15:POKE54277,50:POKE54278,167:POKE5
     4276,17
1960 FORT=1TO10
1970 POKE54272,230:POKE54273,33
1980 NEXT
1990 FORT=1TO10
2000 POKE54272,180:POKE54273,28
2010 NEXT
2020 IFL<6THENL=L+1:GOTO1950
2030 FORD=54272TO54272+28:POKED,0:NEXT
2040 BOM=BOM+1:BA=BA+1:SC=SC+5:PRINT"{CLR}":GOSUB1
     450:GOTO1890
2050 POKE54296,14:Q1=1482:Q2=1484:Q3=1522:Q4=1524:
     K=0:Q5=Q1-41:Q6=Q3+41:Q7=Q1+39
2060 Q8=1526:POKE54277,44:POKE54278,56:POKE54276,1
     29
2070 POKE54272,200:POKE54273,34:KK=8
2080 FORZ=15TO0STEP-2
2090 POKE54296,Z:GOSUB2260:NEXT:POKECS,8:POKE54276
     ,0
```

```
2100 N1=1042:S1=1:N2=1922:S2=1:N3=1464:S3=1:N4=150
     2:S4=1:PRINT"{CLR}"
2110 BASE=BASE-1:IFBASE<>0THENGOSUB1450
2120 RETURN
2130 PRINT"{CLR}"
2140 IFSC=>W1THENA5$=A4$:A4$=A3$:A3$=A2$
2150 IFSC=>W1THENA2$=A1$:W5=W4:W4=W3:W3=W2:W2=W1:W
     1=SC:GOTO2730
2154 REM LINE 2155 MUST BE ENTERED USING KEYWORD A
     BBREVIATIONS
2155 IFSC>=W2ANDSC<W1THENA5$=A4$:A4$=A3$:A3$=A2$:W
     5=W4:W4=W3:W3=W2:W2=SC:GOTO2740
2160 IFSC=>W3ANDSC<W2THENA5$=A4$:A4$=A3$:W5=W4:W4=
     W3:W3=SC:GOTO2750
2170 IFSC=>W4ANDSC<W3THENA5$=A4$:W5=W4:W4=SC:GOTO2
     760
2180 IFSC=>W5ANDSC<W4THENW5=SC:GOTO2770
2190 GOSUB2510:PRINT"{HOME}{BLK}{21 DOWN}
     {12 SPACES}TO PLAY HIT {RVS}{BLK}Y"
2200 GETZ$:IFZ$=""THENFORCC=55312TO55315:POKECC,IN
     T(RND(1)*7+1):NEXT
2210 POKE56165,INT(RND(1)*7+1)
2220 IFZ$=""THEN2200
2230 IFZ$="Y"THEN1120
2240 IFZ$="N"THENPRINT"{CLR}{BLU}":POKECS,27:END
2250 GOTO2190
2260 K=K+1:M=41:N=40:O=39:R=INT(RND(1)*7+1)
2270 IFK>3ANDK<110THENPOKECS,KK:KK=KK+31
2280 POKEQ1,G:POKEQ2,G:POKEQ3,G:POKEQ4,G:POKEQ5,G:
     POKEQ6,G:POKEQ7,G:POKEQ8,G
2290 POKEQ1+C,R:POKEQ2+C,INT(RND(1)*7+1):POKEQ3+C,
     R:POKEQ4+C,INT(RND(1)*7+1)
2300 POKEQ5+C,R:POKEQ6+C,INT(RND(1)*7+1):POKEQ7+C,
     R:POKEQ8+C,INT(RND(1)*7+1)
2310 FORT=1TO10:NEXT
2320 IFK>3THENG=46:PRINT"{CLR}"
2330 IFK<8THENQ1=Q1-O:Q2=Q2-M:Q3=Q3+O:Q4=Q4+M:Q5=Q
     5-N:Q6=Q6+N:Q7=Q7-1:Q8=Q8+1
2340 RETURN
2350 PRINT"{CLR}":RETURN
2360 PRINT"{3 DOWN}":CH=1160:E=1
2370 J0=15-(PEEK(56321)AND15)
2380 FB=PEEK(56321)AND16
2390 IFJ0=8THENE=E+1
2400 IFJ0=4THENE=E-1
2410 IFE=0THENE=26
2420 IFE=27THENE=1
2430 POKECH,E:POKECH+C,7
2440 FORT=1TO100:NEXT
```

```
2450 POKECH+C,1
2460 IFFB=Ø ANDCH=116ØTHENN1$=CHR$(E+64):CH=CH+1:E
     =1:GOTO237Ø
2470 IFFB=ØANDCH=1161THENN2$=CHR$(E+64):CH=CH+1:E=
     1:GOTO237Ø
2480 IFFB=Ø ANDCH=1162THENN3$=CHR$(E+64):CH=CH+1:E
     =32:GOTO237Ø
2490 IFCH=1163THENN5$=N1$+N2$+N3$:RETURN
2500 GOTO237Ø
2510 POKE53281,1
2515 REM THE NEXT LINE MUST BE ENTERED USING KEYWO
     RD ABBREVIATIONS
2520 PRINT"{CLR}{2 SPACES}{BLK}{9 SPACES}CYLON ZAP
      HEROS":PRINT:PRINT"{RED}{12 SPACES}BEST 5 SC
     ORES{OFF}"
2530 PRINT"{HOME}{DOWN}{BLK}{4 DOWN}{14 SPACES}"A1
     $"..."W1
2540 PRINT"{BLU}{2 DOWN}{14 SPACES}"A2$"..."W2
2550 PRINT"{GRN}{2 DOWN}{14 SPACES}"A3$"..."W3
2560 PRINT"{PUR}{2 DOWN}{14 SPACES}"A4$"..."W4
2570 PRINT"{RED}{2 DOWN}{14 SPACES}"A5$"..."W5
2580 RETURN
2590 POKE54296,15:POKE54277,43:POKE54278,73:POKE54
     276,129
2600 FORCO=127TO8STEP-17
2610 POKECS,CO
2620 FORT=1TO10Ø:NEXT:NEXTCO:POKECS,11
2630 IFS1=ØTHENSC=SC+1:GOSUB189Ø:POKEN1,32:N1=1Ø42
     :S1=1
2640 IFS2=ØTHENSC=SC+1:GOSUB189Ø:POKEN2,32:N2=1922
     :S2=1
2650 IFS3=ØTHENSC=SC+1:GOSUB189Ø:POKEN3,32:N3=1464
     :S3=1
2660 IFS4=ØTHENSC=SC+1:GOSUB189Ø:POKEN4,32:N4=15Ø2
     :S4=1
2670 FORSØ=54272TO54272+28:POKESØ,Ø:NEXT:BOM=BOM-1
     :GOTO126Ø
2680 S=54272:FORE=STOS+28:POKEE,Ø:NEXT
2690 POKE54296, 15 :POKE54277, 51 :POKE54278, 84
2700 POKE 54276, 17 :FORJ=1TO4ØSTEP4:POKE 54273,J:
     POKE54272,255-J-25:NEXT
2710 FORT=1TO 10Ø :NEXT:POKE54276, 32:FORT=1TO 5Ø:
     NEXT
2720 FORE=STOS+28:POKEE,Ø:NEXT:RETURN
2730 PRINT"{HOME}NUMBER 1 ENTER YOUR INITIALS":GOS
     UB1745:GOSUB236Ø:A1$=N5$:GOTO219Ø
2740 PRINT"{HOME}NUMBER 2 ENTER YOUR INITIALS":GOS
     UB1745:GOSUB236Ø:A2$=N5$:GOTO219Ø
```

```
2750 PRINT"{HOME}NUMBER 3 ENTER YOUR INITIALS":GOS
     UB1745:GOSUB2360:A3$=N5$:GOTO2190
2760 PRINT"{HOME}NUMBER 4 ENTER YOUR INITIALS":GOS
     UB1745:GOSUB2360:A4$=N5$:GOTO2190
2770 PRINT"{HOME}NUMBER 5 ENTER YOUR INITIALS":GOS
     UB1745:GOSUB2360:A5$=N5$:GOTO2190
```

Laser Gunner

Gary R. Lecompte   64 Translation by Philip I. Nelson

This arcade-style game achieves an impressive graphics animation without the use of any machine language.

"Laser Gunner" is an arcade-type action game. The player controls a laser gun which moves up and down on the left of the screen behind a force field and fires at invading enemy spaceships. The invaders also fire lasers and attempt to open holes in the force field. Every hit weakens the force field until an entire hole is made. A hit through a hole ends the game.

Laser Gunner is an example of animation accomplished without the use of machine language routines. The drawback of this type of programming is obvious. Only one string may be animated at a time with any speed. However, by working your game format around this limitation, you can still make action games fast and challenging.

The animation of the laser gun and the position of laser fire, as well as the location of the invaders, are controlled by the location routines. The row and column values are POKEd into memory locations 214 and 211. A PRINT statement following these routines will print that string beginning at the location determined by the row and column values. Changing the row and column values and printing the same string again accomplishes animation.

The force field changes are made by PEEKing the location of the hit, determining the character at that location, and POKEing the value of the next character to that location.

Invader explosions are done by coding cursor movements and printing characters from the invader string.

Sound routines are intermixed with laser and explosion routines. This assures that animation and sound will blend.

Invader ship location and laser fire are determined by randomizing routines. Skill level is provided by giving the player a minimum preset delay. Actual time before invader laser blasts is always unpredictable.

Stars are created with simple POKE statements to predetermined locations.

All routines are placed in order of importance, with those used most at the beginning. This allows for the fastest program execution possible to increase animation speed. REM statements should be deleted for best effect. The key to speed is simplicity. The shorter the program statements, the greater the speed.

Changing the Shapes

It is possible to change the shape of the ships. Lines 85 and 86 contain the statements which produce the shape. To make your own ships, you can use any graphic symbols from the front of the 64 keyboard. Pick the characters you want, and substitute them for the shifted characters within the quotes for IN$, G1$ and G2$ in lines 85 and 86. Remember, you get the left-side graphic character by holding down the Commodore key rather than SHIFT.

Laser Gunner

```
5 POKE53280,0:POKE53281,0:GOSUB190:PRINT"{CLR}":GO
  TO85
10 POKEROW,X:POKECOL,Y:PRINT"{UP}";:RETURN
11 POKEROW,A:POKECOL,B:PRINT"{UP}";:RETURN
12 POKEROW,Z:POKECOL,B:PRINT"{UP}";:RETURN
13 GOSUB10:PRINTG1$;
14 GOTO38
16 TT=TT+1:R=1+INT(RND(1)*10):IFTT>TDTHENIFR=10GOT
   O43
18 IFPEEK(197)=6THEN29
19 IFPEEK(197)=5THEN23
20 IFPEEK(197)=3THEN26
21 GOTO16
23 X=X-1:IFX<1THENX=1
24 GOSUB10:PRINTG1$;:GOTO16
26 X=X+1:IFX>21THENX=21
27 GOSUB10:PRINTG2$;:GOTO16
29 GOSUB180
30 X=X+1:Y=3:GOSUB10:FORI=1TO185STEP5:PRINT"{PUR}>
   ";:NEXT::GOSUB10
31 FORI=1TO37:PRINT" ";:NEXT:X=X-1:Y=0
33 IFX+1=ATHEN60
34 IFX+1=A+1THEN60
35 IFX+1=A+2THEN60
36 GOTO16
38 A=1+INT(RND(1)*21):IFA<3THENA=3
39 IFA>19THENA=19
41 GOSUB11:PRINTIN$:GOTO16
```

```
43 GOSUB17Ø:Z=A+1:B=B-1:GOSUB12:FORI=1TO72STEP2:PR
   INT"{RED}←{2 LEFT}";:NEXT
45 PRINT"{RIGHT}{UP}N{2 DOWN}{LEFT}M":GOSUB12:FORI
   =1TO36:PRINT" {2 LEFT}";:NEXT:PRINT"{RIGHT}{UP}
    {2 DOWN}{LEFT} ":B=B+1
47 HT=SR+((Z-1)*4Ø):RD=PEEK(HT)
48 IFRD=16ØTHENRN=1:GOTO57
49 IFRD=231THENRN=2:GOTO57
5Ø IFRD=234THENRN=3:GOTO57
51 IFRD=246THENRN=4:GOTO57
52 IFRD=97THENRN=5:GOTO57
53 IFRD=117THENRN=6:GOTO57
54 IFRD=116THENRN=7:GOTO57
55 IFRD=1Ø1THENRN=8:GOTO57
56 IFRD=32THENRN=8:GOTO68
57 FORI=1TORN:READFE:NEXT:POKEHT,FE:RESTORE:GOTO16
58 DATA 231,234,246,97,117,116,1Ø1,32
6Ø GOSUB11:PRINT"{RED}{2 LEFT}<{UP}{YEL}+{2 DOWN}
   {3 LEFT}{DOWN}{2 LEFT}[*]{DOWN}{RVS}£"
61 FORI=1TO2Ø:NEXT:GOSUB11:PRINT"{2 LEFT} {UP}
   {2 DOWN}{3 LEFT}{DOWN}{2 LEFT} {DOWN} "
62 GOSUB11:PRINT"{2 UP}{LEFT}£{2 DOWN}{3 LEFT}
   {RED}M{YEL}↑{2 DOWN}{3 LEFT}{YEL}←{DOWN}{LEFT}
   {DOWN}{LEFT}"
63 FORI=1TO2Ø:NEXT:GOSUB11:PRINT"{2 UP}{LEFT}
   {2 DOWN}{3 LEFT}{2 SPACES}{2 DOWN}{3 LEFT}
   {DOWN}{LEFT}{DOWN}{LEFT}":GOSUB16Ø
64 FORI=1TO2Ø:NEXT
65 GOSUB11:PRINT" {LEFT}{DOWN} {LEFT}{DOWN} {LEFT}
   {DOWN}":GOSUB77
67 SC=SC+1:TT=Ø:GOTO38
68 FORI=1TO5ØØ:NEXT
7Ø PRINT"{CLR}{WHT}{3 DOWN}{1Ø SPACES}YOU HIT";SC;
   "INVADERS":GOSUB17Ø:GOSUB17Ø:GOSUB16Ø
71 GOSUB16Ø:GOSUB16Ø:GOSUB16Ø:PRINT"{3 DOWN}
   {14 SPACES}TRY AGAIN?{3 SPACES}"
72 GOSUB17Ø:GOSUB16Ø:GETC$:IFC$=""THEN72
73 IFC$<>"Y"ANDC$<>"N"THEN72
74 IFC$="N"THENPRINT"{CLR}":END
75 SC=Ø:GOTO123
76 REM------GENERATE STARS------------
77 SR=SR-2:P=46
78 POKESR+15,P:POKESR+28,P:POKESR+127,P:POKESR+158
   ,P:POKESR+175,P:POKESR+226,P
79 POKESR+33Ø,P:POKESR+46Ø,P:POKESR+474,P:POKESR+3
   9Ø,P:POKESR+575,P
8Ø POKESR+6Ø5,P:POKESR+628,P:POKESR+7Ø3,P:POKESR+7
   15,P:POKESR+73Ø,P
```

```
81 POKESR+806,P:POKESR+819,P:POKESR+837,P:POKESR+8
   68,P:POKESR+883,P
82 POKESR+904,P:POKESR+928,P:POKESR+947,P:POKESR+9
   64,P:POKESR+992,P
83 SR=SR+2:RETURN
84 REM-------SET VARIABLES------------
85 ROW=214:COL=211:X=5:Y=.:IN$="{YEL}{RVS}£{DOWN}
   {LEFT}{RVS}<{OFF}{DOWN}{LEFT}{*}{WHT}":B=39:
   G1$="{CYN}{RVS}V{OFF}{DOWN}{LEFT}+{DOWN}{LEFT}
   {RVS}V{OFF}{DOWN}{LEFT} "
86 G2$="{CYN}{UP} {DOWN}{LEFT}{RVS}V{OFF}{DOWN}
   {LEFT}+{DOWN}{LEFT}{RVS}V{OFF}":SR=1026:M=21:I=
   RND(-TI)
87 GOSUB77:GOTO91
88 REM--------LOCATION ROUTINE--------
89 POKEROW,M:POKECOL,Y:PRINT"{UP}";:RETURN
90 REM-------TITLES TO 118-----------
91 M=M-1:GOSUB89:PRINTG1$;
92 IFM>7THEN91
93 FORI=1TO200:NEXT:GOSUB89
94 PRINT"{DOWN}{2 RIGHT}";:FORI=1TO110STEP10:PRINT
   "{GRN}>";:NEXT:PRINT"{RVS} LASER GUNNER {OFF}";
95 GOSUB180:FORI=1TO120STEP10:PRINT">";:NEXT
96 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO11:PRIN
   T" ";:NEXT:PRINT"{14 RIGHT}";
97 FORI=1TO12:PRINT" ";:NEXT:GOSUB160
98 GOSUB89:M=M+1:PRINT"{DOWN}"G2$;
99 IFM<12THEN98
100 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO200:NE
    XT
101 FORI=1TO170STEP10:PRINT"{RED}>";:NEXT:GOSUB170
    :PRINT"{PUR}AN";
102 FORI=1TO170STEP10:PRINT"{RED}>";:NEXT:GOSUB170
103 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO17:PRI
    NT" ";:NEXT:PRINT"{2 RIGHT}";:FORI=1TO17
104 PRINT" ";:NEXT
105 GOSUB89:M=M+1:PRINT"{DOWN}"G2$;
106 IFM<16THEN105
107 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO200:NE
    XT
108 FORI=1TO112STEP10:PRINT"{PUR}>";:NEXT:GOSUB170
    :PRINT"{YEL}ACTION GAME";
109 FORI=1TO110STEP10:PRINT"{PUR}>";:NEXT:GOSUB170
110 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO12:PRI
    NT" ";:NEXT:PRINT"{11 RIGHT}";
111 FORI=1TO13:PRINT" ";:NEXT
112 GOSUB89:M=M+1:PRINT"{DOWN}"G2$;
113 IFM<22THEN112
114 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO200:NE
    XT
```

```
115 FORI=1TO9ØSTEP1Ø:PRINT"{RED}>";:NEXT:GOSUB18Ø:
    PRINT"{CYN}{RVS} WANT INSTRUCTIONS?{OFF}";
116 FORI=1TO1ØØSTEP1Ø:PRINT"{RED}>";:NEXT:GOSUB16Ø
117 GOSUB89:PRINT"{DOWN}{2 RIGHT}";:FORI=1TO9:PRIN
    T" ";:NEXT:PRINT"{19 RIGHT}";
118 FORI=1TO1Ø:PRINT" ";:NEXT
119 REM-----WANT INSTRUCTIONS-------
12Ø GETC$:IFC$=""THENGOSUB17Ø:GOSUB17Ø:GOSUB17Ø:GO
    SUB16Ø:GOTO114
121 IFC$="Y"THEN134
122 REM------PICK SKILL LEVEL---------
123 PRINT"{CLR}{YEL}{6 DOWN}{5 SPACES}{RVS}
    {2 SPACES}PICK SKILL LEVEL{5 SPACES}(1-3)
    {OFF}"
124 GOSUB17Ø:GETC:IFC=ØTHEN124
125 IFC>3THEN124
126 PRINT"{CLR}{YEL}{6 DOWN}{8 SPACES}{RVS} PRESS
    {SPACE}[SPACE] TO BEGIN {OFF}"
127 GOSUB16Ø:GETC$:IFC$=""THEN127
128 IFC=1THENTD=15
129 IFC=2THENTD=8
13Ø IFC=3THENTD=Ø
131 REM-----BUILD FORCE FIELD---------
132 PRINT"{CLR}":GOSUB77:PRINT"{HOME}";:FORI=ØTO23
    :PRINTTAB(2)"{RVS}{GRN} {OFF}":NEXT:GOTO13
133 REM--------INSTRUCTIONS----------
134 PRINT"{CLR}{DOWN}{2 SPACES}{RVS}{CYN} YOU ARE
    {SPACE}LASER GUNNER ON A STARSHIP "
135 PRINT"{DOWN} YOU ARE UNDER ATTACK BY ALIEN INV
    ADERS"
136 PRINT"{DOWN}{5 SPACES}YOU MUST MOVE YOUR LASER
     INTO"
137 PRINT"{4 SPACES}POSITION, AND FIRE IT TO DESTR
    OY"
138 PRINT"{11 SPACES}THE ALIEN SHIP"
141 PRINT"{2 DOWN}{3 SPACES}YOU ARE PROTECTED BY A
     FORCE FIELD"
142 PRINT"{4 SPACES}BUT THE FORCE FIELD IS WEAKENE
    D"
143 PRINT"{5 SPACES}WITH EVERY HIT BY AN INVADER"
144 PRINT"{2 DOWN}{5 SPACES}A HIT IN A HOLE ENDS T
    HE GAME"
145 PRINT"{2 DOWN}{7 SPACES}TO MOVE UP----HIT
    {RVS} F3 {OFF} KEY"
146 PRINT"{7 SPACES}TO FIRE-------HIT {RVS} F5
    {OFF} KEY"
147 PRINT"{7 SPACES}TO MOVE DOWN--HIT {RVS} F7
    {OFF} KEY"
148 PRINT"{2 DOWN}{8 SPACES}{RVS}PRESS SPACE TO CO
    NTINUE{OFF}"
```

