COMPUTE!'s SECOND BOOK OF

VIC GAMES

Twenty-six of the best VIC-20 games from COMPUTE! Publications, all ready to type in and run. Also includes detailed information on creating your own computer games.

A **COMPUTE! Books** Publication

$12.95

COMPUTE!'s SECOND BOOK OF

VIC GAMES

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

The following articles were originally published in *COMPUTE!* magazine, copyright 1984, COMPUTE! Publications, Inc.:
"Demons of Osiris" (January)
"Gotcha!" (February)
"Quatrainment" (February)
"Worm of Bemer" (April)
"Snertle" (May)
"Olympiad" (June)

The following articles were originally published in *COMPUTE!'s Gazette*, copyright 1984, COMPUTE! Publications, Inc.:
"Alpha-Shoot" (January)
"Cave-In for VIC-20" (January)
"Hardhat Climber" (January)
"Checkers" (February)
"Typing Derby" (February)
"CUT-OFF!: All-Machine-Language Game for Commodore 64 and VIC-20" (March)
"Poker" (March)
"Tree Tutor for Tots" (March)
"Trenchfire" (March)
"Mind Boggle" (May)
"The Beginner's Corner: Planning a Game Program" (June)
"Castle Dungeon" (June)
"The Frantic Fisherman" (June)
"Therapy" (June)
"Word Scramble" (June)

Printed in the United States of America

ISBN 0-942386-57-4

10 9 8 7 6 5 4 3 2 1

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is one of the ABC Publishing Companies and is not associated with any manufacturer of personal computers. VIC-20 is a trademark of Commodore Electronics Limited.

Contents

Foreword

How would you like to explore a medieval dungeon or travel in time?

Or maybe you'd rather help your child learn the alphabet or basic math, or just sharpen your reflexes and coordination.

Do you feel like a quiet round of checkers, a friendly hand of poker, a challenging mind twister—or a fast-paced arcade game?

Sound appealing? All you need for all of this, and more, is *COMPUTE!'s Second Book of VIC Games.*

A collection of some of the best games from recent issues of *COMPUTE!* and *COMPUTE!'s Gazette,* the *Second Book of VIC Games* also includes several games and tutorials that have never been published before.

There's something here for everyone. Adventurers can rescue trapped miners in "Cave-In" or escape from the future in "Time Capsule." Parents can type in "Snertle" or "Alpha-Shoot" to make a child's learning fun. Arcade fans can spend hours maneuvering the "Worm of Bemer" through mushroom-packed mazes, and fishing enthusiasts can enjoy a quiet (?) day on the lake in "The Frantic Fisherman."

If you want to create your own games, you'll find articles to help. There are guides to designing your own game scenarios, as well as hints on how to make your ideas come alive on the screen. There are even explanations of sophisticated adventure-game programming techniques.

If you're familiar with other COMPUTE! Books, you know that you can expect clear writing and high-quality programs. Each of the 26 programs has been thoroughly tested, and each is ready to type in and run. Complete listings are included, too, and many of the articles tell you how to modify and customize the games to suit your own needs.

The VIC-20 already has a reputation as a fine games machine. Now, with *COMPUTE!'s Second Book of VIC Games,* it's more exciting than ever.

1
Ideas and Applications

Ideas and Applications

Every good game begins with an idea. But once you get the inspiration, what happens then? The articles in this chapter will help you bring your video visions to life on the monitor screen. For instance, you may want to develop a computer version of some existing game. In "Planning a Game Program," C. Regena shows you how to take a simple game (tic-tac-toe) and turn it into an exciting contest between you and the computer.

On the other hand, you may want to create a wholly new game, perhaps one set in some fantasy world. Gregg Keizer's "World Building" shows you how to start with a germ of an idea and expand it into the full-scale world your game needs.

Once you have the world, you're ready to create the game itself—and Orson Scott Card's "Thinking It Through" gives a step-by-step description of how an idea can be turned into an exciting, playable game.

Finally Gary McGath introduces you to the exciting art of creating text adventures in "Writing Adventure Games." Text adventures open new universes to the game creator, and McGath will get you started.

Planning a Game Program

C. Regena

Even if you're a beginning VIC programmer, it's not hard to plan and create game programs. This article takes a simple game with which everyone is familiar—tic-tac-toe—and shows you how to program it on the unexpanded VIC.

Let's explore a step-by-step procedure for writing a VIC game. To keep it simple yet worthwhile, we'll create "Tic-Tac-Toe." It's easy to understand, and everyone knows the game, but programming it involves graphics, logic, and strategy.

Start out with the graphics. Tic-Tac-Toe is graphically simple, requiring only an X marker, an O marker, and four straight lines. On the VIC, let the X and O markers each take up a pattern measuring four characters by four characters; thus, the basic game grid must be a pattern measuring six by six spaces.

Creating the Grid

The grid lines are made up of solid blocks one character wide (the reverse space). You can either PRINT the grid or use a series of POKEs to place the colored squares on the screen. I chose to use the POKE method. First the screen is cleared and a random color is chosen for the grid (line 38). The random color can be any one of six colors, but not black or white. You couldn't see a white grid, and I didn't like the black grid.

The grid is drawn in lines 39–40. To draw a reverse space on the screen, you need to POKE a screen location with 160, the ASCII value for a SHIFTed space, then POKE the corresponding color location with the number which represents the color you want for that location. The variable C relates the screen memory location to the color memory location, and T is the random color selected by line 38. Line 41 places numbers in the positions to be chosen as plays are made.

The X and O markers are drawn in subroutines at the beginning of the program (lines 2–7). The nine coordinate po-

sitions for the markers to be drawn are READ in as S(I) in lines 28–29. The graphics are now complete.

Next, I programmed the player moves. The squares are numbered, so the player just presses a number from 1 to 9. I like to avoid INPUT if at all possible. In this case, only one key press is necessary, so we can use GET. GET E$ (line 47) gets the key pressed, and line 48 makes sure that the key is one of the numbers from 1 to 9. All other keys are ignored.

Plotting the Move

In line 49, VAL changes E$ to a numerical value and assigns this value to the variable E. P(E) is the value in that position on the grid—3 for an X, 1 for an O, and 0 if there is no marker in that position. If there is already a marker in the position chosen, the player must choose again. If the square is available, the program continues with line 50 and is sent to the subroutine beginning at line 10. This subroutine determines whether to draw an X or an O and places the appropriate character into the grid. The value representing an X or an O is stored in P(E).

Next it's the computer's move. For the beginner's level, simply let the computer randomly choose any one of the available spots (lines 44–45). Since the value of N or X changes between moves and can be either 1 or 3, the relative formula is N=ABS(N−4) (line 27).

Is the Game Over?

After each marker is placed, the computer checks to see if the game is over. First the rows are checked to see if there are three X's or O's in a row (lines 12–16). Next the columns are checked to see if there are three of a kind in a column (lines 17–21). Next, diagonal wins are checked (lines 22–24). If there isn't a win, all spaces are checked. If all spaces are filled, it is a tie game. If there are empty spots, the game continues (lines 25–27).

If there is a winner, the program branches to lines 89–97 to congratulate the winner and play a tune made up of random notes. The program then offers the option to try again and branches appropriately. Line 31 sets variables for playing the music and the prompter beep, and the subroutine in lines 8–9 plays the tones and delays.

The game could be complete now, but it wouldn't be very

challenging because the computer's moves are chosen randomly. No strategy has been involved. We need to add an intermediate level of play and some method of choosing the computer's moves. I'm calling this an intermediate level, so you can add your own advanced level and perhaps a more sophisticated way of winning.

The Computer Gets Smarter

The computer's intermediate level of play is defined in lines 51–88. The strategy used is first to get the center spot if it is available (line 52). On later turns, if the computer has the center spot, it checks for wins achieved by filling the two diagonals. The columns are checked in lines 58–63. If an opposing marker is in the column, the column is ignored. If there isn't an opposing marker, there is a check to see if two of the computer's markers are in the column. If so, a marker is placed in the remaining spot to win. The rows are checked similarly in lines 64–69.

If the computer doesn't spot a winning possibility, it will then check to prevent the opponent's winning. If there are two of the opponent's markers in any column, row, or diagonal, the computer will block the win (lines 70–87).

If the computer does not spot a column, row, or diagonal with two like markers in it, it simply chooses a place at random.

In the IF-THEN statements, P(K) will contain the value of the marker in a particular position, number K, where K is one of the nine positions. P(K) can be 0 if no marker is present, or 3 or 1 if there is a marker present. After the THEN you can set E to the position chosen, then GOTO a different line.

CLR or Crash

The command CLR is used in line 96 when the option to play again is chosen. This command clears the memory of all variables and unsatisfied FOR-NEXT loops and GOSUB-RETURNs. Without CLR, you will get an OUT OF MEMORY message after several games, which can be caused by jumping out of FOR-NEXT loops with an IF-THEN statement or having too many GOSUBs without a RETURN in effect. Notice that the IF statements in this program transfer control out of FOR-NEXT loops and out of subroutines.

The last step of programming was to add the title and instructions at the beginning of the game. I usually PRINT the

title and instructions as I am defining variables for the program. The title and instructions are in lines 28–31. The options of markers and level of game are in lines 32–37.

The program isn't complete until you test it. Game programs usually involve quite a bit of testing. You need to check all types of player input—right choices, wrong choices, other keys. In this particular game I had to check the player choosing first move or second move and beginner level or intermediate level (all combinations). I also checked the player winning, the computer winning, and a tie game. The supreme test is to have someone else try the game for you.

When you type in this game, be sure to leave out all unnecessary spaces. Notice that the lines are numbered by ones to conserve memory.

Tic-Tac-Toe

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 GOTO28                                                    :rem 212
2 POKEM,77:POKEM+3,78:POKEM+23,77:POKEM+24,78:POKE
  M+45,78                                                   :rem 158
3 POKEM+46,77:POKEM+66,78:POKEM+69,77:RETURN
                                                            :rem 202
4 POKEM,85:POKEM+1,67:POKEM+2,67:POKEM+3,73:POKEM+
  22,66:POKEM+23,32:POKEM+25,93                             :rem 222
5 POKEM+44,66:POKEM+47,93:POKEM+66,74:POKEM+67,64:
  POKEM+68,64:POKEM+69,75:RETURN                            :rem 111
6 FORI=M+C TOM+C+66 STEP22:POKEI,CC(N):POKEI+1,CC(
  N):POKEI+2,CC(N)                                          :rem 181
7 POKEI+3,CC(N):NEXT:RETURN                                 :rem 241
8 POKEF1,231                                                :rem 160
9 FORD=1TO60:NEXT:POKEF1,0:RETURN                           :rem 170
10 X=N:P(E)=X:M=S(E):ONX GOSUB2,2,4                          :rem 69
11 GOSUB6                                                    :rem 24
12 FORI=1TO7STEP3                                            :rem 74
13 IFP(I)<>P(I+1)THEN16                                      :rem 51
14 IFP(I)<>P(I+2)THEN16                                      :rem 53
15 ONP(I)+1GOTO16,89,89,89                                  :rem 192
16 NEXT                                                     :rem 166
17 FORI=1TO3                                                :rem 220
18 IFP(I)<>P(I+3)THEN21                                      :rem 54
19 IFP(I)<>P(I+6)THEN21                                      :rem 58
20 ONP(I)+1GOTO21,89,89,89                                  :rem 184
21 NEXT                                                     :rem 162
```

```
22 IFP(5)<>X THEN25                                    :rem 49
23 IF(P(1)=X)AND(P(9)=X)THEN89                         :rem 223
24 IF(P(3)=X)AND(P(7)=X)THEN89                         :rem 224
25 FORI=1TO9:IFP(I)=0THEN27                            :rem 153
26 NEXT:PRINT"TIE GAME!":GOTO92                        :rem 173
27 N=ABS(N-4):RETURN                                   :rem 228
28 PRINT"{CLR}{BLU}":PRINTTAB(5)"TIC-TAC-TOE":FORI
   =1TO9:READS(I):NEXT                                 :rem 191
29 DATA7726,7733,7740,7880,7887,7894,8034,8041,804
   8                                                   :rem 98
30 PRINT"{2 DOWN}CHOOSE ONE OF THE":PRINT"POSITION
    NUMBERS.":PRINT"{DOWN}GET 3 IN A ROW.":rem 102
31 POKE36878,15:F1=36876:C=30720:CC(1)=6:CC(3)=2:Y
   =3:Z=1:H=2                                          :rem 69
32 PRINT"{DOWN}{BLK}X{BLU} GETS FIRST MOVE.":PRINT
   "{DOWN}CHOOSE {BLK}F1{BLU} FOR {BLK}X":PRINTTAB
   (7)"F3{BLU} FOR {BLK}0{BLU}"                        :rem 8
33 GETE$:IFE$<>"{F1}"ANDE$<>"{F3}"THEN33               :rem 57
34 IFE$="{F1}"THENY=1:Z=3                              :rem 95
35 PRINT"{2 DOWN}CHOOSE {BLK}F1{BLU} BEGINNER":PRI
   NTTAB(7)"{BLK}F3{BLU} INTERMEDIATE"                 :rem 153
36 GETE$:IFE$<>"{F1}"ANDE$<>"{F3}"THEN36               :rem 63
37 IFE$="{F1}"THENH=1                                  :rem 77
38 PRINT"{CLR}":T=INT(6*RND(0))+2:FORI=1TO9:P(I)=0
   :NEXT                                               :rem 182
39 FORI=7709TO8129STEP22:POKEI,160:POKEI+C,T:POKEI
   +7,160:POKEI+7+C,T:NEXT                             :rem 46
40 FORI=7835TO7854:POKEI,160:POKEI+C,T:POKEI+154,1
   60:POKEI+154+C,T:NEXT                               :rem 80
41 FORI=1TO9:POKES(I)+23,I+48:POKES(I)+23+C,0:NEXT
                                                       :rem 250
42 N=1:IFH=2THEN51                                     :rem 55
43 IFY=1THEN46                                         :rem 86
44 E=INT(9*RND(0)+1):IFP(E)<>0THEN44                   :rem 58
45 GOSUB10                                             :rem 74
46 GOSUB8                                              :rem 34
47 GETE$:IFE$=""THEN47                                 :rem 1
48 IFE$<"1"ORE$>"9"THEN47                              :rem 116
49 E=VAL(E$):IFP(E)<>0THEN46                           :rem 126
50 GOSUB10:ONH GOTO44,52                               :rem 153
51 IFY=1THEN46                                         :rem 85
52 IFP(5)=0THENE=5:GOTO88                              :rem 2
53 IFP(5)=X THEN58                                     :rem 254
54 IFP(1)=0ANDP(9)=Z THENE=1:GOTO88                    :rem 64
55 IFP(1)=Z ANDP(9)=0THENE=9:GOTO88                    :rem 73
56 IFP(3)=0ANDP(7)=Z THENE=3:GOTO88                    :rem 68
57 IFP(3)=Z ANDP(7)=0THENE=7:GOTO88                    :rem 73
58 FORK=1TO3:IF(P(K)=X)+(P(K+3)=X)+(P(K+6)=X)THEN6
   3                                                   :rem 207
59 IFP(K)+P(K+3)+P(K+6)<>2*Z THEN63                    :rem 158
```