```
149 GETC$:IFC$=""THEN149
15Ø GOTO123
155 REM--------SOUND SUBROUTINES-------
16Ø POKEW1,21:POKEW2,129:FORZ=2ØTO1STEP-2:POKEH1,Z
    :POKEL1,Z
161 POKEH2,INT(RND(1)*7Ø)+3:POKEL2,Z:NEXT:POKEW1,Ø
    :POKEW2,Ø:RETURN
17Ø POKEW1,17:POKEW2,129:FORZ=35TOØSTEP-7:POKEH1,Z
    :POKEL1,Z:POKEL2,Z
171 POKEH2,INT(RND(1)*7Ø):NEXT:POKEW1,Ø:POKEW2,Ø:R
    ETURN
18Ø POKEW1,21:FORZ=1TO3:FORZX=ØTO1ØØSTEP15:POKEH1,
    ZX:POKEL1,ZX:NEXT:NEXT
181 POKEW1,Ø:RETURN
19Ø POKE54296,15:POKE54277,15:POKE54291,65 :W1=542
    76:W2=5429Ø:H1=54273:L1=54272
191 H2=54287:L2=54286:RETURN
```

Chapter 6

Machine Language Games

Using the Machine Language Editor: MLX

Charles Brannon

Three of the games in this chapter are written completely in machine language. The "Machine Language Editor" will make typing a perfect copy of those games a snap.

Remember the last time you typed in a long machine language program? You typed in hundreds of DATA statements, numbers, and commas. Even then, you couldn't be sure if you'd typed it in right. So you went back, proofread, tried to run the program, crashed, went back and proofread again, corrected a few typing errors, ran again, crashed, rechecked your typing . . . . Frustrating, wasn't it?

Until now, though, that has been the best way to enter machine language into your machine. Unless you happen to own an assembler and are willing to wrangle with machine language on the assembly level, it is much easier to enter a BASIC program that reads the DATA statements and POKEs the numbers into memory.

Some of these BASIC loaders will use a checksum to see if you've typed the numbers correctly. The simplest checksum is just the sum of all the numbers in the DATA statements. If you make an error, your checksum will not match up. Some programmers have made your task easier by creating checksums every ten lines, so you can zero in on your errors.

But MLX comes to the rescue! The "Machine Language Editor" (MLX) is a great way to enter all those long machine language programs with a mininum of fuss. MLX lets you enter the numbers from a special list that looks similar to BASIC DATA statements. It checks your typing on a line-by-line basis. It won't let you enter illegal characters when you should be typing numbers. It won't let you enter numbers greater than 255. It will prevent you from entering the wrong numbers on the wrong line. In short, MLX will make proofreading obsolete.

Boot Disks

In addition, MLX will generate a ready-to-use tape or disk file. You can then use the LOAD command to read the program into the computer, just like any other program. Specifically, you enter:

LOAD "program", 1, 1 (for tape)

or

LOAD "program", 8, 1 (for disk)

To start the program, you need to enter a SYS command that transfers control from BASIC to machine language. The starting SYS will always be given in the appropriate article.

Using MLX

Type in and save MLX (you'll want to use it in the future). When you're ready to type in the machine language program, RUN MLX. The program will ask you for two numbers: the starting address and the ending address. Below is a table that lists this information for each of the games that use MLX.

Starting and Ending Addresses

Game	Start address	End address	Command to Run
Munchmaze	12288	13956	SYS 12311
Richthofen's Revenge	2049	5817	RUN or SYS 2063
Zuider Zee	49152	52040	

Once you have entered the starting and ending addresses, you'll get a prompt to start entering the data. The prompt is the current line you are entering from the listing. Each line is six numbers plus a checksum. If you enter any of the six numbers wrong, or enter the checksum wrong, the 64 will ring the buzzer and prompt you to reenter the line. If you enter it correctly, a pleasant bell tone will sound, and you go on and enter the next line.

A Special Editor

You are not using the normal Commodore 64 editor with MLX. For example, MLX will accept only numbers as input. If you need to make a correction, press the <INST/DEL> key; the entire number is deleted. You can press it as many times as necessary to get back to the start of the line. If you enter three-digit numbers as listed, the computer will automatically print the comma and go on to accept the next number. If you enter less than three digits, you can press either the comma, space bar, or RETURN key to advance to the next number. The checksum will automatically appear in inverse video; don't worry—it's highlighted for emphasis.

When testing it, I've found it to be extremely easy to enter long listings. With the audio cues provided, you don't even have to look at the screen if you're a touch-typist.

When you get through typing, assuming you type it all in one session, you can then save the completed and bug-free program to tape or disk. Follow the screen instructions. If you get any errors while writing, you probably have a bad disk, or the disk was full, or you made a typo when entering the MLX program. (Sorry, it can't check itself.)

Command Control

What if you don't want to enter the whole program in one sitting? MLX lets you enter as much as you want, save the whole schmeer, and then reLOAD the file from tape or disk when you want to continue. MLX recognizes these few commands:

SHIFT-S: Save
SHIFT-L: Load
SHIFT-N: New Address
SHIFT-D: Display

Hold down SHIFT while you press the appropriate key. You will jump out of the line you've been typing, so I recommend you do it at a new prompt. Use the Save command to save what you've been working on. It will write the tape or disk file as if you've finished, but the tape or disk won't work, of course, until you finish the typing. Remember what address you stop on. The next time you RUN MLX, answer all the prompts as you did before, then insert the disk or tape. When you get to the entry prompt, press SHIFT-L to reLOAD the file into memory. You'll then use the New Address command to resume typing.

New Address and Display

After you press SHIFT-N, enter the address where you previously stopped. The prompt will change, and you can then continue typing. Always enter a New Address that matches up with one of the line numbers in the special listing, or else the checksum won't match up. You can use the Display command to display a section of your typing. After you press SHIFT-D, enter two addresses within the line number range of the listing. You can abort the listing by pressing any key.

Tricky Stuff

The special commands may seem a little confusing, but as you work with MLX, they will become valuable. For example, what if you forgot where you stopped typing? Use the Display command to scan memory from the beginning to the end of the program. When you see a bunch of 170s, stop the listing (press a key) and continue typing where the 170s start. Some programs contain many sections of 170s. To avoid typing them, you can use the New Address command to skip over the blocks of 170s. Be careful, though; you don't want to skip over anything you *should* type.

You can use the Save and Load commands to make copies of the completed game. Use the Load command to reLOAD the tape or disk, then insert a new tape or disk and use the Save command to create a new copy.

One quirk about tapes made with the Save command: when you load them, the message "FOUND program" may appear twice. The tape will load just fine, however.

Programmers will find MLX an interesting program, in protecting the user from mistakes. There is also some screen formatting. Most interesting is the use of ROM Kernal routines for LOADing and SAVEing blocks of memory. Just POKE the starting address (low byte/high byte) into 251 and 252, and POKE the ending address into 254 and 255. Any error code can be found in location 253 (an error would be a code less than ten).

I hope you will find MLX to be a true labor-saving program. Since it has been tested by entering actual programs, you can count on it as an aid for generating bug-free machine language.

MLX

```
100 PRINT"{CLR}{RED}";CHR$(142);CHR$(8);:POKE53281
    ,1:POKE53280,1
101 POKE 788,52:REM DISABLE RUN/STOP
110 PRINT"{RVS}{40 SPACES}";
120 PRINT"{RVS}{15 SPACES}{RIGHT}{OFF}E*3£{RVS}
    {RIGHT} {RIGHT}{2 SPACES}E*3{OFF}E*3£
    {RVS}£{RVS}{13 SPACES}";
```

```
130 PRINT"{RVS}{15 SPACES}{RIGHT} [G]{RIGHT}
    {2 RIGHT} {OFF}£{RVS}£[*]{OFF}[*]{RVS}
    {13 SPACES}";
14Ø PRINT"{RVS}{4Ø SPACES}"
15Ø V=53248:POKE2Ø4Ø,13:POKE2Ø41,13:FORI=832TO894:
    POKEI,255:NEXT:POKEV+27,3
16Ø POKEV+21,3:POKEV+39,2:POKEV+4Ø,2:POKEV,144:POK
    EV+1,54:POKEV+2,192:POKEV+3,54
17Ø POKEV+29,3
18Ø FORI=ØTO23:READA:POKE679+I,A:POKEV+39,A:POKEV+
    4Ø,A:NEXT
185 DATA169,251,166,254,164,255,32,216,255,133,253
    ,96
187 DATA169,Ø,166,251,164,252,32,213,255,133,253,9
    6
19Ø POKEV+39,7:POKEV+4Ø,7
2ØØ PRINT"{2 DOWN}{PUR}{BLK}{3 SPACES}A FAILSAFE M
    ACHINE LANGUAGE EDITOR{5 DOWN}"
21Ø PRINT"[5]{2 UP}STARTING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTS:F=1-F:C$=CHR$(31+119*F)
22Ø IFS<256OR(S>4Ø96ØANDS<49152)ORS>53247THENGOSUB
    3ØØØ:GOTO21Ø
225 PRINT:PRINT:PRINT
23Ø PRINT"[5]{2 UP}ENDING ADDRESS?{8 SPACES}
    {9 LEFT}";:INPUTE:F=1-F:C$=CHR$(31+119*F)
24Ø IFE<256OR(E>4Ø96ØANDE<49152)ORE>53247THENGOSUB
    3ØØØ:GOTO23Ø
25Ø IFE<STHENPRINTC$;"{RVS}ENDING < START
    {2 SPACES}":GOSUB1ØØØ:GOTO 23Ø
26Ø PRINT:PRINT:PRINT
3ØØ PRINT"{CLR}";CHR$(14):AD=S:POKEV+21,Ø
31Ø PRINTRIGHT$("ØØØØ"+MID$(STR$(AD),2),5);":";:FO
    RJ=1TO6
32Ø GOSUB57Ø:IFN=-1THENJ=J+N:GOTO32Ø
39Ø IFN=-211THEN 71Ø
4ØØ IFN=-2Ø4THEN 79Ø
41Ø IFN=-2Ø6THENPRINT:INPUT"{DOWN}ENTER NEW ADDRES
    S";ZZ
415 IFN=-2Ø6THENIFZZ<SORZZ>ETHENPRINT"{RVS}OUT OF
    {SPACE}RANGE":GOSUB1ØØØ:GOTO41Ø
417 IFN=-2Ø6THENAD=ZZ:PRINT:GOTO31Ø
42Ø IF N<>-196 THEN 48Ø
43Ø PRINT:INPUT"DISPLAY:FROM";F:PRINT,"TO";:INPUTT
44Ø IFF<SORF>EORT<SORT>ETHENPRINT"AT LEAST";S;"
    {LEFT}, NOT MORE THAN";E:GOTO43Ø
45Ø FORI=FTOTSTEP6:PRINT:PRINTRIGHT$("ØØØØ"+MID$(S
    TR$(I),2),5);":";
451 FORK=ØTO5:N=PEEK(I+K):PRINTRIGHT$("ØØ"+MID$(ST
    R$(N),2),3);",";
```

```
460 GETA$:IFA$>""THENPRINT:PRINT:GOTO310
470 NEXTK:PRINTCHR$(20);:NEXTI:PRINT:PRINT:GOTO310
480 IFN<0 THEN PRINT:GOTO310
490 A(J)=N:NEXTJ
500 CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT
510 PRINTCHR$(18);:GOSUB570:PRINTCHR$(20)
515 IFN=CKSUMTHEN530
520 PRINT:PRINT"LINE ENTERED WRONG : RE-ENTER":PRI
    NT:GOSUB1000:GOTO310
530 GOSUB2000
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT:POKE54272,0:POK
    E54273,0
550 AD=AD+6:IF AD<E THEN 310
560 GOTO 710
570 N=0:Z=0
580 PRINT"[+]";
581 GETA$:IFA$=""THEN581
585 PRINTCHR$(20);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N670
590 IFA>128THENN=-A:RETURN
600 IFA<>20 THEN 630
610 GOSUB690:IFI=1ANDT=44THENN=-1:PRINT"{LEFT}
    {LEFT}";:GOTO690
620 GOTO570
630 IFA<48ORA>57THEN580
640 PRINTA$;:N=N*10+A-48
650 IFN>255 THEN A=20:GOSUB1000:GOTO600
660 Z=Z+1:IFZ<3THEN580
670 IFZ=0THENGOSUB1000:GOTO570
680 PRINT",";:RETURN
690 S%=PEEK(209)+256*PEEK(210)+PEEK(211)
691 FORI=1TO3:T=PEEK(S%-I)
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT
700 PRINTLEFT$("{3 LEFT}",I-1);:RETURN
710 PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"
720 INPUT"{DOWN} FILENAME";F$
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$
760 OPEN 1,DV,1,F$:POKE252,S/256:POKE251,S-PEEK(25
    2)*256
765 POKE255,E/256:POKE254,E-PEEK(255)*256
770 POKE253,10:SYS 679:CLOSE1:IFPEEK(253)>9ORPEEK(
    253)=0THENPRINT"{DOWN}DONE.":END
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720
```

```
781 OPEN15,8,15:INPUT#15,DS,DS$:PRINTDS;DS$:CLOSE1
    5:GOTO72Ø
79Ø PRINT"{CLR}{RVS}*** LOAD ***{2 DOWN}"
8ØØ INPUT"{2 DOWN} FILENAME";F$
81Ø PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"
82Ø GETA$:IFA$<>"T"ANDA$<>"D"THEN82Ø
83Ø DV=1-7*(A$="D"):IFDV=8THENF$="Ø:"+F$
84Ø OPEN 1,DV,Ø,F$:POKE252,S/256:POKE251,S-PEEK(25
    2)*256
85Ø POKE253,1Ø:SYS 691:CLOSE1
86Ø IFPEEK(253)>9 OR PEEK(253)=Ø THEN PRINT:PRINT:
    GOTO31Ø
87Ø PRINT"{DOWN}ERROR ON LOAD.{2 SPACES}TRY AGAIN.
    {DOWN}":IFDV=1THEN8ØØ
88Ø OPEN15,8,15:INPUT#15,DS,DS$:PRINTDS;DS$:CLOSE1
    5:GOTO8ØØ
1ØØØ REM BUZZER
1ØØ1 POKE54296,15:POKE54277,45:POKE54278,165
1ØØ2 POKE54276,33:POKE 54273,6:POKE54272,5
1ØØ3 FORT=1TO2ØØ:NEXT:POKE54276,32:POKE54273,Ø:POK
     E54272,Ø:RETURN
2ØØØ REM BELL SOUND
2ØØ1 POKE54296,15:POKE54277,Ø:POKE54278,247
2ØØ2 POKE 54276,17:POKE54273,4Ø:POKE54272,Ø
2ØØ3 FORT=1TO1ØØ:NEXT:POKE54276,16:RETURN
3ØØØ PRINTC$;"{RVS}NOT ZERO PAGE OR ROM":GOTO1ØØØ
```

6

Munchmaze

Gary E. Marsa 64 Translation by Gregg Peele

"Munchmaze" is a fast-action strategy game. Since it is written in machine language, it requires the use of the Machine Language Editor (MLX) for mistake-proof entering.

The character in "Munchmaze" hurries through the maze dropping bread crumbs as it goes. You move your character around by using the I, J, K, and M keys trying to munch up as many of the bread crumbs as you can before the character catches you. The game ends when the two characters collide or when you accumulate 10,000 points.

There are three speed levels: slow, moderate, and fast. Both characters move at the same speed, but the computer character beats you on the corners. Also, you have to change directions manually; it doesn't. The computer character always goes left if it can; it's helpful to remember this when you are looking for a temporary hiding place.

There's another tricky feature, too. Sometimes, when the two characters are moving from opposite directions toward each other, the computer character goes right on by and no collision occurs. Just breathe a sigh of relief and continue munching—you were lucky.

The maze in Munchmaze is not constructed on the screen, but in another area of RAM. It is then transferred to the screen, where it appears all at once; then there is a one-second delay before the action begins. If you break out of the program for any reason, just type SYS 12331 and RETURN to restart.

Typing in Munchmaze

This program is written entirely in machine language, so it is necessary to enter it using the Machine Language Editor (MLX) found at the beginning of this chapter. Be sure to read the directions for using the MLX.

The information needed to enter Munchmaze with the MLX is:

Starting address: 12288
Ending address: 13956
Once Munchmaze is saved to disk or tape, the procedure for loading the program is as follows:
From disk: type

LOAD "MUNCHMAZE", 8,1

From tape: type

LOAD"", 1,1

When the program is loaded into memory, type SYS 12311 to run it.

Munchmaze

```
12288 :020,004,010,000,088,178,044
12294 :187,040,171,084,073,041,090
12300 :058,158,049,048,052,056,177
12306 :000,000,000,234,234,169,143
12312 :021,141,024,208,169,000,075
12318 :141,096,010,141,097,010,013
12324 :169,147,032,210,255,169,250
12330 :007,162,000,157,000,216,072
12336 :157,000,217,157,000,218,029
12342 :157,000,219,232,208,241,087
12348 :169,120,141,229,051,169,171
12354 :052,141,230,051,162,031,221
12360 :169,005,032,220,051,169,206
12366 :000,141,098,010,141,099,055
12372 :010,133,162,024,165,161,227
12378 :105,002,133,166,165,161,054
12384 :197,166,208,250,169,147,209
12390 :032,210,255,169,000,162,162
12396 :000,157,000,216,157,000,126
12402 :217,157,000,218,157,000,095
12408 :219,232,208,241,032,168,196
12414 :051,162,039,169,160,157,096
12420 :039,004,157,151,007,202,180
12426 :208,247,169,080,133,168,119
12432 :169,004,133,169,162,021,034
12438 :160,000,169,160,145,168,184
12444 :160,038,145,168,032,250,181
12450 :051,202,208,240,169,119,127
12456 :141,229,051,169,053,141,184
12462 :230,051,162,166,169,004,188
12468 :032,220,051,162,000,169,046
12474 :032,157,000,011,157,000,031
12480 :012,157,000,013,157,000,019
12486 :014,232,208,241,169,081,119
```

```
12492 :133,168,169,040,133,170,249
12498 :169,011,133,169,133,171,228
12504 :162,000,160,000,169,160,099
12510 :145,170,200,192,039,208,152
12516 :249,024,165,170,105,040,213
12522 :133,170,144,002,230,171,060
12528 :232,224,023,208,229,160,036
12534 :000,169,004,145,168,032,252
12540 :151,224,165,143,041,003,211
12546 :133,165,170,010,168,024,160
12552 :185,075,052,101,168,133,210
12558 :180,185,076,052,101,169,009
12564 :133,181,024,185,075,052,158
12570 :101,180,133,170,185,076,103
12576 :052,101,181,133,171,160,062
12582 :000,177,170,201,160,208,186
12588 :018,138,145,170,169,032,204
12594 :145,180,165,170,133,168,243
12600 :165,171,133,169,076,251,253
12606 :048,232,138,041,003,197,209
12612 :165,208,189,177,168,170,121
12618 :169,032,145,168,224,004,048
12624 :240,026,138,010,168,162,056
12630 :002,056,165,168,249,075,033
12636 :052,133,168,165,169,249,004
12642 :076,052,133,169,202,208,170
12648 :238,076,251,048,169,013,131
12654 :141,229,051,169,054,141,127
12660 :230,051,162,006,138,032,223
12666 :220,051,032,228,255,208,092
12672 :251,032,228,255,240,251,105
12678 :201,081,208,032,169,147,204
12684 :032,210,255,169,000,162,200
12690 :000,157,000,216,157,000,164
12696 :217,157,000,218,157,000,133
12702 :219,232,208,241,032,168,234
12708 :051,169,013,076,210,255,170
12714 :201,049,048,211,201,052,164
12720 :016,207,056,233,048,133,101
12726 :166,169,147,032,210,255,137
12732 :169,000,162,000,157,000,164
12738 :216,157,000,217,157,000,173
12744 :218,157,000,219,232,208,210
12750 :241,162,000,189,000,011,041
12756 :157,000,004,189,000,012,062
12762 :157,000,005,189,000,013,070
12768 :157,000,006,189,000,014,078
12774 :157,000,007,232,208,229,039
12780 :032,168,051,024,165,166,074
```

```
12786 :105,176,141,000,004,141,041
12792 :038,004,024,165,162,105,234
12798 :060,133,254,165,162,197,201
12804 :254,208,250,169,081,133,075
12810 :168,133,180,169,004,133,029
12816 :169,133,181,169,001,133,034
12822 :254,162,002,134,165,160,131
12828 :000,169,102,145,168,169,013
12834 :000,133,162,166,165,138,030
12840 :010,168,024,185,075,052,042
12846 :101,168,133,170,185,076,111
12852 :052,101,169,133,171,160,070
12858 :000,177,170,201,160,208,206
12864 :009,202,138,041,003,133,078
12870 :165,076,037,050,201,081,168
12876 :208,003,076,004,051,169,075
12882 :102,145,170,169,058,145,103
12888 :168,165,170,133,168,165,033
12894 :171,133,169,232,138,041,210
12900 :003,133,165,165,254,240,036
12906 :008,160,000,132,254,169,061
12912 :081,145,180,162,000,165,077
12918 :197,221,083,052,240,008,151
12924 :232,224,004,208,246,076,090
12930 :183,050,138,010,168,024,191
12936 :185,075,052,101,180,133,094
12942 :195,185,076,052,101,181,164
12948 :133,196,160,000,177,195,241
12954 :201,160,240,025,201,058,015
12960 :208,003,032,006,052,160,109
12966 :000,169,081,145,195,169,157
12972 :032,145,180,165,195,133,254
12978 :180,165,196,133,181,165,174
12984 :162,197,166,208,250,173,060
12990 :098,010,201,016,208,061,016
12996 :173,099,010,201,039,208,158
13002 :054,169,081,133,168,169,208
13008 :004,133,169,162,000,160,068
13014 :000,177,168,201,058,208,002
13020 :007,032,006,052,169,032,006
13026 :145,168,200,192,037,208,152
13032 :238,032,250,051,232,224,235
13038 :021,208,228,162,000,189,022
13044 :093,054,240,006,157,051,077
13050 :004,232,208,245,076,127,118
13056 :051,076,033,050,169,102,225
13062 :145,170,169,058,145,168,093
13068 :165,180,133,168,165,181,236
13074 :133,169,056,165,168,233,174
```

```
13080 :041,133,170,165,169,233,167
13086 :000,133,171,169,240,133,108
13092 :166,169,255,133,162,165,062
13098 :170,133,180,165,171,133,226
13104 :181,169,000,133,165,160,088
13110 :000,162,000,177,180,221,026
13116 :087,052,240,005,232,224,132
13122 :008,208,246,134,253,056,203
13128 :169,007,229,253,170,189,065
13134 :087,052,145,180,200,192,166
13140 :003,208,224,024,165,180,120
13146 :105,040,133,180,144,002,182
13152 :230,181,230,165,165,165,208
13158 :201,003,208,203,165,162,020
13164 :208,252,198,166,208,179,039
13170 :162,000,189,110,052,240,099
13176 :006,157,055,004,232,208,014
13182 :245,056,173,098,010,237,177
13188 :096,010,141,100,010,173,150
13194 :099,010,237,097,010,013,092
13200 :100,010,240,017,144,015,158
13206 :173,098,010,141,096,010,166
13212 :173,099,010,141,097,010,174
13218 :032,194,051,076,077,048,128
13224 :162,000,189,095,052,240,138
13230 :006,157,006,004,232,208,019
13236 :245,162,000,189,104,052,164
13242 :240,006,157,022,004,232,079
13248 :208,245,172,096,010,173,072
13254 :097,010,032,145,179,032,181
13260 :221,189,162,000,189,000,197
13266 :001,240,006,157,027,004,133
13272 :232,208,245,096,134,168,019
13278 :133,169,162,000,160,000,078
13284 :189,120,052,240,010,201,016
13290 :255,240,012,145,168,200,230
13296 :232,208,241,032,250,051,230
13302 :232,208,233,096,024,165,180
13308 :168,105,040,133,168,144,242
13314 :002,230,169,096,138,072,197
13320 :152,072,024,173,098,010,025
13326 :105,002,141,098,010,144,002
13332 :003,238,099,010,162,000,020
13338 :181,168,072,232,224,008,143
13344 :208,248,172,098,010,173,173
13350 :099,010,032,145,179,032,023
13356 :221,189,162,000,189,000,037
13362 :001,240,006,157,012,004,214
13368 :232,208,245,162,008,104,247
```

```
13374 :149,167,202,208,250,104,118
13380 :168,104,170,096,234,234,050
13386 :234,001,000,216,255,255,011
13392 :255,040,000,037,033,034,223
13398 :036,160,032,058,102,170,132
13404 :186,127,255,019,003,015,185
13410 :018,005,058,032,048,000,003
13416 :008,009,007,008,058,000,194
13422 :135,129,141,133,160,143,183
13428 :150,133,146,000,079,077,189
13434 :032,032,078,080,099,080,011
13440 :032,079,099,079,077,032,014
13446 :079,080,078,099,099,099,156
13452 :077,079,080,032,079,080,055
13458 :000,101,032,077,078,032,210
13464 :103,032,103,032,101,032,043
13470 :101,032,077,101,103,032,092
13476 :032,079,076,100,101,103,143
13482 :032,101,103,000,101,032,027
13488 :032,032,032,103,032,103,254
13494 :032,101,032,101,032,032,000
13500 :032,103,032,032,101,032,008
13506 :032,101,032,099,032,103,081
13512 :000,101,103,077,078,101,148
13518 :103,032,103,100,101,032,165
13524 :101,103,077,032,103,032,148
13530 :032,076,079,099,101,103,196
13536 :099,101,103,000,076,122,213
13542 :032,032,076,122,077,100,157
13548 :100,100,078,076,122,032,232
13554 :077,122,077,100,100,100,050
13560 :078,076,122,032,076,122,242
13566 :000,000,160,223,032,032,189
13572 :233,231,032,233,160,223,092
13578 :032,160,160,160,160,231,145
13584 :160,160,160,160,160,032,080
13590 :002,025,000,160,160,223,080
13596 :233,160,231,233,160,226,247
13602 :160,223,032,032,233,160,106
13608 :105,160,160,000,160,160,017
13614 :160,160,160,231,160,160,053
13620 :098,160,231,032,233,160,198
13626 :105,032,160,160,160,160,067
13632 :032,032,007,001,018,025,179
13638 :000,160,160,095,105,160,238
13644 :231,160,160,226,160,231,220
13650 :233,160,105,032,032,160,036
13656 :160,000,160,160,032,032,120
13662 :160,231,160,160,032,160,229
```

```
13668 :231,160,160,160,160,231,178
13674 :160,160,160,160,160,032,170
13680 :013,001,018,019,001,000,164
13686 :255,032,032,032,009,032,254
13692 :032,032,032,032,013,015,024
13698 :022,005,032,020,008,005,222
13704 :032,034,081,034,032,021,114
13710 :019,009,014,007,000,032,223
13716 :032,032,030,032,032,032,082
13722 :032,032,020,008,005,032,027
13728 :012,005,020,020,005,018,240
13734 :019,058,032,032,032,000,083
13740 :032,032,032,093,000,010,115
13746 :060,067,081,067,062,011,014
13752 :032,032,009,032,061,032,126
13758 :013,015,022,005,032,021,042
13764 :016,000,032,032,032,093,145
13770 :032,032,032,032,032,010,116
13776 :032,061,032,013,015,022,127
13782 :005,032,012,005,006,020,038
13788 :000,032,032,032,022,032,114
13794 :032,032,032,032,011,032,141
13800 :061,032,013,015,022,005,124
13806 :032,018,009,007,008,020,076
13812 :000,032,032,032,013,032,129
13818 :032,032,032,032,013,032,167
13824 :061,032,013,015,022,005,148
13830 :032,004,015,023,014,000,094
13836 :255,160,032,160,032,032,171
13842 :032,032,003,008,015,015,123
13848 :019,005,032,019,016,005,120
13854 :005,004,032,006,001,003,081
13860 :020,015,018,058,000,000,147
13866 :160,032,160,032,032,177,123
13872 :032,061,006,001,019,020,187
13878 :044,032,178,032,061,032,177
13884 :013,015,004,005,018,001,116
13890 :020,005,044,032,179,032,122
13896 :061,032,019,012,015,023,234
13902 :000,000,160,032,160,032,206
13908 :032,032,032,032,032,032,020
13914 :032,032,032,032,032,032,026
13920 :016,018,005,019,019,032,205
13926 :145,032,020,015,032,017,107
13932 :021,009,020,046,000,255,203
13938 :153,143,149,167,146,133,237
13944 :160,129,137,142,142,133,195
13950 :146,161,161,032,255,032,145
13956 :255,255,255,255,255,255,126
```

Richthofen's Revenge

Chris Metcalf Marc Sugiyama

"Richthofen's Revenge" is an arcade-style game that even the most experienced game players will find challenging. This program requires special care to enter correctly; please see the section "Typing in the Program."

The airborne forces of Richthofen, the dreaded Red Baron, have been mobilized. Because of your reputation as a swift pilot and accurate gunner, you have been chosen to defend the front line. Only a few planes are available, with no time to build more. Prepare yourself to meet the hordes of Richthofen.

As you encounter each succeeding wave of the enemy, another airplane will be delivered to the front. Once all the planes have been destroyed, however, there will be nothing to stop the enemy from an all-out invasion. Your skills are all that stand between Richthofen's forces and your country.

Typing in the Program

This program is written entirely in machine language, so it should be entered using the Machine Language Editor (MLX) found earlier in this chapter.

The steps to typing in a machine language program using MLX are simple, but they must be followed exactly in order to get a playable copy of the game. Once you have a copy of "Richthofen's Revenge" saved on disk or tape, you will be able to LOAD and RUN it just as you would any BASIC program even though it is machine language.

The steps for typing in Richthofen's Revenge are:

1. Reset the computer by turning it off, then back on.
2. Type this line:
   POKE 44,23:POKE 23*256,0:NEW
3. LOAD the Machine Language Editor into memory. (If you have not typed in and SAVEd MLX, you will have to do that first.)

4. RUN the MLX program.
5. Answer the prompts
   START ADDR: 2049
   END ADDR: 5817
6. Type in the data.
7. MLX will prompt you for a filename.
8. Before you load the program, reset the computer.

That's all there is to it. It is not necessary to type in all the data in one session. The instructions for using MLX are at the beginning of this chapter. If you do decide to enter the data in more than one session, it will be necessary to follow the above steps each time you begin a session.

Preparing for Battle

When you first RUN the game, the screen will come up with a landscape, a status line, and the message RICHTHOFEN'S REVENGE. The information given in the status line is the high score, the score of the current game, and the number of backup planes remaining. A short tune will play to prepare you for the combat.

When the message PRESS FIRE TO BEGIN appears, you may begin playing or move to a higher level. By moving the joystick up or down, you can pick any level from 1 to 30. Levels 31 through 40 are reserved for expert players, and the levels above that are only for the true masters.

Once you have selected a level, or at any point after the music begins, you may press the fire button and begin playing. Every time you enter a level, or when a new plane is called up, you begin at the very top of the screen. This area is off-limits to Richthofen's forces due to their limited flight ceilings. However, once you go down into their midst, you too are sealed off from this high-altitude bracket for the duration of the level.

Your Opponents

Richthofen is employing three types of aircraft. Surveillance balloons patrol the areas they have been assigned to in accordance with random wind currents. These have been judged least important by the Air Force (50 points each). The remainder of the enemy forces consists of two types of aircraft: the main attack force, consisting of blue-green planes which always fly west, and the equally important red spy craft. Both types are worth 75 points each.

Some strategies and tips have been given to you by Air Force command. Although your aircraft can dodge mountains and the like without any danger, a number of civilian residences are scattered throughout the combat area. These present a very definite threat to navigation. You can neither fly nor fire through them. Furthermore, the explosions of the enemy craft are deadly to you.

Air Force Briefing

The Air Force has also given you a short list of pointers for fighting the enemy. You will find that balloons are often extremely difficult to hit. This problem may be at least partially remedied by the use of the rapid-fire aspect of your controller. Holding down the fire button will cause your machine gun to fire rapidly after a slight initial delay. At times you may find yourself flying on and on without encountering any enemy craft. Often the problem is that the few surviving enemy fighter planes are going in the same direction as yourself. In such cases, simply turn and wait for them. To determine how far you are from the end of a level, consult the table below.

One final item is of some importance to you as a fighter pilot. The first planes sent out to you were of undeniably high quality and workmanship. The components were all painstakingly hand-formed, and the result was an airplane that could achieve an unusually high speed—enough, in fact, to overtake even the enemy fighter pilots. But as the production of these airplanes increased, the quality declined. Thus as you continue to play, you will find that your planes lose efficiency, until after a number of levels your top speed is barely that of the enemy planes.

Several keyboard controls have been included in the program. Pressing f7 causes all game action and sound to stop until the key is pressed again or the fire button pushed. RUN/STOP has the same effect. F8 ends the program, leaving your country to Richthofen's mercy. F3 turns the sound of your engines on and off, but leaves the noise of shooting and explosions as always. F1 functions as a reset key, checking for a high score then returning you to the initial display.

A variety of melodies has been included in the program. All of them may be skipped by pressing the fire button on your joystick.

Levels of Play

Play Level	Number of Enemy	Accumulated Score
1	12	650
2	16	1650
3	20	2900
4	24	4400
5	24	5900
6	24	7400
7	24	9000
8	28	10800
9	28	12600
10	28	14400
11	28	16300
12	32	18400
13	32	20500
14	32	22700
15	36	25100
16	36	27500
17	36	30000
18	40	32600
19	40	35200
20	40	37700
21	44	39800
22	44	42750
23	44	45550
24	48	48650
25	48	51850
26	48	55100
27	52	58500
28	52	61900
29	52	65450
30	56	69150
31	56	72850
32	56	76650
33	60	80650
34	60	84750
35	60	88850
36	64	93050
37	64	97300
38	64	101350
39	64	105725
40	64	110100
41	64	113300
42	64	118100
43	64	122900
44	64	127275
45	64	130475
etc.		

Richthofen's Revenge

```
2049 :013,008,100,000,158,040,064
2055 :050,048,054,051,041,000,251
2061 :000,000,165,001,041,254,218
2067 :133,001,169,197,141,000,148
2073 :221,169,000,141,023,208,019
2079 :141,029,208,141,027,208,017
2085 :032,006,017,169,003,141,149
2091 :178,002,169,027,141,017,065
2097 :208,169,172,141,000,208,179
2103 :169,011,141,032,208,169,017
2109 :014,141,033,208,169,000,114
2115 :141,034,208,169,012,141,004
2121 :035,208,162,127,169,000,006
2127 :157,000,168,202,016,250,104
2133 :162,032,189,033,017,157,163
2139 :015,168,189,066,017,157,191
2145 :079,168,202,016,241,160,195
2151 :000,185,099,019,153,000,047
2157 :176,200,192,144,208,245,250
2163 :160,000,185,179,019,153,043
2169 :000,177,200,192,168,208,042
2175 :245,160,000,185,091,020,060
2181 :153,000,178,200,192,208,040
2187 :208,245,169,160,141,032,070
2193 :164,169,000,141,000,164,015
2199 :168,185,000,164,024,105,029
2205 :040,153,001,164,185,032,220
2211 :164,105,000,153,033,164,014
2217 :200,192,025,208,234,169,173
2223 :000,141,175,002,141,176,042
2229 :002,141,177,002,169,255,159
2235 :141,168,002,169,001,133,033
2241 :033,120,169,127,141,013,028
2247 :220,169,001,141,026,208,196
2253 :169,000,141,018,208,173,146
2259 :017,208,041,127,141,017,250
2265 :208,173,020,003,141,123,117
2271 :016,173,021,003,141,124,189
2277 :016,169,072,141,020,003,138
2283 :169,016,141,021,003,088,161
2289 :169,000,141,172,002,141,098
2295 :173,002,141,174,002,169,140
2301 :004,141,183,002,173,168,156
2307 :002,141,167,002,169,001,229
2313 :141,033,208,169,160,141,093
2319 :136,002,169,147,032,210,199
2325 :255,169,004,141,136,002,216
2331 :169,006,141,033,208,160,232
```

```
2337 :023,185,105,021,153,154,162
2343 :163,136,016,247,169,081,083
2349 :141,187,163,160,002,162,092
2355 :000,185,175,002,032,202,135
2361 :015,157,179,163,165,021,245
2367 :157,180,163,232,232,136,139
2373 :016,237,169,001,133,027,140
2379 :238,167,002,173,167,002,056
2385 :201,043,144,005,169,039,170
2391 :141,167,002,160,063,169,021
2397 :255,153,064,164,136,016,113
2403 :250,172,167,002,185,215,066
2409 :021,072,141,180,002,185,194
2415 :172,021,072,024,109,180,177
2421 :002,141,180,002,185,129,244
2427 :021,170,024,109,180,002,117
2433 :141,180,002,160,000,224,068
2439 :000,240,009,169,001,153,195
2445 :064,164,200,202,208,247,202
2451 :104,170,240,009,169,002,073
2457 :153,064,164,200,202,208,120
2463 :249,104,170,240,009,169,076
2469 :003,153,064,164,200,202,183
2475 :208,249,160,063,032,148,007
2481 :015,041,031,056,233,010,051
2487 :048,246,201,002,144,242,042
2493 :153,192,164,032,148,015,125
2499 :153,128,164,136,016,230,254
2505 :169,130,133,025,032,006,184
2511 :017,173,183,002,032,202,048
2517 :015,141,188,163,165,021,138
2523 :141,189,163,169,049,141,047
2529 :001,208,169,141,141,024,141
2535 :208,169,003,141,039,208,231
2541 :169,011,141,032,208,169,199
2547 :000,141,028,208,169,160,181
2553 :141,248,163,169,001,141,088
2559 :021,208,169,000,141,182,208
2565 :002,133,032,173,031,208,072
2571 :165,027,208,010,165,033,107
2577 :240,003,032,241,010,076,107
2583 :006,011,173,178,002,141,022
2589 :179,002,032,237,013,169,149
2595 :000,133,027,160,018,169,030
2601 :001,153,153,217,152,024,229
2607 :105,128,153,154,161,162,142
2613 :007,185,043,021,032,221,050
2619 :010,202,016,247,136,016,174
2625 :230,162,000,032,131,016,124
```

```
2631 :224,051,208,111,162,000,059
2637 :160,018,185,062,021,032,043
2643 :221,010,136,016,247,173,118
2649 :000,220,201,111,240,091,184
2655 :142,169,002,162,100,032,190
2661 :245,015,174,169,002,232,170
2667 :224,008,144,222,173,167,021
2673 :002,024,105,001,032,179,200
2679 :015,032,202,015,141,243,255
2685 :161,165,021,141,244,161,250
2691 :173,000,220,201,111,240,052
2697 :048,201,126,208,017,238,207
2703 :167,002,173,167,002,201,087
2709 :030,144,026,169,000,141,147
2715 :167,002,240,019,201,125,141
2721 :208,224,206,167,002,173,117
2727 :167,002,201,255,208,005,237
2733 :169,029,141,167,002,162,075
2739 :060,032,245,015,076,111,206
2745 :010,160,018,169,032,153,215
2751 :154,161,136,016,250,141,025
2757 :243,161,141,244,161,173,040
2763 :000,220,201,127,208,249,184
2769 :206,167,002,173,167,002,158
2775 :141,168,002,076,075,009,174
2781 :010,010,010,141,235,010,125
2787 :152,010,010,010,141,238,020
2793 :010,189,000,178,157,000,255
2799 :180,096,169,000,141,005,062
2805 :212,169,240,141,006,212,201
2811 :169,001,141,001,212,169,176
2817 :033,141,004,212,096,173,148
2823 :000,220,073,127,133,036,084
2829 :169,000,133,031,165,036,035
2835 :041,001,240,010,173,001,229
2841 :208,201,059,144,019,206,094
2847 :001,208,165,036,041,002,228
2853 :240,010,173,001,208,201,102
2859 :214,176,003,238,001,208,115
2865 :165,036,041,004,240,034,057
2871 :169,001,133,031,169,161,207
2877 :141,248,163,173,178,002,198
2883 :041,007,201,007,240,006,057
2889 :238,178,002,076,127,011,193
2895 :169,000,141,179,002,198,000
2901 :025,032,063,014,165,036,164
2907 :041,008,240,032,169,001,070
2913 :133,031,169,160,141,248,211
2919 :163,173,178,002,041,007,155
```

```
2925 :240,006,206,178,002,076,049
2931 :127,011,169,007,141,179,237
2937 :002,230,025,032,063,014,231
2943 :173,178,002,141,179,002,034
2949 :165,031,208,007,169,000,201
2955 :141,000,212,240,005,169,138
2961 :064,141,000,212,238,182,214
2967 :002,169,012,056,237,167,026
2973 :002,048,004,201,007,176,083
2979 :002,169,007,205,182,002,218
2985 :240,012,165,031,208,016,073
2991 :162,001,032,245,015,076,194
2997 :191,011,032,056,014,169,142
3003 :000,141,182,002,173,031,204
3009 :208,208,003,076,129,012,061
3015 :032,006,017,169,141,141,193
3021 :019,212,169,000,141,020,254
3027 :212,141,014,212,169,070,005
3033 :141,015,212,169,008,141,135
3039 :018,212,169,129,141,018,142
3045 :212,173,183,002,072,248,095
3051 :056,233,001,216,141,183,041
3057 :002,169,002,141,039,208,034
3063 :169,000,133,029,133,028,227
3069 :133,030,141,032,208,165,194
3075 :029,024,105,006,133,029,073
3081 :144,002,230,030,165,028,096
3087 :024,101,029,133,028,008,082
3093 :173,001,208,101,030,141,163
3099 :001,208,176,027,040,176,143
3105 :004,165,030,240,008,173,141
3111 :032,208,073,002,141,032,015
3117 :208,173,178,002,141,179,158
3123 :002,032,056,014,076,002,233
3129 :012,104,169,003,141,039,013
3135 :208,169,011,141,032,208,064
3141 :162,255,032,245,015,104,114
3147 :240,003,076,205,009,162,002
3153 :104,032,131,016,160,002,014
3159 :185,172,002,217,175,002,072
3165 :144,016,208,003,136,016,104
3171 :243,160,002,185,172,002,095
3177 :153,175,002,136,016,247,066
3183 :169,000,141,024,212,162,051
3189 :255,032,245,015,169,015,080
3195 :141,024,212,076,241,008,057
3201 :165,032,240,002,198,032,030
3207 :165,036,041,016,208,007,096
3213 :169,080,133,032,076,138,001
```

```
3219 :013,165,032,201,079,240,109
3225 :008,201,000,208,243,169,214
3231 :030,133,032,169,009,141,161
3237 :012,212,169,000,141,013,200
3243 :212,141,007,212,169,030,174
3249 :141,008,212,169,129,141,209
3255 :011,212,160,018,173,248,237
3261 :163,056,233,160,141,170,088
3267 :002,208,002,160,021,162,238
3273 :007,169,000,157,080,176,022
3279 :202,016,250,173,001,208,033
3285 :056,233,002,141,169,002,048
3291 :041,007,170,169,085,157,080
3297 :080,176,173,169,002,074,131
3303 :074,074,056,233,005,141,046
3309 :169,002,170,032,131,015,244
3315 :177,251,201,043,144,004,039
3321 :201,046,144,046,201,052,171
3327 :176,012,201,046,176,032,130
3333 :201,007,144,004,201,010,060
3339 :144,024,169,010,145,251,242
3345 :169,001,145,253,192,000,009
3351 :240,012,136,173,170,002,244
3357 :208,212,200,200,192,040,057
3363 :144,206,140,171,002,076,006
3369 :089,013,140,171,002,162,106
3375 :063,189,192,164,205,169,005
3381 :002,240,006,202,016,245,252
3387 :076,089,013,173,171,002,071
3393 :024,101,025,221,128,164,216
3399 :208,239,189,064,164,201,112
3405 :004,176,232,142,181,002,046
3411 :206,180,002,032,004,016,011
3417 :162,005,032,245,015,169,205
3423 :032,172,171,002,192,020,172
3429 :240,013,145,251,136,174,036
3435 :170,002,240,244,200,200,139
3441 :076,099,013,162,001,032,240
3447 :245,015,173,031,208,169,192
3453 :000,141,011,212,173,178,072
3459 :002,141,179,002,032,063,038
3465 :014,165,197,201,004,208,158
3471 :003,076,085,012,201,005,013
3477 :208,020,165,033,073,001,137
3483 :133,033,240,005,032,241,071
3489 :010,208,061,169,008,141,246
3495 :004,212,208,054,201,003,081
3501 :240,004,201,063,208,050,171
3507 :173,141,002,240,015,169,151
```