```
60 IF P(K)=0THENE=K:GOTO88                          :rem 45
61 IFP(K+3)=0THENE=K+3:GOTO88                       :rem 234
62 E=K+6:GOTO88                                     :rem 121
63 NEXT                                             :rem 168
64 FORK=1TO7STEP3:IF(P(K)=X)+(P(K+1)=X)+(P(K+2)=X)
   THEN69                                           :rem 63
65 IFP(K)+P(K+1)+P(K+2)<>2*Z THEN69                 :rem 155
66 IFP(K)=0THENE=K:GOTO88                           :rem 51
67 IFP(K+1)=0THENE=K+1:GOTO88                       :rem 236
68 E=K+2:GOTO88                                     :rem 123
69 NEXT                                             :rem 174
70 FORK=1TO3:IF(P(K)=Z)+(P(K+3)=Z)+(P(K+6)=Z)THEN7
   5                                                :rem 210
71 IFP(K)+P(K+3)+P(K+6)<>2*X THEN75                 :rem 153
72 IFP(K)=0THENE=K:GOTO88                           :rem 48
73 IFP(K+3)=0THENE=K+3:GOTO88                       :rem 237
74 E=K+6:GOTO88                                     :rem 124
75 NEXT                                             :rem 171
76 FORK=1TO7STEP3:IF(P(K)=Z)+(P(K+1)=Z)+(P(K+2)=Z)
   THEN81                                           :rem 66
77 IFP(K)+P(K+1)+P(K+2)<>2*X THEN81                 :rem 150
78 IFP(K)=0THEN E=K:GOTO88                          :rem 54
79 IFP(K+1)=0THENE=K+1:GOTO88                       :rem 239
80 E=K+2:GOTO88                                     :rem 117
81 NEXT                                             :rem 168
82 IFP(5)=Z THEN87                                   :rem 4
83 IFP(1)=0ANDP(9)=X THENE=1:GOTO88                 :rem 64
84 IFP(1)=X ANDP(9)=0THENE=9:GOTO88                 :rem 73
85 IFP(3)=0ANDP(7)=X THENE=3:GOTO88                 :rem 68
86 IFP(3)=X ANDP(7)=0THENE=7:GOTO88                 :rem 73
87 GOTO44                                           :rem 16
88 GOSUB10:GOTO46                                   :rem 46
89 IFX<>Y THEN91                                    :rem 196
90 PRINT"YOU WIN!!!":GOTO92                         :rem 102
91 PRINT"COMPUTER WINS!!"                           :rem 45
92 FORI=1TO20:POKEF1,INT(20*RND(0)+200):GOSUB9:NEX
   T                                                :rem 112
93 PRINT"{20 DOWN}TRY AGAIN? (Y/N)";                :rem 145
94 GETE$:IFE$="N"THEN97                             :rem 86
95 IFE$<>"Y"THEN94                                  :rem 25
96 CLR:GOTO28                                       :rem 45
97 PRINT"{CLR}":END                                 :rem 229
```

World Building

Creating Game Concepts

Gregg Keizer

Most games, whether fast-action or text adventure, have a story within them. A game is interesting not because of its play mechanics, which are often repetitive, but because of its story. You may know how to program, but how do you come up with a game story? This article tells you how.

I like stories. Almost any kind of stories. Adventure stories, mystery stories, fantasy stories. In fact, one of my professions is story writing.

But I also like games. I've played board games since I was a teenager, and with the computer and arcade explosion I've gone on to enjoy computer and arcade videogames. Not because I like pressing buttons or moving a joystick, but because the story, or the *world,* of the game is so entertaining.

Designing games is an art in itself, just like programming. The two are often thought of as the same thing, but that's not necessarily true. Many software companies have recognized this and split up the functions. Someone who can design a good game by creating an interesting, entertaining game world may not know how to program a computer. Others who know a computer inside out may not know how to come up with a good game concept. So two people work together in the creation of a game.

The *designer* sits down and decides what the game world will be and what will happen in that world. Everything that is in the world and that can occur in that world is written down in plain English. No FOR-NEXTs, no GOSUBs. Then it's passed on to the *programmer,* who actually writes the program in BASIC or machine language. Of course, some of the things the game designer requested may be impossible, or too difficult, or take up too much memory. Compromises may have to be made. There's a lot of give and take in creating videogames.

Unfortunately, when you create a game yourself, you

don't have the luxury of splitting the designing and programming functions. You have to do it all yourself. Unless you're schizophrenic, it can be difficult working in two such different areas.

Later in this section, Orson Scott Card's article, "Thinking It Through: How to Plan a Videogame," shows you how to plan a game program. But first you'll have to come up with concepts and ideas. Let's look closer at that part of game creation.

Storytelling

A story, whether printed on the page or part of a videogame, has several components. Two vital ones are the story line (sometimes called the *plot* in English classes) and the setting (or *milieu)*—in other words, *what happens* and *where it happens.* Of course, there are other elements too. Things such as characterization, theme, and conflict are often discussed when people talk about stories. But let's stick to just two: the world and the story line. Everything else comes from those elements anyway. For instance, characters must be part of their world to be believable, and conflict arises from the combination of characters and story line.

Before you begin programming, you need an idea of what you're trying to do. Planning is important, and Orson Scott Card covers that in detail. Your first step in planning is to have that game world and story line firmly in mind, long before you sit down and start typing on the computer's keyboard.

But how do you come up with worlds and story lines?

A Borrower Be

You can borrow them. If there's a game you really like, perhaps there are parts of its world and story line you find so interesting that you want to use them in a game of your own. It's not plagiarism, as long as the final product is different from the original.

One of my favorite games is *Joust.* Perhaps you've seen it in the arcades. You take the role of a knight mounted on a creature that looks something like an airborne ostrich. Wildly flapping its wings, your steed zips around the screen, bumping into levitating islands and knocking enemy knights from their mounts.

But there are certain pieces of *Joust* I would like to use in a game of my own. The world includes a flaming lava lake, complete with a troll who sometimes reaches out and snares you. I'd like to use that piece of *Joust*'s world, although with a few modifications. What if you had a game where your character hovered over a quiet lake? But instead of trolls reaching out for you, you had to blindly grope for people trapped under the surface. There still might be trolls grabbing for you, but the object would be to rescue as many people as possible. The lake might be muddy part of the time, making it impossible to see whether you're reaching down for a troll or a prisoner. Other times it may clear, but just for a moment, so that you can dart under the surface and pull another captive free from the troll's domain.

You've just created a completely new game world, simply by borrowing one element from another game. You can do the same with a game's story line, but it's more difficult. How could you modify the story line in *Joust*, for example, yet end up with something completely different? Let's try.

In *Joust*, there are several rounds in each game. Some rounds reward the two players for cooperating against a common enemy. Others give enormous bonus points to the winner of a player-against-player battle. In those rounds, you try to ignore the enemy knights and concentrate on dismounting the other player. Maybe you could use that story line element (cooperation and competition) in a game of your own.

What about a two-player text adventure game? Both players are trying to find their way out of a labyrinth of rooms and hallways. Just reaching the outside is the game's object. But one player cannot do it alone. There are places where both players must help each other in order to move closer to the outside. Perhaps it's a wall that's too high to climb alone. And there are no tools, such as ropes or ladders, to help. One character must boost another to the top of the wall. Then the character on top of the wall has to lean down and pull the other up. Betrayal may be a possibility. But what if there's another wall further along? That's a decision the players have to make. Hindering each other would be easy to do, but then neither would see daylight. That gives you the basis of another game, just by extrapolating a single element of *Joust*'s story line. It wasn't as difficult as it seemed, was it?

You can borrow bits and pieces from other things besides games, however. Movies, short stories, even everyday activities can lend ideas for your game world and story line. All you have to do is look for them.

World Building

But what if you want to come up with a game world and story line from scratch? One that's truly original, as far as you know. How do you do that?

Many of the best videogames are those which have borrowed little or nothing from other games. *Joust,* to stick with our example, is fascinating because it's so fresh and original. Another game I enjoy, *Seven Cities of Gold* from Electronic Arts, may owe a debt to some board game simulations I've played (as well as to historical fact), but it still has that original and unique feel that makes a game good.

Something I've used when creating worlds and story lines for my own short stories and novels is a process I call *world building.* It's really an exercise in brainstorming, where you toss out as many ideas as possible, then make one selection or choice after another. It works like this.

Beginning. It's easier if you start with the world or story milieu. This is probably the hardest part, simply because it's the beginning, but you have to start somewhere. Decide what *kind* of world you want your game to inhabit. Will it be a fantasy world, like that of *Joust,* complete with knights and fantastic creatures? Or will it be something from science fiction? Perhaps you're planning a text adventure game where the player explores an alien city, looking for advanced technological discoveries. Or maybe you favor a more realistic setting, like the streets of New York City. Choose something general, because the rest of the world-building exercise will be a process of repeatedly narrowing the universe you have selected.

For the sake of discussion, make a decision: The world will be an alien one, far removed from your own both in time and in space.

Winnowing. The process of throwing some things out, and keeping others, begins here. What will be found in this alien world? What kind of settings are possible? It could be a prairie planet, where grasslands extend from one side of the continent to the other. Native creatures inhabit this grassland.

Some of them are dangerous, others can be domesticated.

How do we keep them separate in our game-to-be? With a fence. There's a fence-line stretching across this grassland. The domesticated creatures are on one side; the wild ones are in the wilderness on the other side. But the wild creatures sometimes cross the fence-line. Perhaps they break through it, or maybe they manage to leap over it. (That choice you may want to save for later, when you're dealing with the game's mechanics.) However they do it, the wild creatures get into the more civilized area.

What do they do when they've broken through? Again, more winnowing is called for to sort the pieces of this world. Damaging the tame herds of creatures is an obvious, but not terribly original, result. Think of something else. What about contamination? What if the wild creatures infect the domesticated herds? Not with a disease, perhaps, but with the idea that freedom lies just beyond the fence. Since this is an alien world, with alien creatures, we can assume they're more intelligent than beasts such as cattle. They think, even rationalize to an extent, so the concept of freedom is possible.

So the wild creatures have the ability to convince the others that they should cross the fence and return to the wilderness. The player, however, has a vested interest in keeping the herds intact. It's the player's job to keep the tame creatures on the right side of the fence-line.

Story line. You've already been thinking about the story line, whether you know it or not. As soon as you start thinking about the world, you're automatically thinking about what *happens* in the world—and that's what story lines are made of. That's the magic of world building. By tossing out ideas, keeping some and rejecting others, you can easily slip into consideration of the game's story line.

The player controls the herds, but only indirectly. It's impossible to just talk to the tame creatures and persuade them to stay. Instead, the player has control over a few natives who can communicate to the herds. Those natives are scarce, and thus valuable. By moving these natives from herd to herd, the player can calm the creatures. But the natives can't be everywhere at once. The player would have to decide where and when to send his native translators to talk to the herds. How that's actually done in a game is up to you. Again, that's a decision you would probably make later, during the

setting up of the game's mechanics.

If the player wants to risk it, the native translators could be sent out to eliminate any wild creatures that have crossed the fence (or even sent into the wilderness to search out and destroy them all). But that's risky because there's a chance that the natives will be hurt or even killed. With each native who's eliminated, there's less chance of keeping the domesticated herds in line.

Got the Idea?

Almost without realizing it, you've got a game ready to program. You have developed a world, created characters to inhabit it, and decided on the abilities and skills of those characters. You also have a story line. Something happens in the world, and the player is part of the resulting action. How the player reacts to the story line decides the outcome of the game. You've even got conflict. The player wants to retain the herds, while the wild creatures want to take them away. That's all you need for an excellent, original game.

World building isn't difficult. It really amounts to little more than thinking things through. Brainstorming is the key. Don't be afraid to include even the most fantastic idea in your game world or story line; they can always be discarded later, and they may even make your game so unique that it becomes one of the all-time classics.

Try out world building when you design your next game. Like professional game designers and programmers, you may have to make some concessions when you actually begin programming. That's unavoidable. But at least you've got a framework to build on. It's far better than the alternative—sitting down at the keyboard, ready to program, but not knowing *what* to program.

Thinking It Through

How to Plan a Videogame

Orson Scott Card

There's more to designing and writing a computer game than just programming. Before you begin typing those POKEs, PEEKs, GOTOs, and GOSUBs, you should have an idea of what your game will do, how it does it, and why.

You don't create a videogame by simply sitting down at the computer and typing in BASIC or machine language commands. A videogame is a complex program, and if you don't think it through in advance, you're begging for hours and hours of needless revision and debugging—with a good chance that you'll end up with a second-rate game. But if you develop a well-thought-out plan before you write the first line of the program, it can be smooth and pleasant.

The plan I'm talking about isn't a matter of flowcharts and diagrams and dull calculations. That's the engineering approach to game creation. It works for building bridges or CPUs, but it doesn't cut it when developing computer games. Good games aren't engineered. They're created.

The process of planning a game may seem much like daydreaming. You visualize the figures, the scenes, the objects on the screen. You think of the way they move and the way they're controlled. What goes on when they bump into each other. How they affect and change each other. It's as though you leaned back in your softest chair and spent a few hours telling yourself stories. In other words, it's play-testing a game that hasn't yet been programmed.

Every gamewright creates games out of his or her imagination. Two games will never be exactly alike, unless one game designer is deliberately trying to copy another's work. But the basic design requirements are much alike from game to game.

So let's go through the steps involved in planning a game. You'll undoubtedly think of things that didn't occur to me—and by the time you're through reading this chapter, you may well have a complete game design of your own. I hope so.

Getting the Idea

A game idea can come from almost anywhere. You might be playing a conventional game and suddenly realize, "This could be better on the computer." So you begin to think of ways to use your system to simulate baseball or a board game or charades. It doesn't even have to be a game. For instance, how would you duplicate the work of a traffic cop and turn it into a game? Games that are based on conventional games or real-world activities are called *simulations*.

You might notice a setting that is particularly dramatic: the naked girders and beams of a skyscraper under construction; the dramatic arroyos and mesas of the Arizona desert; the vast distances and three-dimensional movements of outer space. It doesn't even have to be a setting that actually exists. The designer of *Joust* may well have gotten his flying islands from seeing Roger Dean's fantastic illustrations for the covers of Yes albums.

Once you have a dramatic *milieu*, or setting, you develop the game from there: What could happen in this world I've imagined? You might start with a movement or *play mechanic* you want to create. There are many different play mechanics already in use. For instance, the *Pong*-type games use paddle controllers to move the player's figure instantly from one point on a line to another. Many games use direct joystick movement, where the player moves the joystick in a direction and the figure moves that way on the screen.

Others are much more complex, however. In *Joust*, a left-right joystick determines the horizontal direction and speed, but because the figure is a knight mounted on a flying ostrich, vertical movement is done by repeatedly pressing a button which causes the ostrich's wings to flap and the bird to rise. No one had ever used this play mechanic before, and the game may well have begun with that idea. Similarly, the game *Mario Bros.* introduced the "bump from beneath" play mechanic, in which the figure attacks enemies by getting one level beneath them, jumping up, and bumping into the floor that the enemy happens to be walking on. Actually, that is

only a small variation from the *Donkey Kong* play mechanic, yet it makes a huge difference in the feel and play of the game.

There isn't anything wrong with adapting ideas from other games, as long as you make enough changes that the game becomes your own. It's only natural for you to look at what another designer has done and think, "Why wasn't *this* done, too?" *Donkey Kong*, for instance, obviously owes a debt to *Space Panic*, a ladder game, by way of *Jump Man*, which introduced the press-the-button-to-jump play mechanic. But the *Donkey Kong* designers got rid of the holes and added moving obstacles (barrels and flames).

The designer of *Lode Runner* (by Doug Smith, for Brøderbund) also started with *Space Panic*—you'll recognize the "stonework" floors and ladders—but he went in quite a different direction from *Donkey Kong*. Hole-digging was kept, but other features were added—and each new screen became a new geometric puzzle. Both *Lode Runner* and *Donkey Kong* stand on the shoulders of the earlier game *Space Panic*. But both are so different that they qualify as genuinely new and creative games.

Wherever your game idea comes from, though, you eventually need to take all these things into account. You'll have to come up with a play mechanic and a milieu and adapt any real-world features that might be part of your game. And since you have known and loved (and probably hated) quite a few videogames, you'll be borrowing or avoiding features from other gamewrights' work, whether you mean to or not.

The Story of the Game

Most fast-action games have a story, and since I'm a story writer by profession, it's hardly a surprise that this is where I start *my* game designs. Some stories are pretty rudimentary, like *Space Invaders* or *Asteroids:* The bad guys are coming, so either shoot or get shot. Others are more complex, like the story line of *Donkey Kong*: Ape catches girl, Mario climbs up and performs insanely heroic acts until he reaches girl, ape falls down, and girl kisses Mario. It's the variety of obstacles, milieus, and actions Mario must perform that give the story line its complexity.

Start with a story.

You're a knight in a castle in the middle of a river, near the place where the river flows into the sea. An enemy has sailed his fleet up the river and has anchored it around the castle, laying siege to your fortress. His primitive cannon are pounding away at the castle walls, gradually wearing them down. And even if you manage to survive the artillery barrage, you can't get any food supplies or ammunition into or out of the castle.

In the daytime, you can fight back by firing your four cannons at the enemy fleet. However, you have only a limited supply of ammunition. Every shot must count.

At night, you can put out a small boat and attach explosives to the sides of enemy ships, just under the waterline. When the charges explode in the morning, any ships you've successfully mined sink. During the night you can also use your small boat to run the blockade, getting more ammunition from your confederates on shore.

When your castle is worn completely away, or when your little boat is sunk, you lose. When all the enemy ships are sunk or so damaged that they sail away, you win.

Admittedly, it's not much of a story. There is no characterization, and the plot is repetitive. But that's deceptive. The player supplies the characterization. The main character is the player, of course, and the player supplies the plot complications. In early plays of the game, the guy in the little boat is going to be pretty clumsy and slow, and the cannoneer in the castle is not going to be much of a shot. But after a while, the boatman will be rowing circles around the anchored siege ships, and the cannoneer will score hits with every projectile.

How to Control a Cannon

Now that you have an idea, you need to expand on it and think of how it might actually play as a game. It's time to develop a play mechanic.

What does the player control? In the daytime, it's the castle cannons. Make it a four-cornered castle, with a cannon in each corner. The player can only aim and fire one cannon at a time. The cannon he's firing will be white; the other three cannons will be black.

How should he select which cannon to fire? The player could press the 1 key to fire cannon 1, the 2 key to fire cannon 2, and so on. But that forces him to take his eyes off the

screen and choose one of four different keys. Too complicated.

You could designate one key as the "cannon select key." Just press it, and the next cannon would be selected, proceeding clockwise around the castle. That simplifies things—just one key to select. But remember, there also has to be a way for the player to aim the cannon *and* a way to fire it. That's at least two more controls.

The simplest choice of all, at least for the player, is to have the program automatically go from one cannon to the next. Fire cannon 1, and cannon 2 is automatically selected; fire cannon 2, and cannon 3 is then ready to fire. If cannon 3 is out of ammunition, the program can sense that and skip over it. It's simple, because the player doesn't have to *do* anything. He only has to notice the color change to know which cannon he's controlling.

Of course, it means that he can't fire cannon 2 five times in a row; he has to fire each of the others first. But that just adds a little challenge to the game. You can explain this by adding another element to the story: "Once fired, the cannons have to cool down before you can reload them."

There's an even simpler solution, though: Put a central tower in the middle of the castle (the castle's keep) and place a single cannon on top of it. Then there's no selection at all. Just the one cannon, which you aim and fire until your powder runs out.

Simplicity

There you have it: four different ways to program the same basic idea. But which one is best? I don't know. *I* may think the first two are too complicated for a fast-action game, but *you* might think they give the player more freedom of choice. I think the last one, the single-cannon solution, is acceptable (and easy to program). But having four cannons appeals even more. If you left it up to me, I'd choose the four cannons.

Does this seem familiar? It might. The arcade game *Missile Command* had three missile bases, each with its own supply of missiles. But when it was translated to the home machine, it was simplified and given one base instead of three.

You face a similar choice here. Time after time you will have to make such choices between simplicity and complexity. You want a general rule? If you're a novice programmer, go for simplicity. It makes a game easier to program.

You want another general rule? If you're an expert programmer, you should *still* go for simplicity. It makes a game easier to play.

But I still like the four cannons, automatically selected, better than the single cannon in the middle of the fort. So I'll break my own general rule and go for slightly more complex programming—but never for needless complexity in playing. Maybe that's the best rule of all: Make it as tough as you want on yourself, but keep the play mechanics easy for the player.

Aiming the Cannon

But you still haven't decided how to *aim* the cannon!

Rotation seems like the way to go. Could you rotate the cannon so it can be aimed in *any* direction? Maybe. But why not just use the joystick to choose one of eight directions, and fire by pressing the fire button? Wherever it's pointing is where it aims. It will take only a little trial and error for the player to learn to aim fairly precisely. However, rotating cannons will mean we have to have as many different cannon shapes (eight, to be exact) as there are possible directions to aim. Every decision has a cost.

But wait a minute. In the real world (remember, this is partly a simulation) cannoneers also have to determine range. They control range by deciding the elevation of the gun and changing the amount of gunpowder in the charge. You could control that by . . . .

Forget it. This is a game, not an artillery textbook. If you're a cannoneering purist, you might want to go through the agony of programming all the math. But you'd better find another purist to play it, because the game will be complex and slow. As far as I'm concerned, I'll let an imaginary cannoneer figure the elevation and charge. As long as the player selects the correct horizontal aim, the cannonball will automatically go just the right distance. Another victory for the simple approach.

Movement

There are other play mechanics to worry about. First, the movement of the enemy ships. Do they stay in the same place, or do they move around? Do they always appear in the same locations every time you play, or are their positions randomly chosen?

There are many different options, but my choice would be this: The program will define twenty or so areas, each of which might hold a ship. By defining these areas, you can be sure none of the ships will overlap. However, you'll start with only five enemy ships. The program will randomly decide which areas should be used to produce an arrangement that's slightly different from the time before. That will help keep the game interesting and challenging.

If you were programming in machine language, it would be a simple matter to assign each ship an individual course around the castle and then let it move in a regular pattern. At machine language speeds, you could keep a dozen ships moving without slowing down the game. In BASIC, however, each individual ship movement would slow things down. And since I just decided this will be an all-BASIC game, you can't have the ships move.

At least, not constantly. Anything that happens constantly adds to the amount of time that passes *after* the program has checked for the player's instructions and *before* it checks again. The less often the program checks for the player's instructions, the slower and less responsive the game will feel. Too much of that, and it isn't a fast-action game anymore!

But you can have *occasional* movement of the ships, and the story provides a perfect excuse. The current of the river always pushes the ships downstream, but as the tide flows upriver, the current moves the opposite direction. The tide flows twice each day, once in the daytime and once at night. When the tide flows, the ships swing around so they are on the upriver side of their anchors. When the tide ebbs and the river's current takes over, the ships swing around to the downriver side of their anchors.

That means that half the day all the ships will face one way, and half the day they'll face the other. It's a simple change, but it adds to the completeness of the milieu. It supports the story, and it adds to the realism of the simulation.

Controlling the Boat

In a two-phase game like this, it's good if the two phases can be controlled in a similar way. For instance, you can let the same key that rotates the cannon also rotate the boat. Press the Cursor Left-Right key and the boat rotates counterclockwise. Press the fire button, and the boat leaves behind a mine.

That takes care of directions and mine-laying, but what about movement through the water? You need a third control, to give the boat speed. We could steal a page from *Joust* and let the oars move each time we press a certain key—the space bar, for instance. Each time you press the bar, the boat surges forward a little ways in the direction it's pointing. If the boat is going against the current, the surge is weaker; if it's going with the current, the surge is much greater. And if you don't row at all, the boat will drift in the direction of the current. If the boat leaves the screen, it's gone.

That's one solution. If I was working in machine language, it's the one I'd probably use. There is a simpler choice, however. You could use a joystick throughout the game. During the cannon phase, pushing the joystick in *any* direction makes the cannon rotate, and pressing the fire button makes it fire. In the boat phase, however, the boat moves in the direction the joystick points, and keeps moving in that direction as long as the joystick is pushed. The button lays mines. This has the virtue of being simpler; also, since the boat can only move in either four or eight directions (depending on whether you allow diagonals) you need to have only one boat shape for each direction. Thus, programming and play will be much easier.

It was important to aim the cannon in many different directions, because the cannon couldn't actually move. But since the boat *can* move around, having only four directions available isn't necessarily a serious limitation. And you can still keep the tidal drift in both versions.

The Planning Outline

I've gone into a lot of detail in order to show you what the thought process can be like and the reasons for some decisions. But from now on, so that this chapter doesn't become the whole book, you'll move much more quickly through an outline of the decisions that need to be made. You can use this outline for almost any arcade game plan.

Play mechanics. What does the player control in the game, and how? (In this case, it's a cannon and a boat, both controlled with a joystick, as well as cannon fire and mines.)