```
3513 :Ø71,141,Ø24,ØØ3,169,254,Ø79
3519 :141,Ø25,ØØ3,169,ØØØ,133,15Ø
3525 :198,ØØØ,Ø36,197,Ø8Ø,252,192
3531 :169,ØØØ,141,Ø24,212,173,154
3537 :ØØØ,22Ø,2Ø1,111,24Ø,Ø14,227
3543 :165,197,2Ø1,Ø63,24Ø,ØØ4,Ø61
3549 :2Ø1,ØØ3,2Ø8,239,Ø36,197,Ø81
3555 :Ø8Ø,252,169,Ø15,141,Ø24,14Ø
3561 :212,Ø76,ØØ6,Ø11,165,Ø26,217
3567 :141,169,ØØ2,165,Ø25,141,114
3573 :17Ø,ØØ2,16Ø,ØØØ,173,171,153
3579 :ØØ2,24Ø,ØØ5,Ø44,Ø17,2Ø8,255
3585 :Ø16,251,173,179,ØØ2,141,251
3591 :178,ØØ2,174,169,ØØ2,189,2Ø9
3597 :Ø99,Ø17,17Ø,Ø32,131,Ø15,221
3603 :169,Ø32,145,251,174,17Ø,192
3609 :ØØ2,189,Ø99,Ø18,Ø72,189,Ø82
3615 :Ø99,Ø17,17Ø,Ø32,131,Ø15,239
3621 :169,Ø13,145,253,1Ø4,145,Ø98
3627 :251,238,169,ØØ2,238,17Ø,Ø87
3633 :ØØ2,2ØØ,192,Ø4Ø,2Ø8,21Ø,133
3639 :Ø96,169,ØØØ,141,171,ØØ2,122
3645 :24Ø,ØØ5,169,ØØ1,141,171,Ø2Ø
3651 :ØØ2,Ø32,237,Ø13,162,ØØØ,ØØ1
3657 :142,169,ØØ2,174,169,ØØ2,219
3663 :189,Ø64,164,2Ø1,Ø16,144,Ø89
3669 :Ø71,2Ø1,Ø57,144,Ø58,16Ø,ØØ8
3675 :Ø15,173,18Ø,ØØ2,2Ø8,Ø43,2ØØ
3681 :236,181,ØØ2,2Ø8,Ø38,169,163
3687 :ØØ3,141,Ø39,2Ø8,169,Ø12,163
3693 :141,Ø32,2Ø8,248,173,183,Ø7Ø
3699 :ØØ2,Ø24,1Ø5,ØØ1,141,183,Ø59
37Ø5 :ØØ2,216,165,Ø25,133,Ø26,176
3711 :162,Ø52,Ø32,131,Ø16,169,177
3717 :Ø11,141,Ø32,2Ø8,Ø76,Ø75,164
3723 :ØØ9,Ø72,1Ø4,136,2Ø8,251,151
3729 :Ø76,113,Ø15,Ø74,Ø74,Ø74,Ø59
3735 :Ø24,1Ø5,ØØ3,254,Ø64,164,253
3741 :141,184,ØØ2,188,128,164,196
3747 :14Ø,185,ØØ2,188,192,164,Ø1Ø
3753 :14Ø,186,ØØ2,172,171,ØØ2,Ø74
3759 :2Ø8,1Ø5,2Ø1,ØØ3,24Ø,Ø55,219
3765 :2Ø1,ØØ2,24Ø,Ø32,176,Ø95,159
3771 :Ø32,173,Ø15,24Ø,ØØ3,Ø32,17Ø
3777 :255,Ø14,Ø32,173,Ø15,24Ø,154
3783 :Ø82,Ø32,173,Ø15,24Ø,ØØ6,235
3789 :2Ø6,185,ØØ2,2Ø6,185,ØØ2,223
3795 :238,185,ØØ2,Ø76,Ø26,Ø15,241
38Ø1 :238,185,ØØ2,Ø32,148,Ø15,Ø69
```

```
3807 :041,007,208,055,032,255,053
3813 :014,206,185,002,076,026,226
3819 :015,206,185,002,032,148,055
3825 :015,041,007,208,036,032,068
3831 :255,014,238,185,002,076,249
3837 :026,015,032,173,015,240,242
3843 :011,173,186,002,201,002,066
3849 :240,014,206,186,002,096,241
3855 :173,186,002,201,021,240,070
3861 :003,238,186,002,096,189,223
3867 :128,164,056,229,026,201,063
3873 :040,176,017,168,189,192,047
3879 :164,170,032,131,015,169,208
3885 :032,145,251,136,048,002,147
3891 :145,251,173,185,002,056,095
3897 :229,025,201,040,176,035,251
3903 :168,174,186,002,032,131,244
3909 :015,174,184,002,189,089,210
3915 :021,145,253,138,024,105,249
3921 :042,145,251,136,048,011,202
3927 :189,080,021,145,253,138,145
3933 :024,105,032,145,251,174,056
3939 :169,002,173,185,002,157,019
3945 :128,164,173,186,002,157,147
3951 :192,164,238,169,002,173,025
3957 :169,002,201,064,240,003,028
3963 :076,076,014,165,025,133,100
3969 :026,096,189,000,164,133,225
3975 :251,133,253,189,032,164,133
3981 :133,252,073,120,133,254,082
3987 :096,056,165,140,101,143,080
3993 :101,144,133,139,138,072,112
3999 :162,004,181,139,149,140,166
4005 :202,016,249,104,170,165,047
4011 :139,096,032,148,015,041,130
4017 :001,096,162,000,134,020,078
4023 :162,008,248,010,072,165,080
4029 :020,101,020,133,020,104,075
4035 :202,208,244,165,020,216,226
4041 :096,072,041,015,024,105,042
4047 :071,133,021,104,074,074,172
4053 :074,074,024,105,071,133,182
4059 :020,096,160,002,162,000,147
4065 :185,172,002,032,202,015,065
4071 :157,160,163,165,021,157,030
4077 :161,163,232,232,136,016,153
4083 :237,096,152,072,160,255,191
4089 :072,104,136,208,251,202,198
4095 :208,248,104,168,096,188,243
```

```
4101 :064,164,185,098,021,248,017
4107 :024,109,172,002,141,172,119
4113 :002,173,173,002,105,000,216
4119 :141,173,002,173,174,002,176
4125 :105,000,141,174,002,216,155
4131 :169,016,157,064,164,032,125
4137 :221,015,169,138,141,019,232
4143 :212,169,000,141,020,212,033
4149 :141,014,212,169,060,141,022
4155 :015,212,169,008,141,018,110
4161 :212,169,129,141,018,212,178
4167 :096,173,025,208,141,025,227
4173 :208,041,001,240,043,162,004
4179 :233,173,178,002,009,016,182
4185 :168,173,018,208,016,004,164
4191 :162,006,160,000,142,018,071
4197 :208,173,017,208,041,127,107
4203 :141,017,208,140,022,208,075
4209 :173,013,220,041,001,240,033
4215 :005,198,002,076,049,234,171
4221 :104,168,104,170,104,064,071
4227 :032,006,017,232,189,002,097
4233 :022,141,169,002,202,160,065
4239 :000,189,002,022,153,003,000
4245 :212,153,010,212,153,017,138
4251 :212,232,200,192,004,208,179
4257 :238,169,002,141,170,002,115
4263 :169,212,133,021,032,016,238
4269 :017,172,170,002,185,102,053
4275 :021,133,020,160,000,189,190
4281 :002,022,145,020,232,200,038
4287 :192,002,208,245,189,002,005
4293 :022,240,062,232,142,171,042
4299 :002,170,160,004,169,008,204
4305 :145,020,173,169,002,145,095
4311 :020,134,002,172,000,220,251
4317 :192,111,208,011,032,006,013
4323 :017,172,000,220,192,111,171
4329 :240,249,096,166,002,208,170
4335 :234,041,254,160,004,145,053
4341 :020,172,170,002,136,016,249
4347 :002,160,002,140,170,002,215
4353 :174,171,002,208,168,160,116
4359 :023,169,000,153,000,212,052
4365 :136,016,250,169,008,141,221
4371 :004,212,141,011,212,141,228
4377 :018,212,169,015,141,024,092
4383 :212,096,248,000,000,160,235
4389 :031,252,144,001,000,136,089
```

```
4395 :002,124,135,255,249,128,168
4401 :000,005,127,000,007,017,205
4407 :240,133,000,015,249,000,180
4413 :002,032,000,001,192,000,032
4419 :000,031,063,248,005,000,158
4425 :128,009,062,064,017,159,000
4431 :255,225,160,000,001,224,176
4437 :000,254,161,015,136,159,042
4443 :240,000,004,064,000,003,146
4449 :128,000,022,022,022,022,057
4455 :022,022,022,022,022,022,235
4461 :022,022,022,022,022,022,241
4467 :022,022,022,022,022,022,247
4473 :022,022,022,022,022,022,253
4479 :022,022,022,022,022,022,003
4485 :022,022,022,022,022,022,009
4491 :022,021,021,020,019,018,004
4497 :017,016,015,014,014,015,236
4503 :016,017,018,019,019,020,004
4509 :020,019,018,017,017,016,008
4515 :016,016,017,017,017,017,007
4521 :017,018,019,020,021,021,029
4527 :021,020,020,020,019,019,038
4533 :018,018,018,018,019,019,035
4539 :020,021,021,021,021,021,056
4545 :021,021,021,021,021,021,063
4551 :021,021,021,020,020,019,065
4557 :018,017,016,016,015,015,046
4563 :015,016,016,016,016,017,051
4569 :017,017,017,017,017,017,063
4575 :017,016,016,016,016,017,065
4581 :018,018,017,017,016,016,075
4587 :015,015,014,013,013,012,061
4593 :011,010,009,008,007,007,037
4599 :006,006,006,006,007,008,030
4605 :009,010,011,012,013,014,066
4611 :015,016,017,018,019,020,108
4617 :021,022,022,022,022,022,140
4623 :022,022,021,021,020,020,141
4629 :020,020,020,019,018,017,135
4635 :016,016,015,014,014,013,115
4641 :013,013,013,013,014,014,113
4647 :015,015,016,016,017,018,136
4653 :018,019,019,020,020,021,162
4659 :021,021,021,021,021,021,177
4665 :020,020,019,019,018,018,171
4671 :017,017,016,016,015,015,159
4677 :014,014,013,013,013,013,149
4683 :013,014,014,015,015,016,162
```

```
4689 :Ø17,Ø18,Ø19,Ø19,Ø2Ø,Ø2Ø,194
4695 :Ø2Ø,Ø2Ø,Ø2Ø,Ø21,Ø21,Ø21,21Ø
47Ø1 :Ø2Ø,Ø2Ø,Ø2Ø,Ø21,Ø21,Ø22,217
47Ø7 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,Ø99
4713 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,1Ø5
4719 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,111
4725 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,117
4731 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,123
4737 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,129
4743 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØ3,138
4749 :ØØ4,ØØ1,ØØ1,ØØ1,ØØ1,ØØ1,15Ø
4755 :ØØ1,ØØ1,ØØ2,ØØ2,ØØ2,ØØ2,157
4761 :ØØ2,ØØ5,ØØ6,ØØØ,ØØØ,ØØ1,167
4767 :ØØ1,ØØ3,ØØ4,ØØ1,ØØØ,ØØ2,17Ø
4773 :ØØ5,ØØ6,ØØ7,ØØ8,ØØ9,ØØ2,2Ø2
4779 :ØØ2,ØØ2,ØØØ,ØØØ,ØØØ,ØØ3,178
4785 :ØØ4,ØØØ,ØØ3,ØØ4,ØØ1,ØØØ,189
4791 :ØØØ,ØØ2,ØØ5,ØØ6,ØØ2,ØØØ,198
4797 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,189
48Ø3 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,195
48Ø9 :ØØØ,ØØ3,ØØ4,ØØ1,ØØ1,ØØ1,211
4815 :ØØ1,ØØØ,ØØ7,ØØ8,ØØ9,ØØØ,232
4821 :ØØØ,ØØ5,ØØ6,ØØØ,ØØØ,ØØØ,224
4827 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØ3,222
4833 :ØØ4,ØØØ,ØØ2,ØØ2,ØØØ,ØØØ,233
4839 :ØØ3,ØØ4,ØØ3,ØØ4,ØØ3,ØØ4,252
4845 :ØØ1,ØØ3,ØØ4,ØØ1,ØØ1,ØØ1,248
4851 :ØØ1,ØØ1,ØØ3,ØØ4,ØØ3,ØØ4,ØØ3
4857 :ØØØ,ØØ2,ØØ2,ØØ2,ØØ2,ØØ2,ØØ3
4863 :ØØ2,ØØ2,ØØ2,ØØ2,ØØ2,ØØ2,Ø11
4869 :ØØ2,ØØ2,ØØ2,ØØ2,ØØ2,ØØØ,Ø15
4875 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,Ø11
4881 :ØØ3,ØØ4,ØØ7,ØØ8,ØØ9,ØØ3,Ø51
4887 :ØØ4,ØØ1,ØØ1,ØØ1,ØØ3,ØØ4,Ø37
4893 :ØØ1,ØØ3,ØØ4,ØØ3,ØØ4,ØØØ,Ø44
4899 :ØØ5,ØØ6,ØØ5,ØØ6,ØØ5,ØØ6,Ø68
49Ø5 :ØØ5,ØØ6,ØØ2,ØØ5,ØØ6,ØØ5,Ø7Ø
4911 :ØØ6,ØØ5,ØØ6,ØØØ,ØØØ,ØØØ,Ø64
4917 :ØØØ,ØØØ,ØØØ,ØØØ,ØØ3,ØØ4,Ø6Ø
4923 :ØØ3,ØØ4,ØØ3,ØØ4,ØØ3,ØØ4,Ø8Ø
4929 :ØØ3,ØØ4,ØØ3,ØØ4,ØØ3,ØØ4,Ø86
4935 :ØØ3,ØØ4,ØØØ,ØØ5,ØØ6,ØØ5,Ø94
4941 :ØØ6,ØØ5,ØØ6,ØØ2,ØØ2,ØØ2,1ØØ
4947 :ØØ5,ØØ6,ØØØ,ØØØ,ØØØ,ØØ5,Ø99
4953 :ØØ6,ØØØ,ØØØ,ØØØ,ØØ7,ØØ8,11Ø
4959 :ØØ9,ØØ5,ØØ6,ØØØ,Ø85,ØØØ,2ØØ
4965 :ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,ØØØ,1Ø1
4971 :ØØ1,ØØ1,ØØ4,ØØ4,Ø16,Ø16,149
4977 :Ø64,Ø64,Ø64,Ø64,Ø16,Ø16,145
```

```
4983 :004,004,001,001,000,000,129
4989 :000,000,001,004,016,064,210
4995 :001,004,016,064,000,000,216
5001 :000,000,064,016,004,001,222
5007 :000,000,000,000,000,000,143
5013 :000,000,064,016,004,001,234
5019 :005,005,024,024,106,038,101
5025 :038,070,000,000,067,067,147
5031 :147,128,128,149,000,192,143
5037 :240,240,240,128,128,149,018
5043 :000,000,000,000,000,000,179
5049 :000,000,003,015,015,015,233
5055 :003,001,002,002,194,192,073
5061 :240,058,042,010,002,000,037
5067 :063,008,032,255,249,246,032
5073 :063,008,000,000,001,003,028
5079 :003,001,000,000,000,003,222
5085 :007,015,015,007,003,000,012
5091 :003,015,062,058,058,062,229
5097 :015,003,000,003,006,012,016
5103 :012,006,003,000,000,000,004
5109 :001,002,002,001,000,000,251
5115 :000,000,000,001,001,000,253
5121 :000,000,000,000,000,000,001
5127 :000,000,000,000,192,240,183
5133 :240,240,192,064,128,128,237
5139 :168,048,012,170,218,122,245
5145 :168,048,194,002,010,248,183
5151 :252,240,192,000,000,000,203
5157 :128,192,192,128,000,000,165
5163 :000,192,224,240,240,224,139
5169 :192,000,192,240,188,172,009
5175 :172,188,240,192,000,192,015
5181 :096,048,048,096,192,000,029
5187 :000,000,128,064,064,128,195
5193 :000,000,000,000,000,128,201
5199 :128,000,000,000,000,000,207
5205 :000,000,000,000,000,000,085
5211 :238,238,238,254,254,238,015
5217 :238,238,254,254,056,056,169
5223 :056,056,254,254,124,254,077
5229 :224,238,238,238,254,124,145
5235 :126,254,224,252,126,014,087
5241 :254,252,124,254,238,224,187
5247 :224,238,254,124,252,254,193
5253 :238,254,252,238,238,238,055
5259 :254,254,224,252,252,224,063
5265 :254,254,124,254,238,238,227
5271 :238,238,254,124,056,120,157
```

```
5277  :248,Ø56,Ø56,Ø56,254,254,Ø57
5283  :252,254,Ø14,Ø28,112,224,Ø23
5289  :254,254,252,254,Ø14,124,Ø41
5295  :124,Ø14,254,252,Ø14,Ø3Ø,Ø95
53Ø1  :126,238,254,254,Ø14,Ø14,Ø57
53Ø7  :254,254,224,252,254,Ø14,159
5313  :254,124,124,252,224,252,143
5319  :254,238,254,124,254,254,Ø41
5325  :Ø14,Ø28,Ø56,Ø56,Ø56,Ø56,215
5331  :124,254,238,124,254,238,163
5337  :254,124,124,254,238,254,185
5343  :126,Ø14,126,124,ØØØ,ØØØ,1Ø1
5349  :114,Ø38,254,24Ø,ØØØ,ØØØ,1Ø7
5355  :254,254,Ø56,Ø56,Ø56,Ø56,199
5361  :Ø56,Ø56,254,254,224,252,Ø57
5367  :252,224,224,224,2Ø6,238,Ø79
5373  :254,254,254,254,238,23Ø,2Ø1
5379  :Ø28,Ø56,112,ØØØ,ØØØ,ØØØ,199
5385  :ØØØ,ØØØ,238,238,238,238,193
5391  :238,238,124,Ø56,252,254,153
5397  :238,254,252,224,224,224,157
54Ø3  :252,254,238,254,252,238,235
54Ø9  :254,252,ØØØ,ØØØ,ØØØ,ØØØ,Ø27
5415  :ØØØ,ØØØ,ØØØ,ØØØ,ØØ5,ØØ1,Ø45
5421  :ØØ4,ØØØ,Ø18,ØØ7,Ø19,ØØ6,Ø99
5427  :Ø2Ø,Ø21,ØØ3,Ø25,ØØ5,ØØ6,131
5433  :Ø22,ØØ6,Ø2Ø,ØØ2,ØØ6,Ø23,136
5439  :ØØ5,ØØ6,ØØ3,ØØ3,Ø25,Ø19,124
5445  :ØØ1,ØØ5,ØØ6,Ø25,Ø18,ØØ7,131
5451  :Ø25,Ø24,ØØ6,ØØ2,ØØ1,Ø2Ø,153
5457  :ØØ9,Ø15,Ø11,ØØ1,ØØ1,ØØ9,127
5463  :ØØ3,ØØ4,ØØ2,ØØ9,Ø11,Ø11,127
5469  :ØØ1,ØØ1,ØØ9,ØØ3,ØØ4,ØØ2,113
5475  :Ø8Ø,117,117,ØØØ,ØØ7,Ø14,178
5481  :Ø67,Ø68,Ø71,Ø69,Ø7Ø,Ø32,226
5487  :Ø71,Ø71,Ø71,Ø71,Ø71,Ø71,Ø25
5493  :Ø32,Ø32,Ø64,Ø65,Ø66,Ø64,184
5499  :Ø32,Ø67,Ø68,Ø71,Ø69,Ø7Ø,244
55Ø5  :ØØ4,ØØ8,Ø1Ø,Ø12,Ø12,Ø12,187
5511  :ØØ8,Ø12,Ø12,Ø12,ØØ8,Ø12,199
5517  :Ø12,ØØ8,Ø12,Ø12,ØØ8,Ø16,2Ø9
5523  :Ø16,Ø2Ø,Ø18,Ø14,Ø2Ø,Ø2Ø,255
5529  :Ø16,Ø14,Ø2Ø,Ø2Ø,Ø14,Ø2Ø,ØØ1
5535  :Ø2Ø,Ø16,Ø2Ø,Ø16,Ø16,Ø24,Ø15
5541  :Ø22,Ø3Ø,Ø17,Ø17,ØØØ,ØØØ,251
5547  :Ø64,ØØ3,ØØ4,ØØ4,ØØ2,ØØ6,254
5553  :Ø1Ø,ØØ8,ØØ8,ØØ6,Ø1Ø,ØØ4,223
5559  :Ø1Ø,Ø14,Ø1Ø,Ø12,ØØ6,Ø1Ø,245
5565  :Ø12,Ø16,Ø1Ø,Ø18,Ø16,Ø12,Ø17
```

```
5571 :014,016,020,016,024,014,043
5577 :018,014,020,020,022,026,065
5583 :020,016,017,017,030,000,051
5589 :064,000,003,004,006,010,044
5595 :006,002,008,008,010,006,003
5601 :016,010,006,014,012,018,045
5607 :018,012,008,010,008,014,045
5613 :012,014,016,014,016,008,061
5619 :024,018,022,020,020,022,113
5625 :018,020,026,017,030,017,121
5631 :064,000,000,000,033,076,172
5637 :201,031,021,015,143,010,170
5643 :015,024,014,060,031,021,176
5649 :015,143,010,015,024,014,238
5655 :060,031,021,015,143,010,047
5661 :015,024,014,030,031,021,164
5667 :015,143,010,015,024,014,000
5673 :030,031,021,015,143,010,035
5679 :015,024,014,060,000,000,160
5685 :000,000,033,076,201,209,060
5691 :018,015,000,000,005,209,050
5697 :018,010,031,021,010,209,108
5703 :018,010,195,016,010,210,018
5709 :015,011,195,016,012,209,023
5715 :018,021,165,031,022,165,249
5721 :031,023,049,028,024,030,018
5727 :025,042,041,021,043,209,220
5733 :018,044,000,000,000,002,165
5739 :065,128,249,071,006,020,134
5745 :071,006,010,097,008,060,109
5751 :071,006,020,097,008,010,075
5757 :143,010,060,071,006,020,179
5763 :097,008,010,143,010,030,173
5769 :071,006,020,097,008,010,093
5775 :143,010,030,071,006,020,167
5781 :097,008,010,143,010,060,221
5787 :097,008,020,143,010,010,187
5793 :143,012,060,143,010,020,037
5799 :097,008,010,071,006,060,163
5805 :071,006,020,071,006,010,101
5811 :097,008,060,000,000,000,088
5817 :000,000,000,000,000,000,185
```

6

Zuider Zee

Marc Sugiyama

Your mission is to save your village from flooding. This BASIC and machine language game will provide hours of fun. Requires the use of the MLX program.

Your village in Holland is built on land reclaimed from the ocean. High dikes keep the cold waters of the North Sea from flooding your land. But word has come that a terrible storm is approaching. Heavy rains and giant waves will undoubtedly break down sections of the dikes, flooding parts, perhaps all, of your land.

But you are prepared. You and your fellow Dutchmen have been battling the sea for centuries. In the old days, bucket brigades and sandbaggers would have fought the storm, and many lives might have been lost. Times have changed. Helicopters will rescue all the people whose homes are flooded, and as for repairing damage to the dikes and pumping out the water, that can all be done by one person. You.

You Are the Dikemaster

As dikemaster, you are responsible for repairing the dikes and pumping out the floodwaters.

You have a truck with the latest landfill equipment, so that all you have to do is back it into place where you want to repair a broken dike. The truck does the rest.

You also have four pumps. When a dike has been repaired, you then have to pick up one of the pumps and put it in place on the dike. Then you set it up to pump water from the flooded fields and dump it back into the ocean. But be careful. If you set the pump wrong, it can pump water from the ocean and pour it onto land, making the flood worse than ever.

When you have successfully repaired all the dikes and pumped out all the water, you can't relax. You immediately get a promotion, and have to do the same for another village, where the storm is even worse.

And if you ever get so far behind that all your land is flooded at the same time—well, you can certainly understand why your fellow villagers will start looking for a new dikemaster.

How to Play

At the beginning of the game, you will be asked to choose a starting level. Until you get the hang of driving the truck and setting up the pumps, you'd probably better start at level 1, in which the storm is pretty mild and new gaps don't open up so often. Later, though, you can try higher levels.

The village. At the beginning of the game, the screen is filled with plowed fields, trees, and houses. The dikes are built, with the dikemaster's depot in the middle. Then the sea covers all the land outside the dikes. Finally, several breaks open in the dikes, and sections of the village lands are flooded. It's time for you to get to work!

Scoring. Scoring depends on several factors: how much land is covered with water; what level you are playing at; and how long you can keep the storm from entirely flooding the village.

Moving the truck. You drive your truck along the tops of the dikes by using the joystick. The dikes are slightly wider than the truck, so you can maneuver a little from side to side. You can't accidentally drive the truck off the dike.

Repairing the dike. Drive the truck to a break in the dike. You will want to dump a load of dirt into the break, to block it. Hold down the joystick button. This puts the truck in reverse. When you move the joystick, the truck will back up, moving the *opposite* direction from the direction in which you moved the joystick.

As long as you keep pressing the button and moving the joystick, the truck will keep backing up. When it reaches the edge of the dike, it stops and dumps a load of dirt off the edge of the dike. This creates a new section of dike. If you steered the truck correctly, the new section will repair the break in the dike. If not, you'll just have an extra load of dirt that doesn't connect with anything.

Your truck constantly scoops up more dirt as you drive from place to place—you will never run out of material to repair the dike.

Pumping out the water. Once a flooded area is completely surrounded by the dike, with no breaks, you can begin pumping. First, you must go and pick up a pump. At the beginning of the game, all four pumps are just outside the depot. Drive on top of the pumps, push the joystick button, and your truck will automatically pick up a pump. Then drive to the edge of the flooded field you want to drain.

You will need to place the pump on the dike between the flooded field and the place where you want the water to be dumped. Usually you will want the water to be dumped in the ocean, but sometimes you will dump from one flooded field to another, or even from a flooded field to a field with no water on it.

You place the pump by holding down the joystick button and then moving the joystick in the direction where you want the pump to dump the water. Remember, move the joystick in the direction where you want the floodwater to end up.

You will hear the sound of the pump starting up, and when you drive away, the pump will stay behind.

At any time you can go to a pump and pick it up by driving right onto it and pushing the joystick button *without moving*. You'll always hear the sound of the truck picking up the pump. Then, when you release the button and drive away, the pump will go with you.

How pumps work. The pumps are just machines. They aren't very smart. If you set a pump to pick up water from the ocean and dump it onto a field, the pump will do exactly that, and the flooding will get worse instead of better. You'll also come closer to losing your job.

However, if you set the pump to draw water from a field that isn't flooded, or set it so that it dumps onto or picks up from the dike instead of a field or the ocean, nothing will happen at all.

The joystick button. When you move the joystick without pressing the button, the truck drives around.

Pressing down the joystick button can do one of three things:

If you do not move the joystick, and the truck is touching a pump but not already carrying one, the truck will pick up that pump and the pump will stop functioning.

If you move the joystick when the truck is not carrying a pump, the truck will go into reverse and move the opposite way from the direction you are moving the joystick. As soon as it reaches the edge of the dike, it will dump a load of dirt.

If you move the joystick when the truck is carrying a pump, the truck will unload the pump and, if possible, begin dumping water where the joystick movement indicated.

Getting promoted. If you ever uncover all the land of your village, you will be promoted and moved to the next village. The game will stop, and the new village will be drawn on the screen. You will be at a harder level of play, which means that breaks will occur more often, and more land will be flooded at the beginning of play. However, you will also get more points at the higher levels.

Strategy Tips

At lower levels of play, it is possible to repair all the dikes and completely pump out all the water. At higher levels, however, the storm is too intense, and dikes break too often. Here the best strategy is to choose four relatively small enclosures, set a pump on each, and then spend the rest of your time repairing breaks in the dikes as often as possible. The pumps will function whenever the field they are pumping is completely enclosed by dike walls. Since the game ends as soon as all the fields are completely flooded, it's better to keep one area dry, sacrificing the others, than to overextend yourself.

You can also take advantage of the fact that your truck will create a dike section wherever you want it. It is possible to build whole new dikes and create new fields. It is also possible to divide a large field into several smaller ones by building new dikes across it. This is particularly helpful at higher levels, when the dike breaks so often that you can't keep a large field completely enclosed long enough for it to be pumped dry.

Typing in the Program

Most of the program is written in BASIC, but certain key routines are written in machine language and must be entered and SAVEd using the Machine Language Editor (MLX) found at the beginning of this chapter.

The MLX is a program that checks your DATA statements as you enter them and prevents you from entering the data incorrectly. Several other games in this chapter and programs in other COMPUTE! books for the Commodore 64 use the MLX program, so if you type it in once and SAVE it, you will use it again and again to enter error-free machine language programs.

The first step is to enter and SAVE the machine language routines using MLX. The MLX will ask you for two numbers. Answer the prompts as follows:

Starting address: 49152
Ending address: 52040

Then start entering the data using the instructions given with the MLX program.

The next step is to type in and SAVE the BASIC program. The best way to save the two parts of this program is to save the machine language on a tape first and then save the BASIC part immediately after the machine language program.

Loading the Program

Once you have both parts of the program SAVEd, you are ready to LOAD the program. First LOAD the machine language as follows:

From disk: LOAD"fn",8,1
From tape: LOAD"",1,1

where fn is the filename.

Type NEW and LOAD the BASIC part as you would any other BASIC program. To begin play, type RUN and the game will begin.

Program 1. Zuider Zee: Part 1. BASIC

```
100 FORI=ØTO27:POKE54272+I,Ø:NEXT:POKE53264,Ø
110 IFPEEK(49161)<>76THENPRINT"{DOWN}?NO MACHINE L
    ANGUAGE{2 SPACES}ERROR";:END
120 PRINT"{CLR}{BLK}@@@@@@"
130 SYS49161:SYS49164:POKE53272,4:POKE648,128
140 PRINT"{CLR}{GRN}"CHR$(8)CHR$(14):POKE53280,Ø:P
    OKE53281,Ø
150 POKE55,Ø:POKE56,128:CLR
160 GOSUB1Ø4Ø
170 :
180 REM MAIN LOOP
190 SYS49167:SYS4917Ø
200 IFPEEK(9Ø8)THENPOKE851,1:GOSUB41Ø:POKE851,Ø
210 IFPEEK(844)=ØTHEN22Ø
215 PN=PEEK(844)-2:POKE851,1:GOSUB66Ø:PF(PN)=PF:PO
    KE844,Ø:POKE851,Ø:GOSUB56Ø
220 IFPEEK(845)=ØTHEN26Ø
230 PF(PEEK(845)-2)=Ø:POKE845,Ø
240 POKEFQ,2Ø:POKEAD,Ø:POKESR,243:POKECT,17:POKECT
    ,16
250 GOSUB56Ø
260 IFPEEK(9Ø8)=ØANDPEEK(851)=ØTHEN3ØØ
270 FORPN=1TO4:IFPF(PN)=ØTHEN29Ø
280 GOSUB67Ø:PF(PN)=PF
290 NEXT:POKE851,Ø:POKE9Ø8,Ø
300 GOSUB87Ø:SYSHM:H1=FND(69Ø):P=H1/HØ
310 IFP>=1THEN241Ø
```

```
320 IFINT(P*100)<3THEN2610
330 GOSUB560:SC=SC+INT(MD*P)
340 GETA$:IFA$=""THEN190
350 IFA$="Q"THEN2620
360 IFA$<>"{F1}"THEN190
370 POKE53280,14:POKE834,0:POKE198,0:WAIT198,1:POK
    E198,0:POKE834,1:POKE53280,6
380 GOTO190
390 :
400 REM FLOOD
410 FS=FND(900):IS=FND(902)
420 X=PEEK(680):Y=PEEK(681)
430 IT=PEEK(907):FI=PEEK(906):FL=PEEK(909)-33:TL=P
    EEK(910)-33
440 IFFL<0ORTL>14ORFL>14THENRETURN
450 IFTL<0THENTL=0
460 POKEX+Y*40+S,11:IFFIANDITTHENLV=40:GOTO510
470 IFTL=FLTHENLV=TL:GOTO510
480 IFFIORITTHENLV=7:GOTO510
490 POKEFQ,8:POKEAD,0:POKESR,122:POKECT,129
500 LV=(TL*IS+FL*FS)/(IS+FS)
510 IFLV=0THENLV=7
520 SYSFM,X,Y,31,14:SYSFM,X,Y,LV+33,14:POKECT,128
530 RETURN
540 :
550 REM STATUS LINE
560 POKE214,23:PRINT:PC=-(P>.25)-(P>.50)-(P>.75)-(
    P>1)+1
570 PRINT"{RVS}{YEL} RANK:"MID$(STR$(SK),2)" SCORE
    :";
580 PRINTTAB(14)RIGHT$("000000"+MID$(STR$(INT(SC/1
    0)*10),2),6);
590 PRINT" ST:"MID$("{RED}{CYN}{YEL}{GRN}{WHT}",PC
    ,1)" {YEL}";
600 PRINT" PUMPS:";:FORI=1TO4:PRINTTAB(I*2+30);
610 IFPF(I)THENPRINTMID$("{RED}{CYN}{PUR}{GRN}",I,
    1)MID$(STR$(I),2);:GOTO630
620 PRINT"{YEL} ";
630 NEXT:PRINT"{HOME}":RETURN
640 :
650 REM START/CHECK PUMP
660 XP(PN)=PEEK(848):YP(PN)=PEEK(849):DP(PN)=PEEK(
    850)
670 PF=0:X=XP(PN):Y=YP(PN):D=DP(PN)
680 FP(PN)=X+40*Y+S-D(D):TP(PN)=X+40*Y+S+D(D)
690 FC=PEEK(FP(PN)):IFFC=11ORFC=32ORFC=31THENFC=40
700 FC=FC-33:IFFC<0ORFC>14THENRETURN
710 TC=PEEK(TP(PN)):IFTC=11ORTC=32ORTC=31THENTC=40
720 TC=TC-33:IFTC<0THENTC=0
```

```
730 IFTC>14THENRETURN
740 NX=X-XD(D):NY=Y-YD(D):SYSFM,NX,NY,11,14
750 SYSFM,NX,NY,FC+33,14
760 MF(PN)=Ø:IFPEEK(9Ø5)=ØTHENMF(PN)=1/FND(69Ø)*(8
    -SK/2)
770 NX=X+XD(D):NY=Y+YD(D):SYSFM,NX,NY,11,14
780 IFPEEK(FP(PN))=11THENSYSFM,NX,NY,TC+33,14:MF(P
    N)=Ø:MT(PN)=Ø:GOTO81Ø
790 SYSFM,NX,NY,TC+33,14
800 MT(PN)=Ø:IFPEEK(9Ø5)=ØTHENMT(PN)=1/FND(69Ø)*(8
    -SK/2)
810 IFPF(PN)THEN84Ø
820 FL(PN)=Ø:TL(PN)=Ø
830 POKEFQ,3Ø:POKEAD,Ø:POKESR,243:POKECT,17:POKECT
    ,16
840 PF=1:RETURN
850 :
860 REM OPERATE PUMPS
870 FORI=1TO4:IFPF(I)=ØTHEN1Ø1Ø
880 C1=Ø:CF=Ø:FL(I)=FL(I)+MF(I):TL(I)=TL(I)+MT(I)
890 IFFL(I)<1THEN92Ø
900 CF=1:FL(I)=FL(I)-1:FC=PEEK(FP(I))-34:IFFC<ØTHE
    NFC=-33:PF(I)=Ø
910 IFFC>14THENPF(I)=Ø:FC=14
920 IFTL(I)<1THEN95Ø
930 C1=1:TL(I)=TL(I)-1:TC=PEEK(TP(I))-32:IFTC>14TH
    ENTC=14:PF(I)=Ø
940 IFTC<ØTHENPF(I)=Ø:TC=Ø
950 IFMF(I)=ØORCF=ØTHEN98Ø
960 XN=XP(I)-XD(DP(I)):YN=YP(I)-YD(DP(I))
970 POKE851,1:SYSFM,XN,YN,11,12:SYSFM,XN,YN,FC+33,
    14+(FC=-33):POKE851,Ø
980 IFMT(I)=ØORC1=ØTHEN1Ø1Ø
990 XN=XP(I)+XD(DP(I)):YN=YP(I)+YD(DP(I))
1000 POKE851,1:SYSFM,XN,YN,11,12:SYSFM,XN,YN,TC+33
     ,14:POKE851,Ø
1010 NEXT:RETURN
1020 :
1030 REM INITIALIZE
1040 PRINT"{CLR}";
1050 JY=5632Ø:IFPEEK(1Ø24)=ØTHENGOSUB2Ø9Ø
1060 I=RND(-RND(Ø))
1070 DIM XØ(7),X1(7),YØ(7),Y1(7),XP(4),YP(4),DP(4)
1080 DIM PF(4),FL(4),TL(4),FP(4),TP(4),MT(4),MF(4)
1090 S=32768:C=22528:FM=49152:BX=49155:HM=49158
1100 FQ=5428Ø:AD=54284:SR=54285:CT=54283
1110 DEFFNR(X)=INT(RND(1)*X)
1120 DEFFND(X)=PEEK(X)+256*PEEK(X+1)
1130 REM SPRITE DATA
```

```
1140 IFPEEK(1024)THENFORI=1TO605:READA:NEXT:GOTO12
     20
1150 POKE1024,1:FORI=0TO25:SQ=34816+I*64:J=0
1160 READA:IFA<0THENSQ=SQ-A:J=J-A:GOTO1180
1170 POKESQ,A:SQ=SQ+1:J=J+1
1180 IFJ<63THEN1160
1190 NEXT
1200 PRINTSPC(5)" PRESS THE TRIGGER TO CONTINUE
     {UP}":GOSUB2750
1210 GOSUB1990
1220 POKE53272,8
1230 REM CHAR DATA
1240 PRINT"{CLR}{GRN}":FORI=1TO12:READB:FORJ=0TO7:
     READA
1250 POKE4096+B*8+J,A:NEXT:NEXT
1260 FORI=0TO3:READXD(I),YD(I):NEXT:FORI=0TO3:READ
     D(I):NEXT
1270 POKE53280,6:POKE53281,0:SK=PEEK(1026):GOSUB13
     60
1280 POKE53269,251
1290 A=0:FORI=53254TO53260STEP2:POKEI,162+A:POKEI+
     1,132:A=A+2:NEXT
1300 A=2:FORI=53290TO53293:POKEI,A:A=A+1:NEXT:FORI
     =33786TO33790:POKEI,46:NEXT
1310 TT=0:MD=3+SK*2:MR=26+52*(SK-1)
1320 SC=FND(1027)*10
1330 POKE904,MD:POKE912,MR:POKE834,1:RETURN
1340 :
1350 REM MAKE ISLAND
1360 POKE214,23:PRINT:PRINT"{RVS}{YEL}{39 SPACES}
     {HOME}";
1370 POKE33767,160:POKE56295,7
1380 GOSUB560
1390 SYSFM,RND(1)*40,RND(1)*25,0,13
1400 SYSBX,18,9,21,12,64,14:SYSFM,19,10,1,9:POKES,
     0:POKES+C,13
1410 POKE419+S,1:POKE420+S,2:POKE459+S,64:POKE459+
     S+C,13
1420 POKE460+S,3:POKE460+S+C,13
1430 REM DAMS
1440 FORI=0TO7
1450 X0=FNR(10)*3:X1=X0+(FNR(10)+1)*3:IFX0=0ORX1>3
     8THEN1450
1460 Y0=FNR(7)*3:Y1=Y0+(FNR(7)+1)*3:IFY0=0ORY1>23T
     HEN1460
1470 SYSBX,X0,Y0,X1,Y1,64,14:X0(I)=X0:Y0(I)=Y0:X1(
     I)=X1:Y1(I)=Y1:NEXT
1480 SYSFM,0,0,5,13
1490 REM TREES/HOUSES
```

```
1500 FORI=1TO30
1510 X=FNR(37)+1:Y=FNR(22)+1:T=X+Y*40+S
1520 IFPEEK(T)=0ORPEEK(T)=5THENPOKET,3
1530 NEXT
1540 FORI=0TO9
1550 X=FNR(37)+1:Y=FNR(22)+1:T=X+Y*40+S
1560 IFPEEK(T)THEN1550
1570 POKET,4:POKET+C,11:POKE52320+I,X:POKE52352+I,
     Y:NEXT
1580 REM WATER/AMOUNT LAND
1590 SYSFM,0,0,40,14:SYSHM:H0=FND(690):ID=INT(H0*S
     K/10)
1600 REM FIRST BREAKS
1610 R=0:K=0:F2=0:NT=4
1620 FORI=0TO7:GOSUB1760:IFR=5THENI=8
1630 NEXT:K=K+1:IFR<5ANDK<5THEN1620
1640 REM EXTRA BREAKS
1650 F2=1:SYSFM,0,0,32,14:SYSHM:H1=FND(690):IFH0-I
     D>H1THEN1720
1660 SYSFM,0,0,40,14
1670 I=(I+1)AND7:GOSUB1760:IFFTHEN1670
1680 IFF1=0THENPOKET,64:GOTO1670
1690 SYSFM,0,0,32,14:SYSHM:H2=FND(690)
1700 IFH1-H2<3THENPOKET,64:GOTO1660
1710 IFH0-ID<H2+1THENH1=H2:GOTO1660
1720 SYSFM,0,0,40,14
1730 RETURN
1740 :
1750 REM MAKE BREAK
1760 F1=0:F=1:J=0:DI=(RND(1)>.5)
1770 IFDITHEN1800
1780 Y0=Y0(I):X=X0(I):IFRND(1)>.5THENX=X1(I)
1790 GOTO1810
1800 X0=X0(I):Y=Y0(I):IFRND(1)>.5THENY=Y1(I)
1810 J=J+1:IFJ>NTTHENRETURN
1820 IFDITHEN1890
1830 Y1=(Y+FNR(Y1(I)-Y-2)+1):T=Y1*40+X+S
1840 IF(Y1>9ANDY1<13)AND(X=18ORX=21)THEN1810
1850 IFPEEK(T+1)=64ORPEEK(T-1)=64THEN1810
1860 IFPEEK(T+40)<>64ORPEEK(T-40)<>64ORPEEK(T)<>64
     THEN1810
1870 IFF2ANDPEEK(T+1)=40ANDPEEK(T-1)=40THEN1810
1880 GOTO1930
1890 X1=X+FNR(X1(I)-X-2)+1:T=X1+Y*40+S:IF(X1>17AND
     X1<22)AND(Y=9ORY=12)THEN1810
1900 IFPEEK(T+40)=64ORPEEK(T-40)=64THEN1810
1910 IFPEEK(T+1)<>64ORPEEK(T-1)<>64ORPEEK(T)<>64TH
     EN1810
1920 IFF2ANDPEEK(T+40)=40ANDPEEK(T-40)=40THEN1810
```

```
1930 PL=Ø:F=Ø
1940 IFPEEK(T+4Ø)=4ØORPEEK(T-4Ø)=4ØORPEEK(T+1)=4ØO
     RPEEK(T-1)=4ØTHENPL=4Ø:F1=1
1950 IFF2THENPL=6
1960 POKET+C,13-(PL=4Ø):POKET,PL:R=R+1:RETURN
1970 :
1980 REM SKILL/TITLE
1990 SK=1:PRINT"{CLR}"SPC(14)CHR$(142)"{GRN}
     {2 DOWN}ZUIDER{2 SPACES}ZEE":PRINTSPC(14)"
     {3 DOWN}RANK NUMBER"
2000 PRINTSPC(12)"{DOWN}{RVS} 1 {OFF} 2{2 SPACES}3
     {2 SPACES}4{2 SPACES}5 "
2010 J=PEEK(JY):IFJ=127THEN2Ø1Ø
2020 PRINTSPC(9+SK*3)"{UP}"SK"{LEFT} "
2030 IF(JAND4)=ØTHENSK=SK-1:IFSK<1THENSK=5
2040 IF(JAND8)=ØTHENSK=SK+1:IFSK>5THENSK=1
2050 PRINTSPC(9+SK*3)"{UP}{RVS}"SK"{LEFT} ":IF(JAN
     D16)THEN2Ø1Ø
2060 PRINT"{CLR}":POKE1Ø26,SK:RETURN
2070 :
2080 REM DO INSTRUCTION SCREEN FOR POWER UP
2090 PRINT"{CLR}{4 DOWN}"SPC(14)"ZUIDER ZEE"
2100 PRINT"{DOWN} YOUR VILLAGE IN HOLLAND IS BUILT
      ON"
2110 PRINT" LAND RECLAIMED FROM THE SEA.{2 SPACES}
     HIGH"
2120 PRINT" DIKES HAVE KEPT THE WATER FROM FLOOD-"
2130 PRINT" ING YOUR LAND, BUT NOW A TERRIBLE"
2140 PRINT" STORM IS APPROACHING.{2 SPACES}HEAVY R
     AINS AND"
2150 PRINT" GIANT WAVES WILL UNDOUBTEDLY BREAK"
2160 PRINT" DOWN SECTIONS OF THE DIKES, FLOODING"
2170 PRINT" YOUR LAND.{2 SPACES}AS DIKEMASTER YOU
     {SPACE}ARE"
2180 PRINT" RESPONSIBLE FOR REPAIRING THE DIKES"
2190 PRINT" AND PUMPING OUT THE FLOODWATERS."
2200 PRINTSPC(5)"{2 DOWN}PRESS THE TRIGGER TO CONT
     INUE:":GOSUB275Ø
2210 :
2220 PRINT"{CLR}{DOWN} USE THE JOYSTICK TO DRIVE T
     HE TRUCK"
2230 PRINT" ALONG THE TOPS OF THE DIKES.{2 SPACES}
     YOU"
2240 PRINT" MAY NOT DRIVE OVER DIKES THAT ARE"
2250 PRINT" DECAYING.{2 SPACES}REMEMBER THAT YOU C
     AN"
2260 PRINT" REPAIR THE DIKES BY DUMPING DIRT ON"
2270 PRINT" THEM.{2 SPACES}THE DIKES BEGIN TO DECA
     Y BEFORE"
```