Simulation. In what ways does the game correspond to real-world activities? To what degree can you duplicate reality without making the game too complex to play? (I decided to

let the cannon aim, but not determine range, and to let the player use the joystick to move the boat in only four possible directions.)

Milieu. What is the setting? What is on the screen besides the moving figures? This is more than just decoration. If you have an airplane game, clouds drifting across the screen add to the illusion of reality. It makes the player feel more like he's really flying. (In this game, make the flow of the river left to right. That means that the banks of the river will be across the top and bottom of the screen. By using character graphics, you can PRINT each shore in a single string. The castle will be right in the middle, but because the TV screen is wider than it is tall, the castle will be rectangular too. It will be a top view, as a bird sees it.)

Missiles. This is a generic term. The player's figure is the screen object whose movement the player controls. Once a missile is launched, however, the player has little or no control over it. The missile can be the ball in a football or baseball simulation, the bullet in a shoot-out, or even the player-figure's fist if it can be "launched" against an enemy. (The only missile used is the ball fired by the cannon, which goes straight in the direction it's fired until it either hits an enemy ship or goes off the screen.)

Collisions. What happens when the player-figure bumps into something on the screen? The figure can respond to the object in several ways. The object might be ....

Transparent. The figure just keeps going as if the object weren't there.

A wall. The figure can't move any further *toward* the object but can slide along it.

A tar baby. Once the figure touches the object, it's stuck.

A bomb. Touching the object is deadly.

A billiard cushion. Touching the object makes you bounce off at an angle.

A balloon. When you touch the object, it disappears, but you're unharmed.

The same responses are possible for missiles.

During the boat phase, for instance, the shore and the castle are walls (the boat can slide along them). However, the enemy ships are tar babies: When you touch one, you stick until you lay a mine and release yourself. The ships are then

marked, and at the end of the nighttime phase all the mines explode at once.

During the cannon phase, the cannonball treats the ships as balloons—they explode when the ball hits them. However, in this particular game, I decided to make the castle and the shore act like balloons too. The player, by clumsy aiming, can destroy the castle *and* the shoreline. In such a case, when he goes to the shore to replenish his supplies, the shore might not be there anymore. And if he's really clumsy, he can even help the enemy finish off his own castle.

But what about the enemy's cannonballs? To save programming headaches, don't actually show those cannonballs. Instead, from time to time a randomly selected, character-sized piece of the castle will explode and disappear, leaving bare rock behind. It can be assumed that an enemy cannonball hit that spot.

Reward and punishment. Games are like life. You obey the rules so that good things will happen. The most common reward is the score—it gets higher each time you do the good things (like blasting the enemy out of the water). There are other rewards, too, however, such as story awards. There are also puzzle awards. Just solving the problem on one level, so you can finish it, is rewarding. The best games have scores, story rewards, and puzzle rewards too.

In this game, if you don't ever get the enemy ships, they'll wipe out your castle. If you do get them, you get points and eventually complete the screen by forcing the enemy ships to go home or by sinking them all. The scoring is fairly complex. Every time one of your castle blocks is destroyed, you lose points; if one of your cannon is blown up, you lose the cannon and a lot more points. However, you get some points just for staying alive and for the number of cannonballs you have left at the end of a game.

Communication. The player needs a lot of information during the game. Did my missile hit its target? Did the enemy score a hit against me? Which object do I control? What in the world am I supposed to *do*?

The single most useful tool you have in communicating with the player is *sound*. Different sounds mean different things—and you don't have to be watching a particular spot on the screen to get the message.

However, explosions and movement also communicate. You'll also use displays of numbers on the screen to tell the player his score, as well as numbers or little pictures (icons) to show how many lives the player has left. You'll want introductory and closing screens to convey more involved messages or tell part of the story in words.

In this game, you might decide that a popping sound says that the cannon has been fired or the mine has been attached. A low *boing* says that a piece of the castle has been blown up. A swishing sound tells the player that the tide is about to change, an explosion says that a ship has been hit, and a much louder and longer sound (followed by a glug-glug sound) conveys the message that a ship has sunk. Sad or happy music is used to signify defeat or victory.

Win-lose conditions. The game has to end sometime, even if only on a current level. You have to decide what conditions end the game and then check from time to time to see if those conditions have been met. The simplest way to do this is to have a variable—XX, for instance—that usually has a value of 0. Then, in any subroutine that has the power to end the game (usually a collision subroutine or a timer subroutine), XX is set to 1 for defeat and 2 for victory. A line in the main loop reads ON XX GOTO 900,920. That jump will occur only when it's time for the game to end. But because it executes only from the main loop, it's much easier to end any FOR-NEXT loops you might be in at the time.

In this game, a player can lose when all four cannons or the entire castle is destroyed, or if all ammunition and mines are used up without destroying the entire enemy fleet. You can win by scoring a certain number of cannonball hits against the enemy or by sinking a certain number of ships or by staying alive until the enemy runs out of ammunition.

Levels. Computer games tend to go on forever. When you meet the win conditions, the game starts over, but it's harder. This allows a novice to get the hang of the game without getting instantly destroyed, while more experienced players still find higher levels more challenging.

In the early levels of the game, for instance, the enemy will have less ammunition, the player cannot blow up parts of his own castle, and there are fewer enemy ships, so the enemy fires less often and the player can blow up all the ships more easily. In addition, the enemy cannot hit the player's cannons.

These features are changed with each level, however, until at expert levels the player doesn't have enough ammunition to survive. He must pick some ammunition up from the shore and bring it back. On higher levels the player will also find that there are as many enemy ships as can fit on the screen without overlapping, that the player can damage himself, that the cannons can be blown up, and that the current in the river is stronger. The list can go on and on.

Animation. If you're particularly ambitious, there are a lot of extras you can add to enhance realism. These things don't actually affect the play of the game, but they *do* make it more fun. Simple animation of figures is easy enough, using either sprites or custom characters, and it can be done with almost no degradation of playing speed.

However, you can also add much more complex animated sequences when play action is stopped. These are like small movies that help support the story (like the opening sequence of *Donkey Kong* in which the gorilla carries the kicking girl up the ladders) or make the milieu complete and believable (like the riderless ostrich in *Joust*, that must make its way offscreen after the knight is defeated). The game would play just as well without these extra sequences, but some of the fun would be gone.

Translate the Plan into a Program

You've jotted down your ideas, you've play-tested the game in your imagination, and you're satisfied. Now you're ready to go to work on the program itself.

The first step is to design your video. How large should the castle be? The screen is 22 characters wide and 23 characters high. Subtract a row at the top and the bottom for the shoreline; also subtract a column at the right which you won't use because PRINTing characters in the rightmost position on a line can mess up the lines below. Then decide how big the castle and river (and boat) should be.

In this case, you decide that the castle should be four characters high and six characters wide. Each corner tower will be a two-by-two square, and each cannon will be a custom character. The player-controlled boat will also be made up of custom characters, as will the enemy ships. They will be PRINTed on the screen as strings. They will usually face either left or right, and all the ships change direction together, which

greatly simplifies animation of the direction changes.

To make all of this work, you need to map video memory and design your character set tables before you even begin to program. It involves lots of tedious calculation, changing dots into bits and bits into decimal numbers and so on, but it will pay off in the end.

Setting Up the Main Loop

You should design the program so that the main loop does as little as possible. The less the main loop does, the more often it repeats and the faster the game plays.

Your game will be a little more complex because it has two main loops, one during daylight phase and one during the nighttime phase. A timer decides how long each phase lasts. Both main loops must check, from time to time, to see if the time is up and the tide should change or the phase should end.

What needs to be in the main loop? It should always contain the routine that gets instructions from the player. In the daylight phase, check first to see if a cannon was fired. If yes, jump to the subroutine that carries the cannonball to its target, where there either is or isn't an explosion. In machine language, you could let a player start aiming or firing the next cannon while the previous cannonball was still moving, but in BASIC it's better to make the player wait for the cannonball to hit before letting him fire again. BASIC would lose too much speed trying to do it all at once.

If the player didn't say to fire, check to see if he wants to aim. If so, jump to the subroutine that moves the cannon.

When that's over, check to see if it's time for an enemy ship to fire again. If it is, jump to the other main loop. Otherwise, go back and start over. Simple enough—the main loop will be fast and tight.

The main loop for the nighttime phase is not so tight. The fire button only places a mine (there's no missile to keep track of), but movement is more complex. The cannon can never run into anything, but the boat can run into enemy ships, the shore, or the castle. The main loop must check to see if the player wants to move; if so, it jumps to the movement subroutine. The subroutine checks to see if the boat has bumped into anything, and if so, what. It might jump to one of the routines that handles collisions, and the movement is changed

accordingly. Also, there is the pull of the current that can gradually move the boat whether the player wants it to move or not. This can't be executed every time through the loop, or the player will never be able to row against the current. Finally, check for the end of the phase and close the loop.

Planning for All the Subroutines

Programming goes much more smoothly if you've decided before you start what subroutines you'll need and where they will be. It helps you keep track of what's going on if you put related subroutines near each other. For instance, in a BASIC program you might want to have all the sound subroutines begin in the 900s, and all the collision subroutines in the 800s.

I usually start every subroutine or group of subroutines at an even-hundred line number. If you use the same approach, you might come up with something like this, which describes all the routines that you'll need:

0–99 Initialization. These lines jump to the setup routines and set certain parameters. They're executed only once.

100–199 Daytime phase main loop. These lines check for the player's instructions, jump to the aim or fire routines if necessary, then check the timer for the end of the phase.

200–299 Fire routine. These lines are executed only if the player has pressed the fire button during the daytime phase main loop. First the routine checks to see which cannon has been fired and what direction it is firing. Then it moves the cannonball in that line, checking each new character it crosses to see if it has collided with anything. If it collides with a ship, it jumps to the ship explosion routine. If it collides with the shore, it jumps to the shore explosion routine. If it collides with the castle, it jumps to the castle explosion routine. If it reaches the edge of the screen without collisions, the cannonball disappears and execution returns from the subroutine. Whenever the cannon fires, the program also goes to the decrease ammunition routine.

300–399 Ship explosion routine. Accessed only from the fire routine, these lines subtract something from the ship's "strength" value, which starts out higher in the harder levels. If the ship's strength is 0 or less, jump to the sinking ship routine. If the ship's strength is above 0, execute the explosion sound and flash the colors of the ship, cancel the missile's

movement, jump to the score change routine, and return to the daytime phase main loop.

400–499 Shore explosion routine. Accessed only from the fire routine, at the easier levels these lines do nothing but end the missile's movement and return to the daytime phase main loop. At higher levels, the shoreline character is flashed while the explosion sound is executed, then the shoreline character is removed.

500–599 Sinking ship routine. Accessed from either the ship explosion routine or the ship mining routine, this animated sequence causes the ship to sink, decrements the count of ships by one, executes the glub-glub sound, jumps to the score change routine, and then returns to the current main loop.

600–699 Aim routine. Until the joystick is no longer being moved or the fire button is pressed, the current cannon is rotated. This is done by flipping from one cannon character shape to another. The same variable that indexes the sprite shapes also indexes the direction variable, which is used in the fire routine. If the joystick is released, return to the daytime phase main loop; if the fire button is pressed, jump to the fire routine, then return to the main loop.

700–799 Score change routine. This routine can be accessed from many points, but the score change variable must already be set. The value of the score change variable is added to the current score (a negative value will, of course, subtract from the score) and the new score is displayed on the screen.

800–899 Tidal change routine. This routine stops all other action while the ships swing around from their old positions to their new positions. The current flow variable is set to either *left* or *right* (−1 or 1). Then check to see if the changing tide has caused a collision between a ship and the player's boat. If yes, jump to the sunk boat routine. If it is also time to change from one phase to another (which will happen every second tidal change), set the phase change variable, which causes one main loop to jump to the starting point of the other. If the tidal change is the beginning of a new daylight phase, check all the enemy ships. If their mine set variable is on, execute the sinking ship routine for that ship.

900-999 Enemy shots routine. These lines select a random location on the perimeter of the castle. If the chosen character is bare ground—that is, if the chosen castle section has already been exploded—there is a chance that the next castle section inward will be selected. If that castle section has also been selected, there is a chance that the *next* castle section will be selected, and so on. Anytime the random choice decides not to select the next castle section, jump to the castle explosion routine. If the chosen castle section is also an ammunition storage location, jump to the decrease ammunition routine. At higher levels, if the chosen castle section is also directly under a cannon, jump to the blow up cannon routine. The odds of the next castle section being chosen change, making the selection more likely at higher levels, which will have the effect of wiping out the castle more rapidly.

1000-1099 Decrease ammunition. This routine decreases the number of cannonballs during the daylight phase and the supply of mines during the nighttime phase. If the amount of ammunition reaches 0, the player can't fire (or attach mines) until the supply is replenished by a visit to shore.

1100-1199 Castle explosion routine. This causes the selected castle section to flash and disappear, leaving bare earth behind. (Actually, it's tempting to have water characters replace exploded castle sections, so the castle appears to be eaten away by the river as well. That's something else to consider!) The *oops* sound executes.

1200-1299 Blow up cannon routine. These lines cause a cannon to flash, crumble, and vanish. That cannon is removed. The explosion sound *and* the *oops* sound are executed. If it was the last cannon, the lose-the-game variable may be set—you can decide later.

1300-1399 Nighttime phase main loop. Checks for player instructions. If the player calls for boat movement (by pushing the joystick), jump to the boat movement routine. If the player doesn't call for any movement, check the timer to see if a tidal change is in order; if so, jump to the tidal change routine. If not, go back to the beginning of the loop.

1400-1499 Boat movement routine. Try to move the boat in whatever direction the player has requested. If there is a collision with anything but a water character, jump to the boat

collision routine. If the movement takes the boat off the edge of the playing area, jump to the lost boat routine. Otherwise, execute the movement and return to the nighttime phase main loop. Remember that if the movement is either diagonally or directly with the current, it is tripled, and that if it is diagonally or directly against the current, it is halved. Therefore, the normal movement increment should be two scan lines.

1500–1599 Boat collision routine. The only movement that can take the boat away from contact with the ship is the exact opposite of the joystick movement that caused the collision. The only other way to release the boat is to attach a mine. These lines cause a low humming noise and constantly check to see if the player wants to place a mine or back out of the way. If the player backs away, execute the backing movement and return to the nighttime main loop. If the player sets a mine, check to see if there are any mines. If so, turn on the mine set variable for that ship: TS (*shipnumber*)=1. If there are no mines left, ignore the request. Once a mine is set, the player is free to move away, and the humming stops.

1600–1699 Ship bump routine. The only movement that can take the boat away from contact with the ship is the exact opposite of the joystick movement that caused the collision. The only other way to release the boat is to lay a mine. These lines cause a low humming noise and constantly check to see if the player wants to lay a mine or back out of the way. If the player backs away, execute the backing movement and return to the nighttime main loop. If the player lays a mine, check to see if there are any mines. If so, turn on the mine set variable for that ship. If there are no mines left, ignore the request. Once a mine is set, the player is free to move away and the humming stops.

1700–1799 Shore dock routine. If the boat has no mines, its mine supply is renewed. If it has any mines at all, the boat is loaded with cannonballs—the boat-loaded variable is set to 1. The swishing sound is executed and the boat is pushed away from shore automatically. Execution returns to the nighttime main loop.

1800–1899 Castle dock routine. If the boat-loaded variable is set, the ammunition variable is increased and the blip sound is executed. The number of cannonballs added depends on the level. Whether the boat was loaded or not, the boat is

pushed away from the castle and the swishing sound is executed.

1900–1999 Sunk boat. The boat sinks under the water and disappears; either it jumps to the daytime phase or we set the player loses variable—you can decide which one later on.

2000–2099 Sound routines. These include the glub-glub, explosion, oops, blip, and swishing sounds.

2100–2199 Player loses routine. The player has lost a life. Check to see how many lives remain, and either go to the closing routine or next life routine.

2200–2299 Player wins routine. The player has completed the level (sunk all enemy ships or inflicted intolerable damage on the fleet). Add 1 to the number of lives left, do a score change, raise the level by 1, and do the next life routine. (If you want, you can also make the enemy ships sail away.)

2300–2399 Closing routine. PRINT the final score and any final messages. Update the high score and PRINT it; ask the player if he wants to play again. If so, reset variables to 0 (except high score) and start the program near the beginning. If not, restore video memory to its normal configuration and END.

2400–2499 Next life routine. These lines duplicate many of the functions of lines 0–99, except that some initialization routines don't need to be repeated—like defining the character set and setting up video memory. At the end of this routine, jump to the beginning of the daytime loop.

2500–2599 Video memory. Executed only at the beginning of play, this routine sets up the character sets and screen memory.

2600–2699 Screen setup. The original castle is set up as one or two strings to be PRINTed all at once. Likewise, the two shores are set up in strings.

2700–2799 Fleet setup. The enemy ships are assigned their locations—how many there are depends on the level. If desired, you can animate their arrival.

2800–2899 Player setup. The player's cannons and boat are put on the screen.

2900–2999 Initialize variables. The lines from 2900 to 2929 set the values of variables that are set only at the beginning of play. Lines 2930–2959 might set the values of variables that are reset when the game is restarted, while lines

2960–2999 could set the values of variables that are reset at the beginning of a life.

3000–? DATA statements. These lines contain the character shapes and sprite shapes used in the setup routines.

This outline of subroutines is still pretty rough—a lot of refinements would be needed during the actual programming. However, by laying out this kind of plan in advance, you would know where to find the major routines and have a good idea of what routines you need to write. Some of these routines would use only a few lines; others might be much more tighly packed within their hundred-line range. This subroutine outline may not sound like much, but it does give you a great deal of control over the programming process.

Naming the Variables

Many of your routines will use the same variable to get information. For instance, the timer variable may be used by many different routines. Similarly, the directional vector that is used in the cannon-aiming routine will also be used in the cannon-firing routine, to decide what direction the projectile should travel.

The best way to keep track of variables is to name them and write them down so you can refer to the list as you program. You also need to decide which variables are arrays, so that you can use other variables as indices into a table. I've sometimes had one integer array variable indexing another, which was used as the index in a string array—all of them indexed by a loop control variable and a directional variable.

If this sounds hopelessly complex, don't worry. Reading about it is complex. Sitting down and doing it is much easier.

Of course, doing it so the program actually *works* can be a little harder. That's why so much of a programmer's time is spent figuring out why it doesn't work and fixing the bugs.

Getting Back to the Real World

In the real world, you'll never plan *everything* in your game in advance. You may *think* you've got it planned, but once your mind is working it doesn't stop. You'll have more ideas in the middle of programming, or you'll run into a programming hurdle you don't know how to get over. You'll start to improvise, and the results will begin to bear less and less resemblance to the original plan.

Don't worry, though. It'll end up much better than the plan. Yet without the plan, many of the eventual improvements would not have been possible.

The point of planning ahead is not to block your creativity. Instead, dig a channel through which your creativity flows. When you have a clear idea of where you're going, you have a much better chance of getting there quickly.

In fact, so much creation goes on after the initial planning is done that I'd be willing to bet that every person who reads this chapter could set out to program a game that followed my plan perfectly—and no two games would look or play alike. In fact, I'd expect that many games would look and play so differently that an uninformed observer would never suspect that they began from the same plan.

Your own games may be much simpler than this one, or much more complex. The most important thing is to design a game that you would love to play—if your heart isn't in it, your game won't be any fun. You can't fool the players.

And when you're through with the planning and programming, you'll discover a great secret: Game designing is the best game of all. No matter how brilliant your game is, no one will ever have as much excitement and frustration and satisfaction and fun playing it as you had in creating it.

Writing Adventure Games

Gary McGath

Programming text adventure games, those popular interactive games where you communicate with the computer through words, is an art in itself. It's not quite the same as creating an arcade-style game, and this article explains some of the basics.

A text adventure is an interactive computer game in which the player assumes the role of a character in a story. As the player, you control the character's actions by typing in commands. The computer responds with a text description of what your character experiences.

The world of most text adventures is composed of a number of *rooms*, or locations. Your character moves from place to place, or from room to room, where objects or other characters may be found. Sometimes these objects and characters aid you; other times they're dangerous. By using the appropriate commands, you can pick up, examine, and even use these objects and characters.

While professionally written adventure programs often understand complicated sentences, many adventures get by with simple two-word commands. However, the vocabulary of even the best text adventure is quite limited, and the program must somehow indicate whether it understands your commands.

The following dialogue is typical of that found in a text adventure. Your commands are printed in boldface, and the computer's responses are in normal type.

You are in a small room lined with shelves. There are doors to the north and west.
There is a gem on the shelf.

Take gem

Your hand is stopped by an invisible shield around the gem.

Examine shield

I don't know the word "shield."

North

You are in a north-south hallway....

Writing a text adventure offers you a chance to exercise your imagination and set up logical puzzles for your friends. Since it requires no special screen formatting or sound effects, and since the program is doing nothing between moves, text adventure programs are easy to debug. And once you've written your first adventure, you can do more of them just by changing the rooms and puzzles in your old program.

Mapmaking

The first steps in designing a text adventure are to create the *story line* (what will happen) and the *milieu* (where things will happen). We'll assume you've already done that. Here, we'll be concerned mainly with the actual programming techniques you'll use, as well as some of the more practical design processes that are useful to the text adventure programmer.

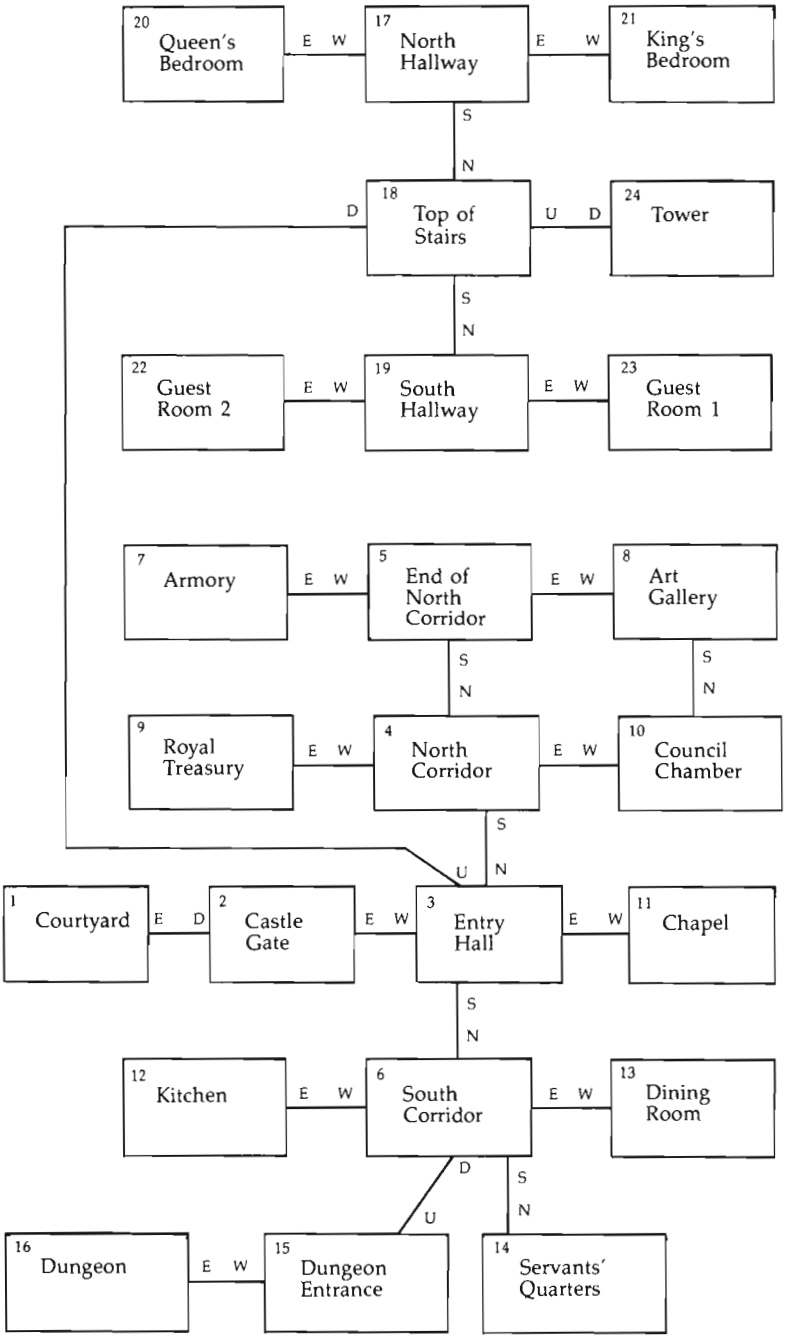
Once you've decided on what your world is, and what will happen in it, you need to design a map of the rooms. Remember that they don't have to actually be *rooms;* we're using that as a generic term. They can be places on a road, paths in a forest, or even corners of a field. Draw a map with a box for each room, and connecting lines labeled with the directions that lead from one room to another (north or south, for instance). Give each room a number and a short description. The room in which the character starts should be *room 1.* Figure 1 shows the map of a typical text adventure game.

Objects, Verbs, and Consequences

In this planning stage, you also need to make several other decisions. Choose the objects that will be in the adventure, and decide where each will be initially located. Some objects might not be in any room at all until the player does something to make them appear. You should also assign numbers to the objects.

Your program also needs a list of the verbs that will be accepted as commands. Certain verbs and commands are almost mandatory, such as NORTH, SOUTH, EAST, WEST, TAKE, DROP, EXAMINE, LOOK, INVENTORY, and QUIT.

Figure 1. The Adventure's Map



Others that might be helpful include ENTER, CLIMB, SHAKE, MOVE, TURN, FIGHT, OPEN, EAT, DRINK, CLOSE, and READ. Abbreviations, such as I for INVENTORY and N for NORTH, are easier for the player to remember and use. Equivalent alternatives like GET and TAKE, which should mean the same thing, can reduce player frustration. Remember, the difficulties in an adventure should come from the logical puzzles, *not* from figuring out how to talk to the program.

What consequences do specific actions have? Will opening a box reveal a gem, or will it set off an explosion? Will pressing a switch start a machine? Will magic words transport the character into a new room? Consequences could include appearances and disappearances, changes in the character's abilities, alteration of the paths between rooms, and transportation from one location to another.

Some actions may have special consequences only under restricted circumstances. A special tool may be needed, such as a crowbar to open a crate. If this tool isn't in the character's inventory, the action won't have the desired effect and might even backfire.

Things may happen independently of the player's actions as well. A troll might be wandering around the adventure's world. Or the character's lamp might go out after a certain number of moves.

When you've considered all of these things, and made your choices, you know what you want the adventure to do. Only now should you worry about the details of the program. As you discover what's easy to program and what isn't, you might change your mind about what features to include. But just like programming any other game, you should start with an overall plan. It will save you countless hours of wasted time later on.

Assigning Variables

Now you're ready to actually begin programming your text adventure game. We'll go through the process step by step, outlining and illustrating exactly how to do it.

The first step is to assign variables to the important parameters of the adventure. It's easier to remember what these variables mean than it is to recall a number; using these

variables also makes it simple to alter the program if you later decide to change the parameters.

One of the first statements of the program, even before the DIM statements, should look something like this:

```
1Ø NR=21:NV=14:NO=16:NI=1Ø:ND=6
```

NR is the Number of Rooms, NV the Number of Verbs, NO the Number of Objects, NI the Number of Items, and ND the Number of different Directions the character can move in. (Note: An *object* is any word that can be used as the second word of a command, whether it corresponds to a physical object or not. An *item* is an object which is located in a room; it usually designates a physical object.)

Adventure Arrays

Next you need to translate the layout of your adventure into a set of data structures. Let's look at each of the required structures and the purpose it serves.

Access array. This is the translation of your map into terms the computer can understand. It's defined by the statement:

DIM AC(NR,ND)

To use the access array (AC), the directions in which the character moves must be translated into numbers. Let's assume the following translation:

North = 1	South = 2
East = 3	West = 4
Up = 5	Down = 6

The value of AC(R,D) specifies which room is reached by going in direction D from room R. If this value is 0, it means the character can't go that *way* from that *room*.

Room description array. This array is defined by:

DIM RD$(NR)

Each of its entries is a string which gives the description of the room; for example, "You are standing on a wide bridge."

Room flag array. Flags are indicators of whether a condition is true or false. A flag takes only one bit, so you can have up to 16 different flags in the room flag array. The array is defined by:

DIM RF(NR)

The different flags are defined as powers of two—1 might indicate that the room is too cold; 2, that magic works; and 4, that water is present. The value of RF(R) for room R consists of the logical OR of all the flag values that are true for that room. If a room is cold and allows magic but doesn't have any water, its entry in the array would be a 3 (1 OR 2).

Verb array. This is an array of the possible first words of commands, defined by:

```
DIM VB$(NV)
```

You should decide how many letters in a word are going to be significant and chop the verbs in this array down to that size. For instance, if two letters are significant, JUMP must be stored as JU. It's a good idea to limit the number of significant letters, so that two-fingered typists have less work to do. Many simple adventure games designate only two letters as significant.

Object array. This is an array of the possible second words of commands (their objects) and is defined by:

```
DIM OB$(NO)
```

Once again, all words in this array should contain only as many letters as are significant.

Verb token array. This serves to translate verbs into numbers. It is dimensioned by:

```
DIM VT(NV)
```

The entries in this array correspond to entries in the verb array. The values stored consist of numbers from 1 to the *number of distinguishable verbs* in the game. This number is normally smaller than NV, since synonyms such as GET and TAKE, or NORTH and N, are not distinguishable. If VB$(2)="N" and VB$(3)="NORTH", then VT(2) and VT(3) will have the same value. This lets the program be indifferent to which word was actually typed.

Object token array. This array translates the second word of a command into a number. It is defined by:

```
DIM OT(NO)
```

Its elements correspond to the object array. However, the elements can be a little trickier than the verb token array's elements. Remember that not all *objects* are *items*. It's convenient to have the object tokens fall into two series. Items, which are

objects that have a particular location, can be numbered from 1 to NI. Other objects, including directions and magic words, can be numbered starting with 101. This makes it easy to add new items without disrupting your numbering system.

Item description array. This contains a text description for each item. Its definition is:

```
DIM ID$(NI)
```

The text description of an item could be the same as the word in the object array for it, but often is a little more. For instance, the object array might have the word LAMP for an object described in the item description array as "Old oil lamp."

Item location array. This locates each item and is defined by:

```
DIM IL(NI)
```

There are three possibilities for where an item is located. It could be in a room, in the character's inventory, or nowhere at all. The third case indicates an item that's been destroyed or one that's not yet available. A positive number in the item location array indicates which room the item is in. A zero says that the character is carrying the item. A negative specifies that the item isn't to be found.

Item flag array. This is similar to the room flag array in concept, except that it specifies conditions that are true or false for items rather than rooms. It is defined by:

```
DIM FI(NI)
```

(It would make sense to call the array IF, but that's a reserved word in BASIC.) Specific bits in the elements of the array are used to indicate such properties as whether or not the item can be carried.

More Variables

Finally you'll need to set a few more variables, such as those listed here:

- **VB** Verb token obtained from the last command entered.
- **OB** Object token obtained from the last command. It can be 0 if only one word was typed.
- **RM** Room the character occupies.
- **NC** Number of items the character is carrying.
- **MI** Maximum number of items the character can carry. NC may never exceed MI.

MC Move counter. This indicates how many moves have occurred since the adventure started. It can serve as a timer for various events.

DF Description request flag. This variable is set to 0 after the current room is described to the player. If a description is required before the next move (because the character went into a new room or decided to LOOK around again), it's set to 1 to get a display of the description. Leaving it at 0 saves having the same description repeated every move.

Specific situations will undoubtedly call for a few more variables, but the arrays and variables listed here will provide the major part of what a simple adventure needs.

The Main Loop

An adventure program consists of two parts: the initialization and the main loop. The initialization section includes DIMensioning arrays and setting up data. We've already looked at some of the initialization section of our example adventure. It uses READ and DATA statements to set up all the initial values. Once the initialization is done, however, the main loop takes over. It runs until the game is completed. The overall flow of the main loop would be something like that shown in Figure 2. The major portions of the main loop, as shown, are the room description, the automatic routines, the command input and parsing, and the action routines. Let's consider how to program each of these in turn.

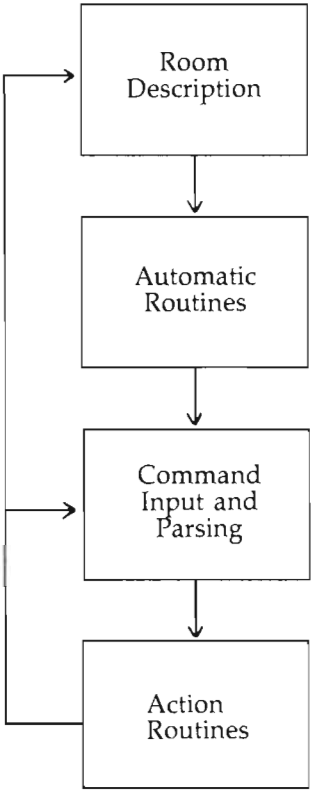
Room Description

Whenever the surroundings change, the character moves into a new room, or the player asks to LOOK at the room again, the room description routine provides the appropriate information. There are two things to be described: the room itself, and whatever items it contains.

Such routines are not long and could look like this:

```
400 IF DF=0 THEN 600
410 PRINT RD$(RM)
420 F=0
430 FOR I=1 TO NI
440 IF IL(I)<>RM THEN 490
450 IF F=0 THEN PRINT "YOU SEE:":F=1
460 PRINT ID$(I)
490 NEXT I
```

Figure 2. The Main Loop



The description request flag in line 400 determines whether this section of the program is executed or skipped over. Remember that 0 indicates the latter. If it is 0, this entire routine is bypassed. If the routine *is* executed, describing the room consists simply of printing the appropriate element of the room description array. That's line 410. Next, a FOR-NEXT loop in line 430 goes through each item in the item location array. For each item that's located in the current room (F=0), it prints the corresponding element of the item description array (lines 450 and 460). That way, the player will see what each room contains.

Automatic Routines

The next section of the main loop takes care of events that aren't directly caused by the player's commands. We can call these routines *automatic*, for they happen independently of what's typed in. An adventure can be written without any automatic routines, but having even a few things outside the player's control gives a much greater sense of realism and excitement.

Automatic routines can be controlled by the move counter, random numbers, or a combination of the two. The commands the player gives can have an effect as well. A passage may close four turns after the character enters a room, or a wraith may start stalking the character only after he's touched a crypt. Extra variables can be used to indicate the move on which something will happen. In the following example routine, MM is a variable indicating the move in which a wall collapses, opening a new passage between rooms 8 and 9.