```
2280 PRINT" THERE ARE VISIBLE SIGNS OF DAMAGE."
2290 PRINT"{DOWN} PRESSING THE TRIGGER CAN HAVE ON
     E OF"
2300 PRINT" THREE EFFECTS:":PRINT"{DOWN} 1) PICK U
     P THE PUMP UNDER THE TRUCK."
2310 PRINT" 2) DROP THE PUMP UNDER THE TRUCK."
2320 PRINT" 3) PUT THE TRUCK IN REVERSE AND DUMP A
     "
2330 PRINT"{4 SPACES}LOAD OF DIRT AT THE EDGE OF A
      DIKE."
2340 PRINT"{DOWN} THERE ARE TWO KEYBOARD CONTROLS:
     "
2350 PRINT"{DOWN} 1) 'F1' PAUSES THE GAME; PRESSIN
     G ANY"
2360 PRINT"{4 SPACES}KEY WILL RESUME YOUR GAME."
2370 PRINT" 2) 'Q' QUITS THE GAME."
2380 PRINTSPC(6)"{DOWN}THE STORM IS APPROACHING!
     {UP}":RETURN
2390 :
2400 REM NEXT LEVEL
2410 POKE834,0:PRINT"{HOME}{RVS}{BLU}{4 SPACES}YOU
      HAVE UNFLOODED THE ISLAND!!":POKE54283,0
2420 POKEFQ,20:POKEAD,0:POKESR,240
2430 B=0:FORI=0TO9:B=B-(PEEK(52384+I)=0):NEXT:B=IN
     T(P*1000*SK)+B*500
2435 FORSC=SCTOSC+BSTEP75:GOSUB560:POKECT,33:POKEC
     T,32:NEXT
2440 FORI=1TO750:NEXT
2450 PRINT"{CLR}{GRN}":POKE53272,6:POKE53280,0:POK
     E53281,0:POKE53269,0
2460 PRINT"SCORE:"MID$(STR$(INT(SC/10)*10),2)
2470 PRINT"{2 DOWN} YOU HAVE SUCCEDED IN RECOVERIN
     G THE"
2480 PRINT" ENTIRE ISLAND.{2 SPACES}YOU ARE BEING
     {SPACE}PROMOTED"
2490 PRINT" BECAUSE OF THIS GREAT ACCOMPLISHMENT!"
2500 PRINT"{DOWN} ANOTHER, STRONGER STORM IS APPRO
     ACHING"
2510 PRINT" AND YOU HAVE BEEN GIVEN AN ISLAND THAT
     "
2520 PRINT" HAS SUFFERED GREATER DAMAGE THAN YOUR"
2530 PRINT" FIRST ASSIGNMENT."
2540 PRINTSPC(15)"{DOWN}GOOD LUCK!"
2550 PRINTSPC(7)"{4 DOWN}PRESS THE TRIGGER TO BEGI
     N"
2560 POKE1027,(SC/10)AND255:POKE1028,SC/2560
2570 GOSUB2750:SK=SK+1:IFSK>5THENSK=5
2580 POKE1026,SK:RUN
2590 :
```

```
2600 REM ISLAND FLOODED
2610 POKE834,0:PRINT"{HOME}{RVS}{BLU}{4 SPACES}THE
      ENTIRE ISLAND HAS FLOODED!!":POKE54283,0
2620 POKEFQ,15:POKEAD,17:POKESR,250:POKECT,33
2630 FORI=1TO250:NEXT:POKECT,32:FORI=1TO750:NEXT
2640 POKE834,0:PRINT"{CLR}{GRN}":POKE53272,6:POKE5
     3280,0:POKE53281,0:POKE53269,0
2650 PRINT"{DOWN}YOUR SCORE WAS:"MID$(STR$(INT(SC/
     10)*10),2)
2660 PRINT"{2 DOWN}YOU HAVE FAILED TO SAVE THE ISL
     AND"
2670 PRINT"AND HAVE BEEN RELIEVED OF YOUR"
2680 PRINT"POSITION AS DIKEMASTER."
2690 PRINT"{DOWN}PERHAPS ALL YOU NEED IS MORE PRAC
     TICE;"
2700 PRINT"WOULD YOU LIKE TO TRY AGAIN?"
2710 PRINTSPC(4)"{2 DOWN}PRESS THE TRIGGER TO BEGI
     N AGAIN"
2715 PRINT"{DOWN}{2 SPACES}'Q' TO QUIT AND 'I' FOR
      INSTRUCTIONS"
2716 GETA$:IFA$="Q"THENPOKE648,4:POKE1024,0:SYS102
     4
2717 IFA$<>"I"THEN2719
2718 GOSUB2090:PRINTSPC(6)"PRESS TRIGGER TO CONTIN
     UE":GOSUB2750:GOTO2640
2719 IFPEEK(JY)AND16THEN2716
2720 IF(PEEK(JY)AND16)=0THEN2720
2721 POKE1027,0:POKE1028,0:SC=0:GOSUB2750:GOSUB199
     0:RUN
2730 :
2740 REM TRIGGER?
2750 IFPEEK(JY)AND16THEN2750
2760 IF(PEEK(JY)AND16)=0THEN2760
2770 RETURN
2780 :
2790 REM SPRITE IMAGES
2800 REM TRUCK{2 SPACES}(U/D/L/R)
2810 DATA -25,60,-2,126,-2,126,-2,126,-2,126,-25
2820 DATA -25,126,-2,126,-2,126,-2,126,-2,60,-25
2830 DATA -25,63,-2,255,-2,255,-2,255,-2,63,-25
2840 DATA -25,252,-2,255,-2,255,-2,255,-2,252,-25
2850 :
2860 REM COPTER (U/D//L/UL/DL//R/UR/DR)
2870 DATA -10,60,-2,102,-2,102,-2,126,-2,126,-2,60
     ,-2,60,-2,60,-2
2880 DATA 24,-2,24,-2,24,-2,24,-2,24,-2,8,-2,56,-1
     0
2890 DATA -10,56,-2,8,-2,24,-2,24,-2,24,-2,24,-2,2
     4
```

```
2900 DATA -2,60,-2,60,-2,60,-2,126,-2,126,-2,102,-
     2,102,-2,60,-10, -63
2910 DATA -24,7,128,0,15,240,0,9,255,224,9,255,160
     ,15,240,32,7,128,0,-21
2920 DATA -9,3,128,0,7,192,0,12,96,0,12,112,0,7,24
     0,0,3,248,-2,124,-2
2930 DATA 60,-2,14,-2,7,-2,3,128,0,1,192,-2,96,-2,
     192,-12
2940 DATA -14,192,-2,96,0,1,192,0,3,128,0,7,-2,14,
     -2,60,-2,124,0
2950 DATA 3,248,0,7,240,0,12,112,0,12,96,0,7,192,0
     ,3,128,-10, -63
2960 DATA -22,1,224,4,15,240,5,255,144,7,255,144,0
     ,15,240,0,1,224,-24
2970 DATA -10,3,128,0,7,192,0,12,96,0,28,96,0,31,1
     92,0,63,128,0,62,-2,60,-2
2980 DATA 112,-2,224,0,1,192,0,3,128,0,6,-2,3,-14
2990 DATA -9,3,-2,6,-2,3,128,0,1,192,-2,224,-2,112
     ,-2,60,-2,62,-2,63,128
3000 DATA 0,31,192,0,28,96,0,12,96,0,7,192,0,3,128
     ,-12
3010 :
3020 REM PUMPS (U/D/L/R)
3030 DATA -22,24,-2,60,-2,126,-2,255,-2,219,-2,24,
     -2,24,-2,24,-19
3040 DATA -22,24,-2,24,-2,24,-2,219,-2,255,-2,126,
     -2,60,-2,24,-19
3050 DATA -22,24,-2,56,-2,112,-2,255,-2,255,-2,112
     ,-2,56,-2,24,-19
3060 DATA -22,24,-2,28,-2,14,-2,255,-2,255,-2,14,-
     2,28,-2,24,-19
3070 :
3080 REM COPTER ROTOR (FRAMES 0-7)
3090 DATA 0,24,-2,24,-2,24,-2,24,-2,24,-2,24,-2,24
     ,-2,24,-2,24,-2,24,-2,24
3100 DATA -2,24,-2,24,-2,24,-2,24,-2,24,-2,24,-2,2
     4,-2,24,-2,24,-2,24,0
3110 DATA -3,3,-2,3,-2,1,128,0,1,128,-2,192,-2,192
     ,-2,96,-2,96,-2,48,-2,56
3120 DATA -2,28,-2,6,-2,6,-2,3,-2,3,-2,1,128,0,1,1
     28,-2,192,-2,192,-3
3130 DATA -9,24,-2,12,-2,6,-2,3,-2,1,128,-2,192,-2
     ,112,-2,24,-2,14,-2,3,-2,1
3140 DATA 128,-2,192,-2,96,-2,48,-2,24,-9
3150 DATA -18,96,-2,60,-2,7,-2,1,192,-2,126,-2,3,1
     28,-2,224,-2,60,-2,6,-18
3160 DATA -30,255,255,255,-30
3170 DATA -20,6,-2,60,-2,224,0,3,128,0,126,0,1,192
     ,0,7,-2,60,-2,96,-20
```

```
3180 DATA -11,24,-2,48,-2,96,-2,192,Ø,1,128,Ø,3,-2
     ,14,-2,24,-2,112,-2,192,Ø,1
3190 DATA 128,Ø,3,-2,6,-2,12,-2,24,-11
3200 DATA -5,192,-2,192,Ø,1,128,Ø,1,128,Ø,3,-2,3,-
     2,6,-2,6,-2,12,-2
3210 DATA 24,-2,48,-2,96,-2,96,-2,192,-2,192,Ø,1,1
     28,Ø,1,128,Ø,3,-2,3,-5
3220 :
3230 REM CHARACTER DATA
3240 DATA Ø,255,2Ø4,Ø,51,255,2Ø4,Ø,51
3250 DATA 1,255,25Ø,238,235,238,25Ø,255,85
3260 DATA 2,253,189,237,173,237,189,253,85
3270 DATA 3,Ø,6Ø,255,255,255,255,6Ø,Ø
3280 DATA 4,85,255,255,168,255,255,69,69
3290 DATA 5,255,2Ø4,Ø,51,255,2Ø4,Ø,51
3300 DATA 6,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
3310 DATA 1Ø,Ø,9Ø,9Ø,6Ø,6Ø,6Ø,Ø,Ø
3320 DATA 11,255,255,255,255,255,255,255,255
3330 DATA 12,85,Ø,6Ø,6Ø,6Ø,6Ø,Ø,85
3340 DATA 31,255,255,255,255,255,255,255,255
3350 DATA 64,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø
3360 :
3370 REM DIRECTIONAL DATA
3380 DATA Ø,-1,Ø,1,-1,Ø,1,Ø
3390 DATA -4Ø,4Ø,-1,1
```

Program 2. Zuider Zee: Part 2. Machine Language Data to Use with MLX

```
49152 :Ø76,Ø21,192,Ø76,175,192,22Ø
49158 :Ø76,Ø6Ø,193,Ø76,139,193,231
49164 :Ø76,Ø23,2Ø2,Ø76,2Ø5,193,Ø19
4917Ø :Ø76,226,194,Ø32,116,193,Ø87
49176 :141,ØØØ,2Ø5,Ø32,116,193,199
49182 :141,ØØØ,2Ø6,Ø32,116,193,2Ø6
49188 :133,ØØ2,Ø32,116,193,133,133
49194 :Ø1Ø,169,ØØØ,141,137,ØØ3,246
492ØØ :141,179,ØØ2,133,Ø13,169,173
492Ø6 :ØØ1,141,143,ØØ3,141,178,149
49212 :ØØ2,133,ØØ9,162,ØØ3,Ø24,137
49218 :189,243,2Ø2,164,Ø13,121,23Ø
49224 :ØØØ,2Ø6,141,183,ØØ2,2Ø1,Ø37
4923Ø :Ø24,176,Ø72,Ø24,189,247,Ø42
49236 :2Ø2,121,ØØØ,2Ø5,141,182,167
49242 :ØØ2,2Ø1,Ø4Ø,176,Ø58,172,227
49248 :183,ØØ2,185,ØØØ,2Ø7,133,Ø38
49254 :158,185,Ø64,2Ø7,133,159,24Ø
4926Ø :172,182,ØØ2,177,158,2Ø1,232
49266 :Ø64,176,Ø41,197,ØØ2,24Ø,Ø66
```

```
49272 :Ø37,165,ØØ2,Ø32,Ø43,193,Ø8Ø
49278 :238,178,ØØ2,2Ø8,ØØ3,238,225
49284 :179,ØØ2,164,ØØ9,173,182,Ø73
4929Ø :ØØ2,153,ØØØ,2Ø5,173,183,Ø86
49296 :ØØ2,153,ØØØ,2Ø6,23Ø,ØØ9,232
493Ø2 :Ø76,158,192,169,ØØ1,141,119
493Ø8 :137,ØØ3,2Ø2,Ø16,16Ø,23Ø,136
49314 :Ø13,166,ØØ9,228,Ø13,2Ø8,Ø31
4932Ø :15Ø,169,ØØØ,141,143,ØØ3,ØØ6
49326 :Ø96,16Ø,ØØØ,132,Ø18,Ø32,1ØØ
49332 :116,193,164,Ø18,153,18Ø,236
49338 :ØØ2,2ØØ,192,ØØ4,2Ø8,241,ØØ9
49344 :Ø32,116,193,133,ØØ2,Ø32,188
4935Ø :116,193,133,Ø1Ø,174,183,239
49356 :ØØ2,189,ØØØ,2Ø7,133,253,22Ø
49362 :189,Ø64,2Ø7,133,254,174,2Ø7
49368 :181,ØØ2,189,ØØØ,2Ø7,133,16Ø
49374 :158,189,Ø64,2Ø7,133,159,1Ø8
4938Ø :172,18Ø,ØØ2,165,ØØ2,Ø32,Ø13
49386 :Ø25,193,2ØØ,2Ø4,182,ØØ2,Ø16
49392 :2Ø8,245,174,181,ØØ2,238,ØØ8
49398 :183,ØØ2,189,ØØØ,2Ø7,133,192
494Ø4 :158,189,Ø64,2Ø7,133,159,138
4941Ø :165,ØØ2,172,18Ø,ØØ2,Ø32,Ø43
49416 :Ø43,193,165,ØØ2,172,182,253
49422 :ØØ2,Ø32,Ø43,193,232,236,24Ø
49428 :183,ØØ2,2Ø8,224,Ø96,Ø72,Ø37
49434 :145,253,165,254,Ø72,Ø73,22Ø
4944Ø :Ø88,133,254,165,Ø1Ø,145,Ø59
49446 :253,1Ø4,133,254,1Ø4,145,ØØ7
49452 :158,165,159,Ø72,Ø73,Ø88,247
49458 :133,159,165,Ø1Ø,145,158,Ø52
49464 :1Ø4,133,159,Ø96,169,ØØØ,2Ø5
4947Ø :141,178,ØØ2,141,179,ØØ2,193
49476 :133,158,169,128,133,159,18Ø
49482 :169,ØØ4,133,Ø18,16Ø,ØØØ,Ø46
49488 :162,ØØØ,177,158,2Ø1,Ø11,Ø21
49494 :24Ø,Ø12,2Ø1,Ø31,176,ØØ8,242
495ØØ :238,178,ØØ2,2Ø8,ØØ3,238,191
495Ø6 :179,ØØ2,23Ø,158,2Ø8,ØØ2,1Ø9
49512 :23Ø,159,232,224,24Ø,2Ø8,117
49518 :227,198,Ø18,2Ø8,223,Ø96,Ø56
49524 :Ø32,253,174,Ø32,158,173,17Ø
4953Ø :165,Ø13,24Ø,ØØ3,1Ø4,1Ø4,239
49536 :Ø96,Ø32,247,183,165,Ø21,1Ø4
49542 :2Ø8,246,165,Ø2Ø,Ø96,16Ø,ØØ5
49548 :ØØØ,132,158,132,253,169,216
49554 :2Ø8,133,159,169,16Ø,133,Ø84
4956Ø :254,12Ø,165,ØØ1,Ø41,251,216
```

```
49566 :133,001,177,158,145,253,001
49572 :200,208,249,230,159,230,160
49578 :254,165,159,201,216,208,093
49584 :239,165,001,009,004,133,215
49590 :001,088,173,000,221,041,194
49596 :252,009,001,141,000,221,044
49602 :169,008,141,024,208,169,145
49608 :024,141,022,208,096,173,096
49614 :173,002,205,136,003,176,133
49620 :081,169,000,141,175,002,012
49626 :169,020,141,145,003,173,101
49632 :027,212,041,031,201,022,246
49638 :176,247,170,232,142,169,086
49644 :002,173,027,212,041,063,242
49650 :201,038,176,247,170,232,026
49656 :142,168,002,172,169,002,135
49662 :185,000,207,024,109,168,179
49668 :002,133,158,185,064,207,241
49674 :105,000,133,159,160,000,055
49680 :177,158,201,064,208,016,072
49686 :169,000,141,174,002,032,028
49692 :052,194,173,174,002,240,095
49698 :003,032,121,194,206,145,223
49704 :003,240,008,174,173,002,128
49710 :236,136,003,144,172,096,065
49716 :162,003,142,171,002,174,194
49722 :171,002,173,168,002,024,086
49728 :125,247,202,072,173,169,028
49734 :002,024,125,243,202,168,066
49740 :104,024,121,000,207,133,153
49746 :253,185,064,207,105,000,128
49752 :133,254,160,000,177,253,041
49758 :201,033,144,017,201,048,226
49764 :176,013,169,001,141,174,006
49770 :002,173,175,002,240,008,194
49776 :032,100,195,206,171,002,050
49782 :016,193,096,173,173,002,003
49788 :205,136,003,176,096,160,132
49794 :000,200,192,016,240,089,099
49800 :185,000,204,208,246,152,107
49806 :072,169,001,153,000,204,229
49812 :173,168,002,153,032,204,112
49818 :173,169,002,153,064,204,151
49824 :169,006,133,010,172,169,051
49830 :002,185,000,207,024,109,181
49836 :168,002,133,158,185,064,114
49842 :207,105,000,133,159,104,118
49848 :024,105,064,072,160,000,097
49854 :032,043,193,238,173,002,103
```

```
49860 :104,132,159,010,038,159,030
49866 :200,192,003,208,248,133,162
49872 :158,165,159,024,105,160,211
49878 :133,159,169,000,160,007,074
49884 :145,158,136,016,251,096,254
49890 :173,173,002,240,119,169,078
49896 :000,141,140,003,173,136,057
49902 :003,074,141,145,003,169,005
49908 :001,141,175,002,173,027,251
49914 :212,041,015,201,015,240,206
49920 :247,170,232,142,177,002,202
49926 :189,000,204,240,083,188,142
49932 :064,204,140,169,002,189,012
49938 :032,204,141,168,002,024,077
49944 :121,000,207,133,158,185,060
49950 :064,207,105,000,133,159,186
49956 :160,000,177,158,201,064,028
49962 :240,019,169,000,141,174,017
49968 :002,032,052,194,173,174,163
49974 :002,240,006,173,140,003,106
49980 :240,032,096,174,177,002,013
49986 :169,000,157,000,204,188,016
49992 :064,204,185,000,207,133,097
49998 :158,185,064,207,133,159,216
50004 :169,064,188,032,204,145,118
50010 :158,206,173,002,206,145,212
50016 :003,208,149,096,173,027,240
50022 :212,205,144,003,144,001,043
50028 :096,120,165,001,041,254,017
50034 :133,001,173,177,002,024,112
50040 :105,064,160,000,132,254,067
50046 :010,038,254,200,192,003,055
50052 :208,248,133,253,165,254,113
50058 :024,105,160,133,254,174,220
50064 :171,002,224,002,208,018,001
50070 :160,255,200,192,008,240,181
50076 :088,177,253,208,247,169,018
50082 :255,145,253,076,245,195,051
50088 :224,003,208,016,160,008,019
50094 :136,048,068,177,253,208,040
50100 :249,169,255,145,253,076,047
50106 :245,195,160,007,224,001,250
50112 :208,025,177,253,162,000,249
50118 :232,224,008,240,010,010,154
50124 :176,248,189,064,203,017,077
50130 :253,145,253,136,016,234,223
50136 :076,245,195,224,000,208,140
50142 :022,177,253,162,000,232,044
50148 :224,008,240,010,074,176,192
```

```
50154 :248,189,056,203,017,253,176
50160 :145,253,136,016,234,160,160
50166 :007,177,253,201,255,240,099
50172 :008,165,001,009,001,133,057
50178 :001,088,096,136,016,239,066
50184 :165,001,009,001,133,001,062
50190 :088,169,000,141,079,003,238
50196 :174,177,002,169,000,157,187
50202 :000,204,206,173,002,169,012
50208 :003,133,018,166,018,173,031
50214 :168,002,024,125,247,202,038
50220 :141,132,003,072,173,169,222
50226 :002,024,125,243,202,141,019
50232 :133,003,168,104,024,121,097
50238 :000,207,133,158,185,064,041
50244 :207,105,000,133,159,173,077
50250 :168,002,056,253,247,202,234
50256 :141,134,003,072,173,169,004
50262 :002,056,253,243,202,141,215
50268 :135,003,168,104,024,121,135
50274 :000,207,133,253,185,064,172
50280 :207,105,000,133,254,160,195
50286 :000,177,158,141,141,003,218
50292 :201,033,144,013,201,048,244
50298 :176,009,177,253,141,142,252
50304 :003,201,048,144,058,198,012
50310 :018,016,154,174,171,002,157
50316 :173,169,002,024,125,243,108
50322 :202,168,185,000,207,133,017
50328 :253,185,064,207,133,254,224
50334 :173,168,002,024,125,247,129
50340 :202,168,177,253,072,172,184
50346 :169,002,185,000,207,133,098
50352 :253,185,064,207,133,254,248
50358 :172,168,002,104,145,253,002
50364 :076,029,197,172,169,002,065
50370 :185,000,207,133,253,185,133
50376 :064,207,133,254,172,168,174
50382 :002,169,160,145,253,169,080
50388 :011,133,002,169,006,133,154
50394 :010,173,132,003,141,000,165
50400 :205,173,133,003,141,000,111
50406 :206,032,043,192,173,178,030
50412 :002,141,132,003,173,179,098
50418 :002,141,133,003,173,137,063
50424 :003,141,138,003,173,134,072
50430 :003,141,000,205,173,135,143
50436 :003,141,000,206,032,043,173
50442 :192,173,178,002,141,134,062
```

```
5Ø448 :ØØ3,173,179,ØØ2,141,135,137
5Ø454 :ØØ3,173,137,ØØ3,141,139,1Ø6
5Ø46Ø :ØØ3,169,ØØ1,141,14Ø,ØØ3,229
5Ø466 :141,Ø79,ØØ3,Ø96,162,ØØ9,Ø12
5Ø472 :189,16Ø,2Ø4,2Ø8,Ø36,188,ØØ1
5Ø478 :128,2Ø4,185,ØØØ,2Ø7,133,135
5Ø484 :251,185,Ø64,2Ø7,133,252,12Ø
5Ø49Ø :188,Ø96,2Ø4,177,251,2Ø1,151
5Ø496 :ØØ4,24Ø,Ø14,169,Ø1Ø,145,134
5Ø5Ø2 :251,165,252,Ø73,Ø88,133,ØØ8
5Ø5Ø8 :252,169,ØØ1,145,251,2Ø2,Ø72
5Ø514 :Ø16,212,Ø96,12Ø,173,Ø66,253
5Ø52Ø :ØØ3,2Ø8,ØØ3,Ø76,Ø49,234,149
5Ø526 :Ø32,Ø38,197,2Ø6,Ø71,ØØ3,129
5Ø532 :2Ø8,Ø23,169,Ø1Ø,141,Ø71,21Ø
5Ø538 :ØØ3,238,Ø72,ØØ3,173,Ø72,155
5Ø544 :ØØ3,Ø41,ØØ3,141,Ø72,ØØ3,119
5Ø55Ø :168,185,Ø52,2Ø3,141,Ø8Ø,179
5Ø556 :16Ø,173,143,ØØ3,2Ø8,Ø41,Ø84
5Ø562 :162,ØØ9,188,128,2Ø4,185,238
5Ø568 :ØØØ,2Ø7,133,251,185,Ø64,2Ø8
5Ø574 :2Ø7,133,252,188,Ø96,2Ø4,198
5Ø58Ø :177,251,2Ø8,Ø16,189,16Ø,125
5Ø586 :2Ø4,ØØ8,169,ØØ4,Ø4Ø,24Ø,Ø51
5Ø592 :ØØ2,169,Ø12,188,Ø96,2Ø4,Ø63
5Ø598 :145,251,2Ø2,Ø16,217,169,142
5Ø6Ø4 :ØØØ,141,Ø67,ØØ3,173,ØØØ,Ø44
5Ø61Ø :22Ø,2Ø1,127,2Ø8,ØØ8,169,Ø87
5Ø616 :ØØØ,141,Ø78,ØØ3,Ø76,Ø92,Ø62
5Ø622 :199,141,Ø68,ØØ3,Ø41,Ø16,146
5Ø628 :2Ø8,ØØ7,169,ØØ1,141,Ø67,Ø21
5Ø634 :ØØ3,2Ø8,ØØ5,169,ØØØ,141,216
5Ø64Ø :Ø78,ØØ3,162,ØØØ,169,ØØ1,1Ø9
5Ø646 :Ø44,Ø68,ØØ3,24Ø,ØØ9,Ø1Ø,Ø76
5Ø652 :232,224,ØØ4,2Ø8,245,Ø76,185
5Ø658 :228,198,138,ØØ9,Ø32,141,2Ø4
5Ø664 :255,131,142,167,ØØ2,142,Ø47
5Ø67Ø :17Ø,ØØ2,173,Ø67,ØØ3,24Ø,125
5Ø676 :ØØ8,173,167,ØØ2,Ø73,ØØ1,156
5Ø682 :141,167,ØØ2,Ø32,149,199,172
5Ø688 :16Ø,ØØ3,173,Ø14,2Ø8,Ø56,1Ø2
5Ø694 :249,Ø36,2Ø3,141,Ø64,ØØ3,19Ø
5Ø7ØØ :144,ØØ8,173,Ø16,2Ø8,Ø41,Ø9Ø
5Ø7Ø6 :128,2Ø8,ØØ1,Ø24,173,Ø64,1Ø4
5Ø712 :ØØ3,1Ø6,Ø74,Ø74,141,Ø64,23Ø
5Ø718 :ØØ3,173,Ø15,2Ø8,Ø56,249,222
5Ø724 :Ø4Ø,2Ø3,Ø74,Ø74,Ø74,141,13Ø
5Ø73Ø :Ø65,ØØ3,Ø32,199,199,2Ø8,236
5Ø736 :ØØ6,136,Ø16,2Ø6,Ø76,Ø92,Ø68
```

```
50742 :199,169,001,077,167,002,157
50748 :141,167,002,032,149,199,238
50754 :173,014,208,056,233,012,250
50760 :072,144,009,173,016,208,182
50766 :041,128,024,240,001,056,056
50772 :104,106,074,074,141,064,135
50778 :003,072,173,015,208,056,105
50784 :233,040,074,074,074,141,220
50790 :065,003,168,185,000,207,218
50796 :133,251,185,064,207,133,057
50802 :252,104,168,169,064,145,248
50808 :251,173,067,003,240,099,185
50814 :173,079,003,240,094,173,120
50820 :143,003,208,089,173,083,063
50826 :003,208,084,174,170,002,011
50832 :173,065,003,056,253,251,177
50838 :202,201,002,144,070,201,202
50844 :023,176,066,168,173,064,058
50850 :003,056,253,255,202,201,108
50856 :001,144,054,201,039,176,015
50862 :050,024,121,000,207,133,197
50868 :251,185,064,207,105,000,224
50874 :133,252,160,000,177,251,135
50880 :240,025,201,011,240,016,157
50886 :201,081,176,023,201,031,143
50892 :144,019,201,065,176,009,050
50898 :201,048,176,011,169,001,048
50904 :141,083,003,169,064,160,068
50910 :000,145,251,076,092,199,217
50916 :173,067,003,240,115,173,231
50922 :078,003,208,110,173,069,107
50928 :003,240,059,172,076,003,025
50934 :208,100,141,076,003,173,179
50940 :014,208,056,233,012,072,079
50946 :144,004,173,016,208,010,045
50952 :104,106,074,074,141,080,075
50958 :003,173,015,208,056,233,190
50964 :040,074,074,074,141,081,248
50970 :003,173,170,002,141,082,085
50976 :003,169,000,141,069,003,161
50982 :169,001,141,078,003,076,250
50988 :092,199,173,077,003,208,028
50994 :041,173,030,208,044,017,051
51000 :208,016,251,173,030,208,174
51006 :010,144,027,074,074,074,209
51012 :074,160,003,074,176,007,050
51018 :200,192,007,208,248,240,145
51024 :011,140,069,003,140,077,008
51030 :003,169,001,141,078,003,225
```

```
51036 :173,069,003,240,049,170,028
51042 :188,044,203,173,016,208,162
51048 :010,144,006,152,013,016,189
51054 :208,208,006,152,073,255,244
51060 :045,016,208,141,016,208,238
51066 :173,255,131,024,105,014,056
51072 :157,248,131,138,010,170,214
51078 :173,014,208,157,000,208,126
51084 :173,015,208,157,001,208,134
51090 :076,225,199,174,167,002,221
51096 :208,003,206,015,208,224,248
51102 :001,208,003,238,015,208,063
51108 :224,002,208,013,206,014,063
51114 :208,016,008,173,016,208,031
51120 :041,127,141,016,208,224,165
51126 :003,208,013,238,014,208,098
51132 :208,008,173,016,208,009,042
51138 :128,141,016,208,096,174,189
51144 :065,003,189,000,207,024,176
51150 :109,064,003,133,251,189,187
51156 :064,207,105,000,133,252,205
51162 :162,000,161,251,201,064,033
51168 :096,173,052,003,240,003,023
51174 :076,028,201,173,061,003,004
51180 :024,105,001,041,007,141,043
51186 :061,003,024,105,050,141,114
51192 :249,131,206,075,003,208,096
51198 :024,173,073,003,073,001,089
51204 :141,073,003,240,004,169,122
51210 :128,208,002,169,129,141,019
51216 :004,212,169,002,141,075,107
51222 :003,169,000,141,053,003,135
51228 :173,016,208,024,041,001,235
51234 :240,001,056,173,000,208,200
51240 :106,205,054,003,240,047,183
51246 :176,024,169,008,013,053,233
51252 :003,141,053,003,238,000,234
51258 :208,208,032,169,001,013,177
51264 :016,208,141,016,208,076,217
51270 :093,200,169,004,013,053,090
51276 :003,141,053,003,206,000,226
51282 :208,016,008,169,254,045,014
51288 :016,208,141,016,208,173,082
51294 :001,208,074,205,055,003,128
51300 :240,027,176,014,169,002,216
51306 :013,053,003,141,053,003,116
51312 :238,001,208,076,129,200,196
51318 :169,001,013,053,003,141,242
51324 :053,003,206,001,208,173,000
```

```
51330 :053,003,208,060,169,001,112
51336 :141,052,003,174,056,003,053
51342 :224,010,176,045,189,160,178
51348 :204,208,040,169,001,157,159
51354 :160,204,188,128,204,185,199
51360 :000,207,133,251,185,064,232
51366 :207,133,252,188,096,204,222
51372 :169,011,145,251,165,252,141
51378 :073,088,133,252,169,014,139
51384 :145,251,169,030,141,070,222
51390 :003,076,049,234,024,105,169
51396 :035,141,248,131,174,053,210
51402 :003,173,001,208,024,125,224
51408 :025,203,141,003,208,173,193
51414 :000,208,024,125,003,203,009
51420 :141,002,208,173,016,208,200
51426 :041,001,125,014,203,041,139
51432 :001,010,141,068,003,173,116
51438 :016,208,041,253,013,068,069
51444 :003,141,016,208,173,016,033
51450 :208,041,001,208,018,173,131
51456 :000,208,201,005,176,011,089
51462 :173,021,208,041,252,141,074
51468 :021,208,076,025,201,173,204
51474 :021,208,009,003,141,021,165
51480 :208,076,049,234,169,128,120
51486 :141,004,212,206,070,003,154
51492 :240,000,169,255,141,057,130
51498 :003,141,058,003,141,056,188
51504 :003,173,016,208,024,041,001
51510 :001,240,001,056,173,000,013
51516 :208,106,074,074,141,059,210
51522 :003,173,001,208,074,074,087
51528 :074,141,060,003,162,009,009
51534 :189,160,204,208,108,188,111
51540 :128,204,140,055,003,185,031
51546 :000,207,133,251,185,064,162
51552 :207,133,252,188,096,204,152
51558 :140,054,003,177,251,201,160
51564 :004,240,080,173,059,003,155
51570 :056,237,054,003,016,005,229
51576 :073,255,024,105,001,032,098
51582 :252,201,165,252,072,165,209
51588 :251,072,173,060,003,056,235
51594 :237,055,003,016,005,073,015
51600 :255,024,105,001,032,252,045
51606 :201,104,024,101,251,133,196
51612 :251,104,101,252,133,252,225
51618 :205,058,003,144,011,208,023
```

```
51624 :Ø22,165,251,2Ø5,Ø57,ØØ3,1Ø3
5163Ø :144,ØØ2,2Ø8,Ø13,165,251,189
51636 :141,Ø57,ØØ3,165,252,141,171
51642 :Ø58,ØØ3,142,Ø56,ØØ3,2Ø2,138
51648 :Ø16,14Ø,174,Ø56,ØØ3,224,Ø37
51654 :255,24Ø,Ø25,189,Ø96,2Ø4,183
5166Ø :Ø1Ø,Ø1Ø,Ø24,1Ø5,ØØ8,141,246
51666 :Ø54,ØØ3,189,128,2Ø4,Ø1Ø,Ø3Ø
51672 :Ø1Ø,Ø24,1Ø5,Ø22,141,Ø55,Ø61
51678 :ØØ3,Ø76,236,2Ø1,169,ØØ1,14Ø
51684 :141,Ø54,ØØ3,169,Ø72,141,Ø4Ø
5169Ø :Ø55,ØØ3,169,ØØØ,141,Ø52,142
51696 :ØØ3,141,Ø73,ØØ3,169,ØØ2,119
517Ø2 :141,Ø75,ØØ3,Ø76,Ø49,234,Ø56
517Ø8 :134,251,162,ØØØ,133,252,16Ø
51714 :168,24Ø,Ø11,169,ØØØ,Ø24,1Ø2
5172Ø :1Ø1,252,144,ØØ1,232,136,1Ø6
51726 :2Ø8,247,134,252,166,251,248
51732 :133,251,Ø96,169,143,141,185
51738 :Ø24,212,169,255,141,Ø14,Ø73
51744 :212,141,Ø15,212,169,24Ø,253
5175Ø :141,Ø2Ø,212,169,129,141,Ø82
51756 :Ø18,212,169,Ø5Ø,141,ØØØ,122
51762 :212,169,Ø17,141,ØØ5,212,Ø38
51768 :169,241,141,ØØ6,212,169,226
51774 :ØØØ,141,173,ØØ2,141,Ø76,Ø83
5178Ø :ØØ3,141,Ø77,ØØ3,141,Ø61,238
51786 :ØØ3,141,Ø35,2Ø8,169,ØØ6,124
51792 :141,Ø32,2Ø8,169,ØØ1,141,ØØ4
51798 :Ø46,2Ø8,141,Ø39,2Ø8,141,1Ø1
518Ø4 :Ø4Ø,2Ø8,141,Ø71,ØØ3,141,184
5181Ø :Ø79,ØØ3,169,166,141,Ø14,158
51816 :2Ø8,169,132,141,Ø15,2Ø8,2Ø9
51822 :169,ØØØ,141,ØØØ,2Ø8,169,Ø29
51828 :144,141,ØØ1,2Ø8,169,Ø32,Ø43
51834 :141,255,131,169,ØØ5,141,196
5184Ø :Ø34,2Ø8,141,ØØ1,212,169,125
51846 :128,141,Ø64,2Ø7,162,ØØØ,Ø68
51852 :142,ØØØ,2Ø7,189,ØØØ,2Ø7,117
51858 :Ø24,1Ø5,Ø4Ø,157,ØØ1,2Ø7,168
51864 :157,Ø26,2Ø7,189,Ø64,2Ø7,234
5187Ø :1Ø5,ØØØ,157,Ø65,2Ø7,157,Ø81
51876 :Ø9Ø,2Ø7,232,224,Ø24,2Ø8,125
51882 :228,169,ØØØ,16Ø,ØØØ,153,112
51888 :ØØØ,2Ø4,174,ØØØ,ØØ4,2Ø8,254
51894 :Ø21,153,ØØØ,136,153,ØØØ,133
519ØØ :137,153,ØØØ,138,153,ØØØ,ØØ1
519Ø6 :139,153,ØØØ,14Ø,153,ØØØ,Ø11
51912 :141,153,ØØØ,142,2ØØ,2Ø8,Ø2Ø
```

```
51918 :224,169,255,160,127,153,014
51924 :000,161,136,016,250,120,127
51930 :169,085,141,020,003,169,037
51936 :197,141,021,003,169,000,243
51942 :141,066,003,141,069,003,141
51948 :169,001,141,052,003,088,178
51954 :096,000,000,255,001,001,083
51960 :255,000,000,255,001,000,247
51966 :000,000,000,255,001,000,254
51972 :000,000,000,253,254,253,252
51978 :000,002,001,002,000,000,015
51984 :000,000,255,255,255,000,013
51990 :000,000,000,000,254,002,022
51996 :000,001,254,002,000,000,029
52002 :254,002,010,014,012,012,082
52008 :040,040,039,041,001,002,203
52014 :004,008,016,032,064,128,042
52020 :102,060,102,195,001,003,003
52026 :007,015,031,063,127,255,044
52032 :128,192,224,240,248,252,068
52038 :254,255,080,255,084,251,225
```

Appendix A

Beginner's Guide to Typing In Programs

A

A Beginner's Guide to Typing In Programs

What Is a Program?

A computer cannot perform any task by itself. Like a car without gas, a computer has *potential*, but without a program, it isn't going anywhere. Most of the programs in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into all Commodore 64s.

BASIC Programs

Computers can be picky. Unlike the English language, which is full of ambiguities, BASIC usually has only one right way of stating something. Every letter, character, or number is significant. A common mistake is substituting a letter such as O for the numeral 0, a lowercase l for the numeral 1, or an uppercase B for the numeral 8. Also, you must enter all punctuation such as colons and commas just as they appear in the book. Spacing can be important. To be safe, type in the listings *exactly* as they appear.

Braces and Special Characters

The exception to this typing rule is when you see the braces, such as {DOWN}. Anything within a set of braces is a special character or characters that cannot easily be listed on a printer. When you come across such a special statement, refer to "How To Type In Programs."

About DATA Statements

Some programs contain a section or sections of DATA statements. These lines provide information needed by the program. Some DATA statements contain actual programs (called machine language); others contain graphics codes. These lines are especially sensitive to errors.

If a single number in any one DATA statement is mistyped, your machine could lock up, or crash. The keyboard and STOP key may seem dead, and the screen may go blank. Don't panic—no damage is done. To regain control, you have to turn off your computer, then turn it back on. This will erase whatever program was in memory, *so always SAVE a copy of your program before you RUN it.* If your computer crashes, you can LOAD the program and look for your mistake.

Sometimes a mistyped DATA statement will cause an error message when the program is RUN. The error message may refer to the program line that READs the data. *The error is still in the DATA statements, though.*

Get to Know Your Machine

You should familiarize yourself with your computer before attempting to type in a program. Learn the statements you use to store and retrieve programs from tape or disk. You'll want to save a copy of your program, so that you won't have to type it in every time you want to use it. Learn to use your machine's editing functions. How do you change a line if you made a mistake? You can always retype the line, but you at least need to know how to backspace. Do you know how to enter inverse video, lowercase, and control characters? It's all explained in your computer's manuals.

A Quick Review

1) Type in the program a line at a time, in order. Press RETURN at the end of each line. Use backspace or the back arrow to correct mistakes.
2) Check the line you've typed against the line in the book. You can check the entire program again if you get an error when you RUN the program.
3) Make sure you've entered statements in braces as the appropriate control key (see "How To Type In Programs" elsewhere in the book).

Appendix B

How to Type In Programs

B

How to Type In Programs

Many of the programs which are listed in this book contain special control characters (cursor control, color keys, reverse video, etc.). To make it easy to know exactly what to type when entering one of these programs into your computer, we have established the following listing conventions.

Generally, any Commodore 64 program listings will contain words in braces which spell out any special characters: {DOWN} would mean to press the cursor down key. {5 SPACES} would mean to press the space bar five times.

To indicate that a key should be *shifted* (hold down the SHIFT key while pressing the other key), the key would be underlined in our listings. For example, $\underline{S}$ would mean to type the S key while holding the shift key. This would appear on your screen as a heart symbol. If you find an underlined key enclosed in braces (e.g., {10 $\underline{N}$}), you should type the key as many times as indicated (in our example, you would enter ten shifted N's).

If a key is enclosed in special brackets, [ ] , you should hold down the *Commodore key* while pressing the key inside the special brackets. (The Commodore key is the key in the lower-left corner of the keyboard.) Again, if the key is preceded by a number, you should press the key as many times as necessary.

Rarely, you'll see a solitary letter of the alphabet enclosed in braces. These characters can be entered on the Commodore 64 by holding down the CTRL key while typing the letter in the braces. For example, {A} would indicate that you should press CTRL-A.

About the *quote mode:* you know that you can move the cursor around the screen with the CRSR keys. Sometimes a programmer will want to move the cursor under program control. That's why you see all the {LEFT}'s, {HOME}'s, and {BLU}'s in our programs. The only way the computer can tell the difference between direct and programmed cursor control is the quote mode.

Once you press the quote (the double quote, SHIFT-2), you are in the quote mode. If you type something and then try to change it by moving the cursor left, you'll only get a bunch of reverse-video lines. These are the symbols for cursor left. The only editing key that isn't programmable is the DEL key; you can still use DEL to back up and edit the line. Once you type another quote, you are out of quote mode.

You also go into quote mode when you INSerT spaces into a line. In any case, the easiest way to get out of quote mode is to just press RETURN. You'll then be out of quote mode and you can cursor up to the mistyped line and fix it.

Use the following table when entering cursor and color control keys:

When You Read:	Press:	See:	When You Read:	Press:	See:
{CLEAR}	SHIFT CLR/HOME	♥	{GRN}	CTRL 6	↑
{HOME}	CLR/HOME	S	{BLU}	CTRL 7	←
{UP}	SHIFT ↑CRSR↓	●	{YEL}	CTRL 8	π
{DOWN}	↑CRSR↓	Q	{F1}	f1	▔
{LEFT}	SHIFT ←CRSR→	|	{F2}	f2	╮
{RIGHT}	←CRSR→	]	{F3}	f3	▁
{RVS}	CTRL 9	R	{F4}	f4	╰
{OFF}	CTRL 0	▁	{F5}	f5	▏
{BLK}	CTRL 1	■	{F6}	f6	╯
{WHT}	CTRL 2	E	{F7}	f7	▕
{RED}	CTRL 3	£	{F8}	f8	■
{CYN}	CTRL 4	◣	←	←	←
{PUR}	CTRL 5	▒	↑	SHIFT ↑	π

Appendix C

Maze Generator

Maze Generator

Charles Bond Translated to machine language by Gary E. Marsa and for the 64 by Gregg Peele.

This program can be the basis for many excellent games.

Here's a remarkably short algorithm which produces random mazes on your TV screen.

To understand how it works, refer to the flowchart and Program 1. The following explanation should clarify the details.

The Background Field

The algorithm operates on a background field which must be generated on the screen prior to line number 210 in Program 1. The field must consist of an odd number of horizontal rows, each containing an odd number of cells: a rectangular array. It's convenient to think of the field as a two-dimensional array with the upper-left corner having coordinates X = 0 and Y = 0, where X is the horizontal direction and Y is vertical. No coordinates are used to identify absolute locations by the program, but the concept is useful in configuring the field.

Given that the upper-left cell of the field has coordinates 0,0, then the terminal coordinates both horizontally and vertically must be even numbers. In addition, the background field must be surrounded on all sides by memory cells whose contents are different from the number used to identify the field. That is, if the field consists of reversed (or inverse video) spaces, then the number corresponding to that character must not be visually adjacent to the field.

This could happen inadvertently if the screen RAM and system ROM have contiguous addresses. A sufficient precaution is to avoid covering the entire screen with field. Leave at least one space at the beginning or end of each line and, in general, leave the uppermost and lowermost lines on the screen blank.

The Maze Generator

The creation of the maze begins by placing a special marker in a suitable starting square. The program here always begins at the square just inside the upper-left cell of the previously drawn field. (Note that with our coordinate scheme this would be cell 1,1.) Any cell with odd-numbered coordinates would work, however, as long as it is internal to the field.

Next, a random direction is chosen by invoking the random number generator in your machine and producing an integer from 0 to 3. This integer, with the aid of a short table, determines a direction and a corresponding cell just two steps away from the current cell. This new cell is examined (PEEKed) to see if it is part of the field. If it is, the direction integer is put there as a marker, and the barrier between it and the current cell is erased.

In addition, the pointer to the current cell is moved to point to the new one. This process is repeated until the new cell fails the test; that is, it is not a field cell. When this happens, the direction vector is rotated 90 degrees and the test is repeated. Thus, the path carved out of the field will continue until a dead end is reached.

A dead end, incidentally, could occur in as few as five steps. When it does occur, we can make use of the markers which were dropped along the way Hansel and Gretel style. These can be checked to determine which direction we came from, so that we can back up and look for untrodden paths. So long as none can be found, the program will back up, one step at a time, erasing the markers as it goes. When a new direction can be taken, the pointer is set off in that direction, and the process continues as before.

Ultimately, the pointer will return to the start, a condition which is detected by the recovery of the special starting (now "ending") marker. This cell is then blanked and the program is done, leaving the pointer as it was at the start.

The Program

The direction table set up in lines 100 and 110 converts an integer to an address offset. In this case (40-column screen), we wish to step two cells to the right, up, left, or down.

Line 120 contains the variable SC, which is the memory address of the start of screen RAM. Lines 130-160 establish the background field on the screen.

The rest of the program draws the maze, as previously explained. Line 310 is simply a convenient stopping point which prevents the screen from scrolling.

It may not be immediately obvious that this algorithm always produces a maze with only one nontrivial path between any two points, or that the maze will always be completely filled, but this can be proved. While the proofs will not be provided here, math buffs may find it interesting that for a maze of any size there will be exactly:

$$\frac{(H-1)(V-1)}{2} - 1 \quad \textbf{empty cells in the completed maze,}$$

where H is the number of cells in each field row and V is the number of rows.

An interesting feature of this algorithm is that it works equally well in certain types of nonrectangular fields. U-shaped fields or fields with holes in them are quite suitable—as long as certain restrictions are observed. Just make sure that the coordinates of the upper-left and lower-right cells of any cut-out area are pairs of odd numbers. Also, if there is a single row of field cells between any cut-out areas and the outside of the original field, it may be removed.

Machine Language Mazes

Program 2 is a machine language translation of Program 1. It is in the form of a BASIC loader. It can be inserted into any BASIC program just as Program 1.

Program 3 is the assembly listing of the machine language routine found in Program 2.

The Mouse

The subroutine on lines 1000 to 1020 of Program 1 produces an artificial mouse which roams the maze endlessly. The mouse adheres to a "left-hand rule" when a choice of directions is possible. That is, when it is confronted with a branch-point, it will move off to the left, if possible. Otherwise, it will go forward. When no choice is available, it will turn around. These lines are unnecessary for the creation of the maze and may be deleted. Programs 2 and 3 do not contain the mouse.

Program 1. BASIC Maze Generator