```
700 MC=MC+1
710 IF MC<>MM THEN 800
720 AC(8,3) = 9: AC(9,4) = 8
730 IF RM=8 THEN PRINT "THE EASTERN";
740 IF RM=9 THEN PRINT "THE WESTERN";
750 IF RM=8 OR RM=9 THEN PRINT " WALL COLLAPSES, O
    PENING A NEW PASSAGE."
```

MC is the move counter. Each time through the main loop, it's incremented by 1 in line 700. Assuming you have previously set MM to the desired turn number (say 8), this automatic routine would not be executed until MC equals MM (in other words, on turn 8). Line 710 insures this. Line 720 actually creates the opening between the rooms. The message then displays, specifying which wall has crumbled. If the character is in room 8, for instance, the eastern wall has fallen, and the character can now move in that direction.

The position of automatic routines within the program is important. Usually they should come *after* the room description, so that the player finds out where his or her character is before being told what happens. Some automatic routines are better placed after the player has completed the move, though. This conveys the sense that what happened immediately followed the move. For instance, if a flock of bats carries the

character out of a room every time he tries to enter, the player may not even see the room until it's discovered how to remove the bats.

Command INPUT and Parsing

At this point the program stops talking to the player; instead, it's the player's turn to communicate with the program. To do this, the program must accept a command and *parse* it. To parse a command simply means to break it up into its components and identify their relationships—an easy job when it consists of just two words.

Here's the first section of an INPUT and parsing routine.

```
1000 INPUT C$
1010 L=LEN(C$):IF L=0 THEN 1000
1020 C1$="":C2$="":C2=0:X=0
1030 FOR I=1 TO L
1040 A$=MID$(C$,I,1)
1050 IF A$<>" " THEN 1080
1060 IF C2$<>"" THEN 1200
1070 X=1:GOTO 1090
1080 IF X=0 THEN C1$=C1$+A$:GOTO 1090
1085 C2$=C2$+A$
1090 NEXT I
```

The program receives a command through the INPUT statement. As the player enters words, a string is created. Then the program separates the two words by looking for one or more spaces between them. (It's best that it be tolerant of more than one space between words, as well as spaces after the command. INPUT automatically strips leading spaces, so they don't pose a problem.) The following program section receives the player's INPUT (line 1000) and creates two strings, C1$ and C2$ (lines 1080 and 1085). Spaces between words are also checked for in line 1050.

The following lines continue the routine:

```
1200 C1$=LEFT$(C1$,6): C2$=LEFT$(C2$,6)
1210 FOR I=1 TO NV
1220 IF VB$(I)=C1$ THEN VB=VT(I):GOTO 1250
1230 NEXT I
1240 PRINT "I DON'T KNOW THE VERB ";C1$:GOTO 1000
1250 IF C2$="" THEN OB=0:GOTO 1400
1255 FOR I=1 TO NO
1260 IF OB$(I)=C2$ THEN OB=OT(I):GOTO 1400
```

```
1270 NEXT I
1280 PRINT "I DON'T KNOW THE OBJECT ";C2$:GOTO 100
     0
```

The two strings, C1$ and C2$, are the first and second words of the command. The next step is to translate these strings into the verb token and the object token. This means looking them up in the verb array and object array, and getting the corresponding elements of the verb token array and object token array. Lines 1220 and 1240 in the section of the routine below do this for the verb and object respectively. Note the checks and messages displayed if the verb or object does not exist in the appropriate array.

The two strings must be truncated to the number of significant characters in order to match the strings in the arrays. Line 1200 assumes truncation to six characters.

In the case of a one-word command, C2$ will be the empty string, so the object token will be set to 0 (line 1250).

Action Routines

Once the program has the command in the form of the verb token and the object token, it's ready to determine what those commands will do. The part of the program that does this is called the *action routines*. This will be the largest portion of the program. However, it consists of a lot of small pieces, so it isn't very difficult to write.

Before figuring out what a specific verb does, the program should do some general checking on whether the object is reasonable. If the object is an item, it has to be either in the room or in the character's inventory. If it's somewhere else, the character can't do anything with it. If the object isn't an item, only a few verbs will work with it, so the program should make sure that the verb is an appropriate one. NORTH, for example, isn't something the character can TAKE, EAT, or OPEN. Only GO makes sense.

In a language that was more generous with names than BASIC, we could assign a variable name to each verb. Trying to think of a two-letter name for each verb that would mean anything, though, is a hopeless exercise. So at this point we resign ourselves to using numbers.

The following routine assumes that the direction object tokens (NORTH, UP, etc.) are numbers 101 to 106, that GO is verb 10, that SHAZAM is object 107, and that SAY is verb 12.

```
1400 IF OB<100 THEN 1600
1405 REM IT'S NOT AN ITEM
1410 IF OB<=106 AND VB<>10 THEN 8000
1420 IF OB=107 AND VB<>12 THEN 8000
1430 GOTO 2000
1599 REM IT IS AN ITEM
1600 IF IL(OB)<>RM AND IL(OB)<>0 THEN PRINT "IT IS
     N'T HERE.":GOTO 1000
8000 PRINT "THAT'S SILLY!":GOTO 1000
```

Line 1400 checks to see if it's an item (with an object token less than 100). If it is, the program jumps to line 1600, where it's determined whether the item is in the room or in the character's inventory. If it's neither, the message IT ISN'T HERE displays. The program chides the player with THAT'S SILLY if a direction (NORTH, UP, etc.) is requested and GO isn't used with it. The player will also see the message if something like SAY (VB=12) SHAZAM (OB=107) is typed in.

Notice that if the command is rejected, the program goes back to the command INPUT (through the GOTO 1000 statements in lines 1600 and 8000), rather than letting anything happen automatically.

If these checks turn up no problems, the program falls through to the action routine for the specific verb. The tool used is the GOTO statement found in line 1430, which sends the program to the ON-GOTO routine beginning at line 2000, as shown below:

```
2000 ON VB GOTO 3000,3100,3200,3300,3400,3500,3600
     ,3700
2010 ON VB-8 GOTO 3800,3900,4000,4100,4200,4300,44
     00,4500
```

Several of these statements will usually be necessary, because of line length limitations. Remember that an ON statement will simply fall through to the next statement if the variable is out of range. Thus, if the variable is 9, it falls through line 2000 to line 2010, where it would access the first line listed, 3800 (9−8=1). Using this technique, we can call up to 16 different verb routines in the above example.

Each of the line numbers in lines 2000 and 2010 is the start of the action routine for a particular verb.

Using Verbs

Certain verbs will be standard in most adventures, so they can be discussed in some detail here. Others will have effects that are peculiar to the situation. They're the ones that make your adventure unique. Once you've seen how the standard verbs work, though, you shouldn't have much trouble adding your own special ones.

Directional verbs and GO. There are two ways a player might specify movement in a given direction. Either a simple direction (for instance, EAST or just E), or GO and a direction (GO EAST) could be entered. It isn't much trouble to include both. A common area of the program can be used to handle all directional movement, using a direction variable which the specific commands set before accessing the actual movement.

For a one-word command, the direction is the verb. In that case, it sets the direction variable and goes to the common routine. The line below illustrates the one-word command NORTH.

```
3100 D=1:GOTO 3620
```

You'll recall that we decided to use 1 as the directional number for NORTH. All that's done in the above line is to set D (the directional variable) to 1 and then GOTO a line which checks to see if that direction leads anywhere. More on that in a moment.

The GO command has to translate its object into a direction before going to the common routine. It's easy to do this if the direction objects are numbered appropriately, so that subtracting a number from the object token gives the right index into the access array. Take a look at the following lines:

```
3700 IF OB<=100 OR OB>106 THEN 8000
3710 D=OB-100:GOTO 3620
```

Notice that if the object (OB) is *not* a direction (checked for in line 3700), the program jumps to line 8000, where the message THAT'S SILLY! is printed. The direction variable D is set in line 3710. If OB equals 101, for instance, signifying that the direction is NORTH, D equals 1. The program then moves to line 3620.

The common routine uses the access array to determine where the move will take the character. This next segment is this common routine used by both one- and two-word commands.

```
3620 IF AC(RM,D)=0 THEN PRINT "YOU CAN'T GO THAT W
     AY.":GOTO 400
3630 RM=AC(RM,D):DF=1:GOTO 400
```

A value of 0, as mentioned before, means that a given direction doesn't lead anywhere. If the command does take the character somewhere, the description request flag is set to 1 so that the player can see the new room. Both of the lines above take the program back to the routine which describes the room.

TAKE. This command transfers, or attempts to transfer, an item from the current room to the character's inventory. The program has to determine whether the item can be picked up, and whether it can be carried. The character might already be carrying as much as allowable. Taking an item might also have side effects, like making another item visible or setting off a trap. The program doesn't have to check whether the object is in the room, since that has already been determined. However, it *does* have to check whether the character is already carrying the item. Take a look at the lines below to see how that can be programmed.

```
4200 IF (FI(OB) AND CF)=0 THEN PRINT "YOU CAN'T PI
     CK THAT UP.":GOTO 400
4210 IF IL(OB)=0 THEN PRINT "YOU ALREADY HAVE IT!"
     :GOTO 400
4220 IF IC=5 THEN PRINT "YOU'RE CARRYING TOO MUCH
     {SPACE}ALREADY.":GOTO 400
4230 IL(OB)=0:IC=IC+1:PRINT "TAKEN."
4240 REM SIDE EFFECTS GO HERE
4290 GOTO 400
```

This assumes that flag CF (in line 4200) in the item flag array indicates whether an item can be taken. If your character already has the item, line 4210 prints a message to that effect. Note that a limit of five items is set in line 4220. If IC (the variable keeping track of the number of items carried) equals five, the character can't take anything else. Line 4230 actually TAKEs the item by placing it in the character's inventory (IL(OB)=0), increments the number of items held, and prints a message that the TAKE was successful.

DROP. The reverse of TAKE, it's even simpler, since an item which is being carried can normally be dropped.

```
4300 IF OB=0 THEN PRINT "DROP WHAT?":GOTO 1000
4310 IF IL(OB)<>0 THEN PRINT "YOU DON'T HAVE IT!":
     GOTO 400
4320 IL(OB)=RM:PRINT "DROPPED."
4330 IC=IC-1
4390 GOTO 400
```

The only question is whether the item is in the character's inventory, which is checked in line 4310. The object is transferred to the room (line 4320), and the inventory count is decremented (line 4330). Again, side effects are possible.

INVENTORY. All this command does is list the items the character is carrying. This involves going through all the items and listing the ones that have a location of 0.

```
4400 PRINT "YOU ARE CARRYING:"
4410 FOR I=1 TO NI
4420 IF IL(I)=0 THEN PRINT ID$(I)
4430 NEXT I
4440 IF IC=0 THEN PRINT "NOTHING."
4450 GOTO 400
```

Line 4420 PRINTs the items the character is carrying. If IC (the number of items carried) is 0, a message indicating that the character holds nothing is displayed.

LOOK. This is one of the simplest commands; it just sets the description request flag with a line such as:

```
4500 DF=1:GOTO 400
```

QUIT. Even simpler, except that it's nice to make sure the player really means it.