```
100 DIMA(3)
110 A(0)=2:A(1)=-80:A(2)=-2:A(3)=80
120 WL=160:HL=32:SC=1024:A=SC+81
```

```
130 PRINT"{CLR}"
140 FORI=1TO23
150 PRINT"{RVS}{WHT}{39 SPACES}"
160 NEXTI
210 POKEA,4
220 J=INT(RND(1)*4):X=J
230 B=A+A(J):IFPEEK(B)=WLTHENPOKEB,J:POKEA+A(J)/2,
    HL:A=B:GOTO220
240 J=(J+1)*-(J<3):IFJ<>XTHEN230
250 J=PEEK(A):POKEA,HL:IFJ<4THENA=A-A(J):GOTO220
310 GETC$:IFC$=""THEN310
1000 POKEA,81:J=2
1010 B=A+A(J)/2:IFPEEK(B)=HLTHENPOKEB,81:POKEA,HL:
    A=B:J=(J+2)+4*(J>1)
1020 J=(J-1)-4*(J=0):GOTO1010
```

Program 2. Machine Language Maze Generator

```
10 I=49152:IF PEEK(I+2)=216THENSYS49160:END
20 READ A:IF A=256 THENSYS49160:END
30 POKE I,A:I=I+1:GOTO 20
49152 DATA 1,0,216,255,255,255,40
49160 DATA 0,169,81,133,251,169,40
49168 DATA 133,253,169,4,133,252,133
49176 DATA 254,169,147,32,210,255,162
49184 DATA 0,160,0,169,160,145,253
49192 DATA 200,192,39,208,249,24,165
49200 DATA 253,105,40,133,253,144,2
49208 DATA 230,254,232,224,23,208,229
49216 DATA 160,0,169,4,145,251,169
49224 DATA 255,141,15,212,169,128,141
49232 DATA 18,212,173,27,212,41,3
49240 DATA 133,173,170,10,168,24,185
49248 DATA 0,192,101,251,133,170,185
49256 DATA 1,192,101,252,133,171,24
49264 DATA 185,0,192,101,170,133,253
49272 DATA 185,1,192,101,171,133,254
49280 DATA 160,0,177,253,201,160,208
49288 DATA 18,138,145,253,169,32,145
49296 DATA 170,165,253,133,251,165,254
49304 DATA 133,252,76,62,192,232,138
49312 DATA 41,3,197,173,208,189,177
49320 DATA 251,170,169,32,145,251,224
49328 DATA 4,240,26,138,10,168,162
49336 DATA 2,56,165,251,249,0,192
49344 DATA 133,251,165,252,249,1,192
49352 DATA 133,252,202,208,238,76,62
49360 DATA 192,169,1,160,0,153,0
49368 DATA 216,153,0,217,153,0,218
49376 DATA 153,0,219,200,208,241,96,256
```

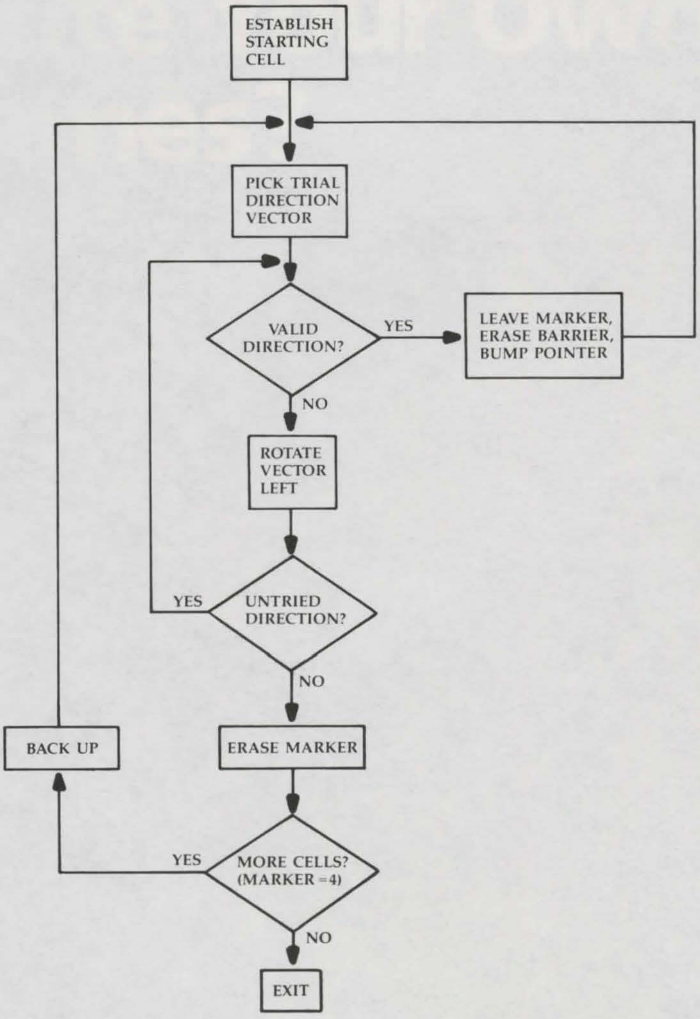
Program 3. Source Listing

```
CØØØ Ø1 ØØ
CØØ2 D8
CØØ3 FF
CØØ4 FF
CØØ5 FF
CØØ6 28
CØØ7 ØØ
CØØ8 A9 51         LDA #$51
CØØA 85 FB         STA $FB
CØØC A9 28         LDA #$28
CØØE 85 FD         STA $FD
CØ1Ø A9 Ø4         LDA #$Ø4
CØ12 85 FC         STA $FC
CØ14 85 FE         STA $FE
CØ16 A9 93         LDA #$93
CØ18 2Ø D2 FF      JSR $FFD2
CØ1B A2 ØØ         LDX #$ØØ
CØ1D AØ ØØ         LDY #$ØØ
CØ1F A9 AØ         LDA #$AØ
CØ21 91 FD         STA ($FD),Y
CØ23 C8            INY
CØ24 CØ 27         CPY #$27
CØ26 DØ F9         BNE $CØ21
CØ28 18            CLC
CØ29 A5 FD         LDA $FD
CØ2B 69 28         ADC #$28
CØ2D 85 FD         STA $FD
CØ2F 9Ø Ø2         BCC $CØ33
CØ31 E6 FE         INC $FE
CØ33 E8            INX
CØ34 EØ 17         CPX #$17
CØ36 DØ E5         BNE $CØ1D
CØ38 AØ ØØ         LDY #$ØØ
CØ3A A9 Ø4         LDA #$Ø4
CØ3C 91 FB         STA ($FB),Y
CØ3E A9 FF         LDA #$FF
CØ4Ø 8D ØF D4      STA $D4ØF
CØ43 A9 8Ø         LDA #$8Ø
CØ45 8D 12 D4      STA $D412
CØ48 AD 1B D4      LDA $D41B
CØ4B 29 Ø3         AND #$Ø3
CØ4D 85 AD         STA $AD
CØ4F AA            TAX
CØ5Ø ØA            ASL
CØ51 A8            TAY
CØ52 18            CLC
CØ53 B9 ØØ CØ      LDA $CØØØ,Y
CØ56 65 FB         ADC $FB
```

```
CØ58 85 AA       STA $AA
CØ5A B9 Ø1 CØ    LDA $CØØ1,Y
CØ5D 65 FC       ADC $FC
CØ5F 85 AB       STA $AB
CØ61 18          CLC
CØ62 B9 ØØ CØ    LDA $CØØØ,Y
CØ65 65 AA       ADC $AA
CØ67 85 FD       STA $FD
CØ69 B9 Ø1 CØ    LDA $CØØ1,Y
CØ6C 65 AB       ADC $AB
CØ6E 85 FE       STA $FE
CØ7Ø AØ ØØ       LDY #$ØØ
CØ72 B1 FD       LDA ($FD),Y
CØ74 C9 AØ       CMP #$AØ
CØ76 DØ 12       BNE $CØ8A
CØ78 8A          TXA
CØ79 91 FD       STA ($FD),Y
CØ7B A9 2Ø       LDA #$2Ø
CØ7D 91 AA       STA ($AA),Y
CØ7F A5 FD       LDA $FD
CØ81 85 FB       STA $FB
CØ83 A5 FE       LDA $FE
CØ85 85 FC       STA $FC
CØ87 4C 3E CØ    JMP $CØ3E
CØ8A E8          INX
CØ8B 8A          TXA
CØ8C 29 Ø3       AND #$Ø3
CØ8E C5 AD       CMP $AD
CØ9Ø DØ BD       BNE $CØ4F
CØ92 B1 FB       LDA ($FB),Y
CØ94 AA          TAX
CØ95 A9 2Ø       LDA #$2Ø
CØ97 91 FB       STA ($FB),Y
CØ99 EØ Ø4       CPX #$Ø4
CØ9B FØ 1A       BEQ $CØB7
CØ9D 8A          TXA
CØ9E ØA          ASL
CØ9F A8          TAY
CØAØ A2 Ø2       LDX #$Ø2
CØA2 38          SEC
CØA3 A5 FB       LDA $FB
CØA5 F9 ØØ CØ    SBC $CØØØ,Y
CØA8 85 FB       STA $FB
CØAA A5 FC       LDA $FC
CØAC F9 Ø1 CØ    SBC $CØØ1,Y
CØAF 85 FC       STA $FC
CØB1 CA          DEX
CØB2 DØ EE       BNE $CØA2
CØB4 4C 3E CØ    JMP $CØ3E
```

```
CØB7 A9 Ø1         LDA #$Ø1
CØB9 AØ ØØ         LDY #$ØØ
CØBB 99 ØØ D8      STA $D8ØØ,Y
CØBE 99 ØØ D9      STA $D9ØØ,Y
CØC1 99 ØØ DA      STA $DAØØ,Y
CØC4 99 ØØ DB      STA $DBØØ,Y
CØC7 C8            INY
CØC8 DØ F1         BNE $CØBB
CØCA 6Ø            RTS
```

Maze Generator Flowchart



Appendix D

Do You Want to Write Your Own Games?

Do You Want to Write Your Own Games?

Orson Scott Card

I remember when videogames first reached my town back in the early seventies. A friend and I dropped a few quarters into a *Pong* machine and had a great time. But all in all, we preferred playing Ping-Pong on a real table.

But then, in a theater lobby, we met *Breakout*, and it changed my life. I became a dedicated videogamer from that time forward.

Because there on a TV screen—not even a color screen, then, just black-and-white with colored plastic strips—the videogame was offering an experience that I couldn't get anywhere else. The speed and the concept both were something entirely new.

Everybody knows where it went from there. Turn *Breakout's* paddle into a spaceship, give the bricks a different shape, and let them march down the screen at you, and you have *Space Invaders*. Turn *Breakout's* paddle into a race car and let it drive over dots instead of bricks, and you have the earliest gobble games. The shoot-outs and gobble games, the climbing games and the simulations—they have all become more sophisticated.

Now, on your own TV at home, you can have the little airplanes of "Richthofen's Revenge" flying around. And you typed the game into your computer yourself.

Getting Behind the Games

If you're like me, however, playing was never really enough. Right from the beginning, I wondered how it was done. I knew nothing about computers then—like many people, I thought computers were for people who were good in math or interested in engineering, and I was definitely neither. But for the first time I *wanted* to have whatever abilities it took to program computers. Because I wanted to make my own games.

I wanted to create a game where I could handle old-time sailing ships through currents and winds to explore different islands and conduct sea battles.

I wanted a game where I could build cities and design traffic flow patterns, create the image of a city's life.

I wanted to have the power of a computer to create whatever world I wanted, and whatever game I wanted to play within that world.

But I knew it would never happen. I wasn't good in math or interested in engineering, and to people like me computers would never be anything but big black boxes.

Unlocking the Little Black Box

The big black boxes have changed, haven't they? You can treat your 64 as a black box, if you want—plug in a game on a ROM cartridge and away you go. But for most games, you still need to type things like LOAD and RUN. And for the games in this book, you need to type in entire programs.

And if it hasn't occurred to you before, it certainly should be plain now. You have the equipment to program all those games you have always wished you could play. Your Commodore 64 can do almost everything the videogames in the arcade can do.

Best of all, though, it can do things that have never been done before. It can display worlds that *you* create, and carry out actions that *you* designed.

And as for the myth that programmers have to be good in math or engineering—you don't believe that anymore, do you? My wife still has to balance the checkbook for me and I can't tell a circuit diagram from a plate of vermicelli, but I have written games that actually work, using BASIC and machine language both. And like those old-time ads ("My Friends Laughed When I Sat Down At The Piano"), I assure you that if I can do it, *anybody* can.

How to Learn How to Program Games

Unfortunately, you won't find a night school class in videogame programming. Colleges and high schools tend to teach programming with a business or mathematical slant. They rarely teach much about the graphics and sound techniques at the heart of game programming.

So the best way to learn programming is to find a friend who's an expert videogame programmer and get him to teach you, step by step, how to solve the problems you run into trying to program your first game. Because you can only learn to program by programming, and having an expert (and patient) friend gets you through the rough places.

The second best way is books.

There are books that teach you BASIC programming for the Commodore 64, reference books that give you valuable information about memory locations and special techniques, books that teach machine language programming for the 6510 that runs your 64, and even a book called *Creating Arcade Games on the Commodore 64*, which sounds like exactly the book you want.

(Before I give you my full list of recommended reading, I'd better explain something. This list will include mostly books published by COMPUTE! Books, which is the publisher of *COMPUTE!'s First Book of Commodore 64 Games*. However, this is *not* merely shameless self-promotion. Wherever I knew of a valuable teaching or reference book by another publisher, I have listed it. But the Commodore 64 is such a new computer that at the time of this writing, most publishers don't have their Commodore 64 books out yet. In fact, many of the books on my list haven't been published yet, either. But because I'm an editor at COMPUTE! Books, I know all about our books that are at the printer or in production or still coming, a chapter at a time, from authors in California, Michigan, Utah, Virginia, Pennsylvania, and New Jersey. Therefore, I can include those books on the list and promise you that they'll help you learn programming. But I can't tell you about forthcoming books by other publishers because, unfortunately, in the world of publishing we don't always tell each other what we have planned. By the time you read this, there may be a hundred other books that can help you; this list will only tell you about the ones I know.)

In the following list, an asterisk (*) marks the books that are useful only if you are planning to use machine language.

BASIC Programming. If you're new at programming, here are some books that can help supplement the manuals published by Commodore.

Camp, David. *Creating Arcade Games on the Commodore 64*. Greensboro, North Carolina: COMPUTE! Books.

Chamberlain, Craig. *All About the Commodore 64*. 2 vols. COMPUTE! Books.

Heilborn, John and Ron Talbott. *Your Commodore 64: A Guide to the Commodore 64 Computer*. Berkeley, California: Osborne/McGraw-Hill.

COMPUTE!'s First Book of Commodore 64.

Commodore 64 Programmer's Reference Guide. West Chester, Pennsylvania: Commodore Business Machines, Inc.

Graphics and Sound Techniques. Once you've mastered the basics of BASIC, you can get into the fascinating techniques of moving shapes and colors on the TV screen and creating sounds from the TV speaker. This is an area where the Commodore 64 is different from every other computer, even its little brother, the VIC-20.

Heilborn, John. *COMPUTE!'s Reference Guide to Commodore 64 Graphics.*

Heilborn, John. *COMPUTE!'s Reference Guide to Commodore 64 Sound.*

COMPUTE!'s *First Book of Commodore 64 Sound and Graphics.*

Reference Books. These are books that give you detailed information about features and key memory locations of the Commodore 64. Many of these features are only usable in machine language, but others are valuable to BASIC programmers as well.

*Heeb, Dan. *The Commodore 64 Tool Kit: Kernal Routines.* COMPUTE! Books.

*Heeb, Dan. *The BASIC Tool Kit: Commodore 64 and VIC-20.* COMPUTE! Books.

Leemon, Sheldon. *Mapping the Commodore 64*. COMPUTE! Books.

Learning Machine Language. These are books that help you learn how to put real speed and complex but smooth animation into your videogames.

Fernandez, Judi N., Donna N. Tabler, and Ruth Ashley. *6502 Assembly Language Programming*. New York: John Wiley and Sons.

Leventhal, Lance A., and Winthrop Saville. *6502 Assembly Language Subroutines*. Osborne/McGraw-Hill.

Mansfield, Richard. *Machine Language for Beginners.* COMPUTE! Books.

Zaks, Rodnay. *Programming the 6502*. Berkeley, California: Sybex.

Index

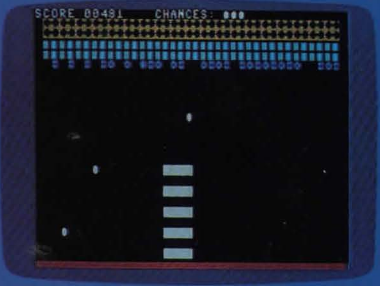
COMPUTE!'s First Book of

Commodore 64 Games

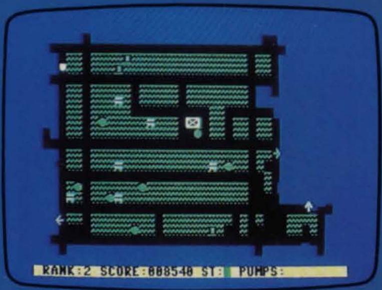
COMPUTE's First Book of Commodore 64 Games includes 19 games complete and ready to type in, so no programming knowledge is necessary.

- Save the Snake in "Snake Escape"
- Mine in "Oil Tycoon"
- Attack the sky skimmer in "The Hawkmen of Dindrin"
- Shoot at the invading spaceships in "Laser Gunner"

Here are 15 of the best games from COMPUTE! Magazine and COMPUTE!'s Gazette, plus four never-before-published games, including the arcade-speed "Richthofen's Revenge," "Spike," and "Zuider Zee."



"Diamond Drop"



"Zuider Zee"

"Richthofen's Revenge"

"Mystery Spell"

ISBN 0-942386-34-5

$12.95