```
4600 PRINT "DO YOU REALLY WANT TO QUIT";
4610 INPUT Y$
4620 IF LEFT$(Y$,1)<>"Y" THEN 1000
4630 END
```

Unusual Commands

Other verbs vary from one adventure game to another. EXAMINE can give you additional information about items. FEEL, SMELL, and TOUCH, might serve a similar purpose. The process of examination might also cause other, previously hidden, items to appear. OPEN could be another way to reveal a hidden item. Words like CUT and BURN might have interesting effects on items, but unless an appropriate tool is in

the character's inventory, these commands would simply return a message like "You can't do that."

Having a few commands that do nothing but return a standard response is useful, just because it adds to the number of commands that get an interesting answer without adding much to the programming effort. For instance, the verb BREAK with any object might get the response "Vandalism won't help your situation." This will also leave the player wondering whether there's some object that *could* be broken for a useful result.

Commands like CLIMB or ENTER might work on certain objects to provide a way of getting from one room to another. Avoid using GO for this, for in spite of what some adventure game programmers think, you don't *go a door.*

Other commands might surprise the player by transporting the character from one place to another. For instance, taking an item might cause a trap door to open, dropping the character into the room below. Magic words can also serve this purpose. A magic word may be restricted in its use to a certain room, so it provides passage only from that room to another.

What Goes into It?

The mechanics of writing an adventure program are only part of the job, just as grammar and spelling are only part of what goes into writing. The other part is what you actually have to say. Creating the content of an adventure can't be reduced to a cookbook approach. Still, some general guidelines are possible.

Quests and hunts. There are two basic types of adventure: the quest and the treasure hunt. In a quest adventure, you're given a particular goal to achieve, such as solving a mystery or obtaining a single treasure. In a treasure hunt, you're trying to find as many treasures as possible, to get a high score.

The quest adventure is an all-or-nothing proposition. The program can give you a score to indicate how close you've come to success, but you probably won't be satisfied until you solve it. The treasure hunt offers more satisfaction to the beginning adventurer, since if even a few treasures are found, there's a sense of accomplishment. If a quest is like climbing a

mountain, a treasure hunt can be compared to hiking across a series of low hills. Each one has its own kind of satisfaction.

Make the pieces fit. In either case, all the pieces should fit together. This is more obvious for a quest—each step is part of a developing story. Even in a treasure hunt, though, everything should be set against a common background and story line. If it's set in a world of Greek mythology, Woden and Brunhilde shouldn't appear. If you've chosen a science-fiction setting, it shouldn't have magical elements that don't fit. Humorous events can certainly liven an adventure, but they shouldn't be jarringly out of place.

The puzzles should be interrelated. Otherwise, what you end up with is a series of small puzzles rather than one complete adventure. Solving one puzzle should provide a tool that's needed for solving the next one. The various items required should be scattered around so that the character has to go back and forth among the rooms, rather than having everything too neatly at hand.

Don't cheat. The puzzles should always be logical. The solution should make sense, at least once the player has stumbled upon it. A puzzle that reduces the player to trying actions at random has failed. If the way to summon a genie in your adventure is to kiss a coconut, be sure to provide some clue that will suggest that action. If you don't, you'll have a hard time getting people to play your second adventure.

Traps should not be sprung unexpectedly. It should be possible for the player to get a hint of danger ahead before walking into it, perhaps by requiring the player to examine things carefully. This doesn't mean that everything should be so easy that a player can solve it the first time. It means that at the end of the puzzle or game, the player sees the program was "playing fair." One adventure game, for example, requires the character to crawl through a passage to survive, yet there was no indication that the passage was dangerous. This forces the player to rely on knowledge gained in a "previous life," something not as realistic as many players would like.

Just as when you create any game, the art of text adventure writing is much like the art of storytelling. To keep the player interested, interesting things have to happen. One event should follow reasonably from another, leading to a climax. Because it is a form of storytelling, the text adventure

offers you, the author, a chance to express yourself, something not often found in other computer games. When you write an adventure, you're doing more than creating a game. You're creating a world.

2
Arcade-Style Games

Arcade-Style Games

Of the many things that make computer games so much fun, one of the most important has got to be the computer's speed. Based on program or player instructions, your VIC can update its world view thousands of times a second—and in few areas is that speed better used than in a well-designed arcade game.

The programs in this chapter leave no doubt about the power of your VIC as an arcade-style game machine. For example, "Hardhat Climber," by Chris Lesher, offers high adventure on a maze of steel girders, where you must recover misplaced toolboxes while dodging runaway barrels full of nails. If your tastes are more down-to-earth, David Lacey's "The Frantic Fisherman" offers equally exciting sea-level shenanigans as a quiet vacation at the lake turns into a battle with marauding sharks and threatening rainstorms.

If high finance is your forte, "Gotcha!," by Doug Smoak, lets you run up big scores (and bank accounts) by grabbing all the money you can while dodging the Collector.

Phil Callister's "Wheeler" lets you follow the path of a lone tire, bouncing along a roadway strewn with scissors, nails, rocks, and potholes. Its fate is at your fingertips, and your hands will be full assuring its survival. If you prefer four tires and a steering wheel, you'll find Steve Elder's "Freeway Zapper" equally challenging. Interstate driving was never so much fun!

In "Worm of Bemer," by Stephen D. Fultz, be ready with quick reflexes to guide Nerm the Worm through the catacombs of Bemer Castle, helping him eat magic mushrooms along the way. And after a few rounds, you'll be ready to uphold the kingdom's honor in "Olympiad," by Kevin Woram and Mike Buhidar, Jr.

There was a time when games of such quality were available only at commercial arcades. Now you have a fine collection right at your fingertips.

Hardhat Climber

Chris Lesher

"Hardhat Climber" is one of the best games we've seen for the unexpanded VIC-20. It is an excellent example of what can be accomplished with BASIC.

You are standing at the bottom of four levels of girders, connected by ladders. At the top is a pile of 12 barrels, and scattered along the girders are toolboxes. The object of "Hardhat Climber" is to walk around the girders and pick up every toolbox while avoiding the barrels that roll down at you. If you pick up all of the toolboxes, you are rewarded with bonus points, and you move on to a more difficult screen.

I wrote the VIC-20 version of Hardhat Climber almost entirely in BASIC, with only a short machine language routine to check the joystick. Using the stick, you can move the climber up, down, left, and right along the girders and ladders. Pressing the fire button makes your climber jump in the direction he was last moving, allowing him to jump over barrels and holes in the girders.

Scoring

You score 150 points for every toolbox you pick up, 1000 points for jumping over a barrel, and 100 points for each barrel remaining after you have picked up all the toolboxes. The score is displayed in the upper-left corner of the screen, while the screen number is shown in the upper-right corner. The number of climbers remaining is displayed between the score and screen number.

You begin the game with three climbers and earn an extra one every 10,000 points. A climber is lost if he is hit by a barrel, walks off a girder, or has not picked up all the toolboxes by the time all 12 barrels have rolled off the pile. The game ends when you lose your last climber.

Several of the program lines are longer than the maximum limit of 88 characters. They must be entered by abbreviating the keywords and omitting the space between the line number and first keyword. Abbreviations may be found in Appendix D of *Personal Computing on the VIC-20,* which came with your computer. If you make an error while entering any

of these lines, the entire line must be retyped using the abbreviations. Remember, when you use "The Automatic Proofreader" while entering lines with abbreviations, the checksum numbers will not match up. To use the checksum, LIST the line, then place the cursor over the line and press return. The checksum will now be correct for that line.

Loading the Programs

Note the variable C in line 1. To save the program to disk, C should equal 8; to save to cassette, change C to 1.

In either case, type in and SAVE Program 1 as "HAT.1". Be sure that you save it before running it. Then type in and SAVE Program 2 as "HAT.2". You must use these filenames, even if you are saving to cassette. To play the game, LOAD and RUN "HAT.1". It will automatically load "HAT.2". Finally, type RUN (and press RETURN) to play.

Program 1. Hardhat Climber, Part One

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 PRINT"{CLR}{3 RIGHT}PLEASE WAIT":C=8:REM LET C=1
   FOR TAPE                                    :rem 146
10 FORN=7616TO7679:READM:POKEN,M:NEXT          :rem 115
11 FORN=828TO899:READM:POKEN,M:NEXT             :rem 15
90 POKE 51,192:POKE 52,29:POKE 55,192:POKE 56,29
                                                :rem 51
100 PRINT"{3 DOWN}LOAD";CHR$(34);"HAT.2";CHR$(34);
    CHR$(44);C;"{HOME}{DOWN}":POKE 631,13:POKE 198
    ,1                                          :rem 106
153 DATA255,255,153,102,102,153,255,255,195,255,25
    5,195,195,255,255,195,60,60,25,255          :rem 122
155 DATA188,60,36,231,3,4,24,24,60,126,126,60,60,6
    6,165,153,153,165,66,60,,24,36              :rem 145
157 DATA126,126,126,126,,,,,,,,,,,,,,,,,,,,,,,,  :rem 139
159 DATA169,,133,1,169,255,141,34,145,169,32,44,31
    ,145,208,5,169,1,133,1,96,169,8,44          :rem 106
161 DATA31,145,208,5,169,2,133,1,96,169,16,44,31,1
    45,208,5,169,3,133,1,96,169,4,44,31         :rem 142
163 DATA145,208,3,133,1,96,169,127,141,34,145,169,
    128,44,32,145,208,4,169,5,133,1,96          :rem 103
```

Program 2. Hardhat Climber, Part Two

```
1 D=37154:P1=37151:P2=37152:DO(0)=-1:DO(1)=1:DI=DO
  (INT(RND(1)*2))                                    :rem 54
3 A$=">>>>>>>>>>>>>>>>>>>>{LEFT}{INST}>":DIMB(11)
  :G=30720:E2=0:K=8164:I=22                          :rem 56
5 SC=1:CH=2:E1=0:D(0)=4:D(1)=2:D(4)=7:Z=57:E3=1:Q=
  10000:J=56                                         :rem 34
7 FOR N=0 TO 11:READB(N):NEXT:POKE 36879,25
                                                     :rem 155
9 POKE 36878,15:POKE36869,255:E4=0:E5=2              :rem 142
11 GOSUB99:H=0:Y=0                                   :rem 67
13 S=8143+INT(RND(1)*20):IFPEEK(S+22)=62ORPEEK(S)=
  59THEN13                                           :rem 139
15 T=PEEK(S):POKES,58:POKES+G,0                      :rem 159
17 V=7712+B(Y):W=62:DO=DO(INT(RND(1)*2))              :rem 5
19 SYS828:ONPEEK(1)GOTO41,23,27,33,37                :rem 117
21 FORN=1TO23:NEXT:GOTO51                            :rem 96
23 IFPEEK(S+22)=ZTHENPOKES,T:POKES+G,D(T-J):S=S+22
  :GOTO49                                            :rem 90
25 GOTO51                                            :rem 6
27 DI=-1:IFPEEK(S+21)<62THENPOKES,T:POKES+G,D(T-J)
  :S=S-1:GOTO49                                      :rem 155
29 IFT<>ZTHENPOKES,T:POKES+G,D(T-J):S=S+DI:T=PEEK(
  S):GOTO71                                          :rem 81
31 GOTO51                                            :rem 3
33 IFT=ZTHENPOKES,T:POKES+G,D(T-J):S=S-22:GOTO49
                                                     :rem 89
35 GOTO51                                            :rem 7
37 DI=1:IFPEEK(S+23)<62THENPOKES,T:POKES+G,D(T-J):
  S=S+1:GOTO49                                       :rem 111
39 GOTO29                                            :rem 16
41 POKE36876,240:POKES,T:POKES+G,D(T-J):S=S-22+DI:
  T=PEEK(S):POKES,58:IFT=60THEN71                    :rem 0
43 IFPEEK(S+22)=60THENSS=SS+1000:PRINT"{HOME}{RVS}
  "TAB(8-LEN(STR$(SS)))SS                            :rem 60
45 FORN=1TO5:NEXT:POKES,T:POKES+G,D(T-J):S=S+22+DI
  :T=PEEK(S):POKES,58                                :rem 160
46 IFPEEK(S+22)>61THEN71                             :rem 141
47 POKES+G,0:POKE36876,0:GOTO51                      :rem 103
49 POKE36876,200:POKE36876,0:T=PEEK(S):POKES,58:PO
  KES+G,0                                            :rem 174
51 IFT=61THENSS=SS+150:PRINT"{HOME}{RVS}"TAB(8-LEN
  (STR$(SS)))SS:H=H+1:T=62:IFH=16THEN87              :rem 246
53 IFT=60THEN71                                      :rem 133
55 GOSUB137                                          :rem 133
57 POKEV,W:POKEV+G,D(W-J):V=V+DO:W=PEEK(V):POKEV,6
  0:POKEV+G,7                                        :rem 148
59 IFDO=22ANDPEEK(V+22)=56THENDO=DO(INT(RND(1)*2))
  :GOTO63                                            :rem 196
```

```
61 IFW=63THENDO=22                                    :rem 86
63 IFW=58THEN71                                       :rem 144
65 IFV<8164THEN19                                     :rem 248
67 Y=Y+1:IFY=12THEN71                                 :rem 17
69 POKEV,62:GOTO17                                    :rem 99
71 POKE36876,0:SO=250:IFT=60THENT=W                   :rem 136
73 POKE36874,SO                                       :rem 115
74 IFPEEK(S+I)<>56ANDS<KTHENPOKES,T:POKES+G,D(T-J)
   :S=S+I:T=PEEK(S):POKES,58:POKES+G,0                 :rem 184
75 FORN=1TO17:NEXT:SO=SO-5:IFSO>150THEN73              :rem 136
77 POKE36874,0:CH=CH-1:IFCH=-1THENPOKED,255:POKE36
   869,240:GOSUB141:END                                :rem 229
79 PRINT"{HOME}{RVS}"TAB(14)CH::POKEV,W:POKEV+G,D(
   W-J):Y=Y+1:IFW=58THENPOKEV,T:POKEV+G,D(T-J)
                                                       :rem 250
81 IFY>10THEN11                                       :rem 129
83 IFS>8163THENPOKES,T:GOTO13                         :rem 101
85 POKES,58:POKES+G,0:GOTO17                          :rem 237
87 IFY=11THEN93                                       :rem 145
89 FORN=Y+1TO11:POKE7712+B(N),62:SS=SS+100:PRINT"
   {HOME}{RVS}"TAB(8-LEN(STR$(SS)))SS:GOSUB137
                                                       :rem 242
91 POKE36877,250:FORM=240TO250:POKE36876,M:NEXT:PO
   KE36876,0:POKE36877,0:NEXT                          :rem 111
93 E2=E2+.05:SC=SC+1:E1=E1+1:IFE1>8THENE1=8
                                                       :rem 226
95 GOTO11                                               :rem 9
97 GOTO97                                              :rem 25
99 PRINT"{CLR}{PUR}";:FORN=1TO21:PRINTA$:NEXT:PRIN
   TA$"{HOME}":B$=">888888888888888888"                :rem 119
101 PRINT"{2 DOWN}"TAB(6)"?>>>>>>>>>?{RED}":PRINTTA
    B(6)"9{PUR}88888888{RED}9":PRINTTAB(6)"9>>>>>>
    >>9"                                               :rem 207
102 PRINTTAB(6)"9>>>>>>>>9{PUR}"                       :rem 192
103 FORN=1TO3:PRINTB$"{4 DOWN}":NEXT:PRINTB$"
    {HOME}";:POKE8185,62                               :rem 205
105 PRINT"{RVS}"TAB(8-LEN(STR$(SS)))SS;TAB(14)CH;T
    AB(17)SC                                           :rem 164
106 POKE7697,163:FORN=0TO11:POKE7712+B(N),60
                                                       :rem 34
107 POKE7712+B(N)+G,7:NEXT:FORN=7834TO8164STEP110:
    IFN=8164THEN119                                    :rem 169
109 FORO=1TO3                                          :rem 20
111 R=N+1+INT(RND(1)*20):IFPEEK(R)<>56THEN111
                                                       :rem 93
113 FORM=RTOR+88STEP22:POKEM,57:POKEM+G,2:NEXT
                                                       :rem 213
114 IFO>1ANDRND(1)<E2THENPOKER+(INT(RND(1)*2)+2)*2
    2,63                                               :rem 121
```

```
115 IFRND(1)<.5ANDPEEK(R-22)=62THENPOKER-22,63
                                                :rem 210
117 NEXT                                        :rem 216
119 FORO=1TO1                                    :rem 88
121 R=N+3+INT(RND(1)*16)                         :rem 45
122 IFPEEK(R)<>56ORPEEK(R-22)<>62ORPEEK(R+1)=62ORP
    EEK(R-1)=62THEN125                           :rem 74
123 POKER,62:POKER-22,63                        :rem 140
125 NEXT                                        :rem 215
127 FORO=1TO4                                    :rem 21
129 R=N-21+INT(RND(1)*20):IFPEEK(R)<>62ORPEEK(R+22
    )=62THEN129                                  :rem 61
131 POKER,61:POKER+G,0:NEXT:NEXT                 :rem 36
133 POKE7710,63:POKE7715,63:POKE7731,63:POKE7738,6
    3                                           :rem 163
135 FORN=7812TO8142STEP110:POKEN,63:NEXT:FORN=7833
    TO8163STEP110:POKEN,63:NEXT:RETURN          :rem 133
137 IFSS>=Q*E3THENCH=CH+1:E3=E3+1:PRINT"{HOME}
    {RVS}"TAB(14)CH                              :rem 39
139 RETURN                                      :rem 125
141 POKE 36879,105:PRINT"{CLR}{WHT}YOUR FINAL SCOR
    E WAS ";SS                                   :rem 60
143 PRINT"{3 DOWN}PLAY AGAIN?(Y/N)              :rem 118
145 GET R$:IF R$="" THEN 145                    :rem 121
147 IF R$="Y" THEN RUN                          :rem 159
149 IF R$="N" THEN RETURN                       :rem 129
151 GOTO 145                                    :rem 106
153 DATA ,1,21,22,23,24,42,43,44,45,46,47 :rem 205
```

Worm of Bemer

Stephen D. Fultz
VIC Translation by Kevin Martin

Nerm the Worm is lost in Bemer Castle and needs your help to get home. You must guide him through 11 rooms and help him find magic mushrooms to eat along the way. The journey is a navigator's nightmare, because you never know where the next mushroom will grow, and if Nerm hits a wall or gets trapped by his tail, he loses one of his lives. For the VIC with at least 8K expansion; a joystick is required.

"Worm of Bemer" is a fast-paced arcade game in which Nerm the Worm travels through rooms eating magic mushrooms. Nerm is lost in Bemer Castle and wants to return home. Guide Nerm to each mushroom, as it appears, so he can keep up his strength for the journey.

After eating five mushrooms in a room, Nerm can exit to the next room. You must guide Nerm through 11 rooms before he finds his home.

Nerm starts out with four lives. If he touches anything besides a mushroom, he will lose one of his lives.

At the top of the screen will be the current score, what room Nerm is in, how many mushrooms Nerm must eat to open the exits, and how many lives Nerm has left, including the current life. You get 100 points, plus bonus points depending on the level, for every mushroom Nerm eats. Nerm gets a bonus life after completing the first two rooms and another for every third room thereafter.

Getting Nerm Home

Here are a couple of hints to help you play Worm of Bemer. First of all, try to leave room between the walls and Nerm's tail. If you block off the exits with Nerm's tail, you'll have trouble getting to the next room. Sometimes it will seem as if you've surrounded the mushroom with Nerm's tail; then all you have to do is move Nerm to another section of the screen, and wait for the tail to shorten enough so that you can get to the mushroom.

Typing In the Program

This is a two-part program requiring at least 8K additional memory. If you are saving to tape, first type in and SAVE Program 1, after deleting lines 10 and 40 and removing the REM from line 11. Then type in Program 2 and SAVE it right after Program 1 on the tape.

If you are saving to disk, type in and SAVE Program 1 just as given. Then type in Program 2 and SAVE it as "NM".

To load from tape, LOAD Program 1 and RUN it. Program 1 will automatically load and run Program 2. To load Worm of Bemer from disk, LOAD and RUN Program 1. It will automatically load Program 2, placing the cursor over the RUN command. Then, when the disk drive stops spinning, press RETURN (to execute the RUN command) to start the game.

Adding More Features

Since Worm of Bemer is written entirely in BASIC, it's relatively easy to modify. You can learn a lot about programming and games by changing the actions and settings of published programs such as Worm of Bemer. Some features you might add include a routine to save the high score to disk, adding more players, or having Nerm go to a different room depending on which exit he takes. Simpler enhancements would be changing the number of mushrooms that Nerm must eat to open the exits, or changing his speed.

Program 1. Worm of Bemer, Part One

For error free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
10 POKE631,13:POKE198,1                          :rem 34
11 REM POKE631,131:POKE198,1                     :rem 56
15 REM WITH TAPE, DELETE LINES 10 AND 40, AND REMO
   VE THE REM IN LINE 11                         :rem 85
20 POKE43,1:POKE44,32:POKE8192,0                 :rem 80
30 POKE36869,240:POKE36866,150:POKE648,30        :rem 54
35 PRINT"{CLR}"                                  :rem 204
40 PRINT"{2 DOWN}LOAD";CHR$(34);"NM";CHR$(34);",8"
   :PRINT"{4 DOWN}RUN{HOME}"                     :rem 179
50 NEW                                           :rem 79
```

Program 2. Worm of Bemer, Part Two

```
0 REM THIS PROGRAM MUST BE SAVED AS "NM" TO LOAD W
  ITH THE LOADER PROGRAM                          :rem 201
1 VS=7680                                         :rem 236
2 POKE37139,0                                     :rem 196
5 POKE36879,8                                     :rem 217
10 GOTO 5000                                       :rem 95
100 GOSUB4000:FORD=1TOSP:NEXT                     :rem 103
110 IFS=7ORS=6ORS=5THENDX=1:DY=0:DI=1:IFOD=2THENDX
    =-1:DY=0:DI=2                                  :rem 18
120 IFS=11ORS=10ORS=9THENDX=-1:DY=0:DI=2:IFOD=1THE
    NDX=1:DY=0:DI=1                               :rem 108
130 IFS=14THENDY=-1:DX=0:DI=4:IFOD=3THENDI=3:DY=1:
    DX=0                                          :rem 122
140 IFS=13THENDY=1:DX=0:DI=3:IFOD=4THENDI=4:DY=-1:
    DX=0                                          :rem 123
145 PO=VS+XA+YA*22:OD=DI:POKEPO,42:POKEPO+SO,9
                                                  :rem 161
150 XA=XA+DX:YA=YA+DY:L=LEN(XA$):XA$=XA$+CHR$(XA):
    YA$=YA$+CHR$(YA)                               :rem 0
155 Z=PEEK(VS+XA+YA*22):IFZ<>32THEN200            :rem 43
162 POKE36876,150:POKE36876,0:PO=VS+XA+YA*22:POKEP
    O,36:POKEPO+SO,13:IFL<WOTHEN100               :rem 250
190 PO=VS+ASC(XA$)+22*ASC(YA$):LL=LEN(XA$)-1:XA$=R
    IGHT$(XA$,LL)                                 :rem 208
191 POKEPO,32:POKEPO+SO,0                          :rem 43
195 YA$=RIGHT$(YA$,LL):GOTO100                     :rem 19
200 POKE36876,200:FORQQ=1TO20:NEXT                 :rem 60
201 POKE36876,0:PO=VS+XA+22*YA:POKEPO,36:POKEPO+SO
    ,13:GOSUB6600:IFZ<>BUTHEN260                  :rem 133
210 WO=WO+5+3*LO:IFWO>127THENWO=127                :rem 146
220 XX=INT(RND(1)*18+2):X=INT(RND(1)*18+3):IFPEEK(
    VS+XX+22*X)<>32THEN220                        :rem 235
221 SC=SC+100+LO*7                                :rem 225
225 HI=HI-1:GOSUB6600:IFHI>0THEN229               :rem 112
226 PO=VS+11+22*2:POKEPO,160:POKEPO+SO,0:PO=VS+11+
    21*22:POKEPO,160                              :rem 198
227 POKEPO+SO,0:PO=VS+22*12:POKEPO,160:POKE36876,1
    75                                            :rem 120
228 PO=VS+22*12+21:POKEPO,160:POKEPO+SO,0:GOTO100
                                                  :rem 203
229 PO=VS+XX+X*22:POKEPO,BUG:POKEPO+SO,13         :rem 133
230 GOTO100                                        :rem 95
260 IFZ<>160ANDLI>1THENGOSUB7500:GOTO290          :rem 242
265 IFZ<>160THEN7500                              :rem 146
270 POKE36876,140                                 :rem 151
271 GOSUB7000:PRINT"{HOME}{23 DOWN}"              :rem 134
```

```
275 FORDE=1TO23:PRINT:POKE36876,DEL*2+140:NEXT:POK
    E36876,0                                    :rem 43
280 LO=LO+1:WO=5:IFLO=12THEN1200                :rem 177
282 PP=PEEK(36879):IFPP=15THENPP=10              :rem 121
283 POKE36879,PP+1                               :rem 5
285 IFLO>EXTHENGOSUB9100                         :rem 29
290 PRINT"{CLR}":GOSUB6600                       :rem 133
300 ONLO GOTO5020,400,500,550,600,700,800,450,550,
    1000,1100,1200                               :rem 176
399 GOTO5015                                     :rem 169
400 REM SECOND SCREEN                            :rem 244
410 FORI=VS+3+10*22TOVS+18+10*22:POKEI,35:POKEI+SO
    ,9:NEXT                                      :rem 192
420 GOTO5020                                     :rem 150
450 REM SCREEN                                   :rem 61
460 FORI=VS+4+10*22TOVS+17+10*22:POKEI,35:POKEI+SO
    ,9:NEXT                                      :rem 197
465 FORI=VS+10+6*22TOVS+10+20*22STEP22:POKEI,35:PO
    KEI+SO,9:NEXT                                :rem 102
470 GOTO5020                                     :rem 155
500 REM THE FOURTH SCREEN                        :rem 242
510 FORI=VS+3+5*22TOVS+18+5*22:POKEI,35:POKEI+SO,9
    :NEXT                                        :rem 105
520 FORI=VS+3+18*22TOVS+18+18*22:POKEI,35:POKEI+SO
    ,9:NEXT                                      :rem 210
530 GOTO5020                                     :rem 152
550 REM FRAME 5                                  :rem 30
560 FORI=VS+5+6*22TOVS+16+6*22:POKEI,35:POKEI+SO,9
    :NEXT                                        :rem 112
575 FORI=VS+9+7*22TOVS+9+20*22STEP22:POKEI,35:POKE
    I+SO,9:NEXT                                  :rem 25
580 GOTO5020                                     :rem 157
600 REM FRAME 6                                  :rem 27
610 FORI=VS+1+10*22TOVS+9+10*22:POKEI,35:POKEI+SO,
    9:NEXT                                       :rem 144
615 FORI=VS+13+10*22TOVS+20+10*22:POKEI,35:POKEI+S
    O,9:NEXT                                     :rem 241
620 GOTO5020                                     :rem 152
700 REM FRAME 7                                  :rem 29
710 FORJ=6TO14:FORI=VS+4+J*22TOVS+9+J*22:POKEI,35:
    POKEI+SO,9:NEXT                              :rem 76
715 FORI=VS+15+J*22TOVS+17+J*22:POKEI,35:POKEI+SO,
    9:NEXT:NEXT                                  :rem 69
720 GOTO 5020                                    :rem 153
800 REM FRAME 8                                  :rem 31
811 FORI=VS+1+8*22TOVS+10+8*22:POKEI,35:POKEI+SO,9
    :NEXT                                        :rem 105
812 FORI=VS+1+15*22TOVS+10+15*22:POKEI,35:POKEI+SO
    ,9:NEXT                                      :rem 198
```

```
813 FORI=VS+9+12*22TOVS+21+12*22:POKEI,35:POKEI+SO
    ,9:NEXT                                    :rem 203
890 GOTO5020                                   :rem 161
1000 FORJ=4TO19:FORI=VS+1+J*22TOVS+19+J*22:POKEI,3
     5:POKEI+SO,9:NEXT:NEXT                    :rem 31
1005 FORJ=4TO19:FORI=VS+1+J*22TOVS+11+J*22:POKEI,3
     2:POKEI+SO,9:NEXT:NEXT                    :rem 25
1010 GOTO5020                                  :rem 194
1100 FORJ=4TO19:FORI=VS+1+J*22TOVS+19+J*22:POKEI,3
     5:POKEI+SO,9:NEXT:NEXT                    :rem 32
1105 FORJ=4TO19:FORI=VS+1+J*22TOVS+14+J*22:POKEI,3
     2:POKEI+SO,9:NEXT:NEXT                    :rem 29
1110 GOTO400                                   :rem 144
1200 REM YOU WIN                               :rem 146
1205 FORZZ=1TO3                                :rem 167
1210 PRINT"{CLR}{6 DOWN}{5 RIGHT}NERM'S HOME"
                                               :rem 244
1212 PRINT"{5 DOWN}{6 RIGHT}THANK YOU"         :rem 13
1215 FORG=1TO5                                 :rem 61
1220 FORI=1TO10                                :rem 103
1230 POKE36876,I+130                           :rem 55
1240 NEXT:NEXT                                 :rem 127
1245 FORI=1TO127:POKE36876,I+128:NEXT          :rem 210
1250 NEXT:GOTO7700                             :rem 72
4000 POKE37154,127:S3=-((PEEK(37152)AND128)=0)::PO
     KE37154,255                               :rem 75
4010 P1=PEEK(37137):S1=-((P1AND8)=0):S2=((P1AND16)
     =0):S0=((P1AND4)=0)                       :rem 24
4020 S=JP(S2+S3+1,S0+S1+1)                     :rem 141
4030 FR=-((P1AND32)=0):RETURN                  :rem 171
5000 REM UP THE GAME                           :rem 73
5005 GOSUB10000:GOSUB11100:BUG=33              :rem 163
5011 SP=35:LI=4:SC=0:LO=1:GOSUB5500:HI=5:WO=5:EX=2
                                               :rem 40
5012 POKE36879,10                              :rem 149
5015 PRINT"{CLR}":GOSUB6500                    :rem 180
5020 XA$="":YA$="":XB$="":YB$="":XA=11:YA=19:DX=0
                                               :rem 161
5021 IFLO=3THENYA=18                           :rem 209
5025 DY=-1:T=0:IFHI<0THENHI=0                  :rem 242
5030 DI=4:IFHI>5THENHI=5                       :rem 197
5050 FORI=VS+22*2TOVS+21+22*2:POKEI,35:POKEI+19*22
     ,35:POKEI+SO,9                            :rem 35
5051 POKEI+SO+19*22,9:NEXT                     :rem 17
5055 FORI=VS+22*2TOVS+22*20STEP22:POKEI,35:POKEI+2
     1,35:POKEI+SO,9                           :rem 213
5056 POKEI+SO+21,9:NEXT:IFHI>0THEN5060         :rem 67
5057 I=VS+11+22*2:POKEI,160:POKEI+SO,0:I=VS+11+21*
     22:POKEI,160:POKEI+SO,0                   :rem 42
```

```
5058 POKEVS+12*22,160:POKEVS+12*22+SO,0    :rem 224
5059 POKEVS+12*22+21,160:POKEVS+12*22+SO+21,0:GOTO
     150                                    :rem 6
5060 XX=RND(1)*19+2:X=RND(1)*18+3:IFPEEK(VS+XX+X*2
     2)<>32THEN5060                         :rem 226
5065 POKEVS+XX+X*22,BU:POKEVS+XX+X*22+SO,13
                                            :rem 178
5085 GOTO150                                :rem 161
5500 PRINT"{CLR}"                           :rem 46
5510 PRINT"{11 DOWN}{6 RIGHT}GET READY"     :rem 90
5540 FORX=1TO14:POKE36876,NN(X):FORD=1TO120:NEXT:N
     EXT:POKE36876,0                        :rem 57
5550 RETURN                                 :rem 175
6500 REM REDEFINING SCREEN                  :rem 74
6575 GOSUB 6600                             :rem 35
6580 RETURN                                 :rem 179
6600 REM PRINT SCORE                        :rem 185
6605 PRINT"{YEL}{HOME}SCORE ";SC            :rem 160
6606 PRINT"{HOME}{13 RIGHT}ROOM";LO         :rem 66
6610 PRINT"MUSHROOMS ";HI;"LIVES";LI        :rem 9
6620 RETURN                                 :rem 174
7000 REM CLEAN UP THE CENTIPEDE             :rem 37
7002 SP=SP-5                                :rem 174
7004 GOSUB 6600:HI=5                        :rem 84
7005 L=LEN(XA$):IFL>127THENL=127            :rem 129
7010 FORI=1TOL-1                            :rem 179
7020 POKE36876,I+128:FORQQ=1TO10:NEXT       :rem 239
7190 PO=VS+ASC(XA$)+22*ASC(YA$):LL=LEN(XA$)-1:XA$=
     RIGHT$(XA$,LL)                         :rem 7
7195 YA$=RIGHT$(YA$,LL):POKEPO,32:POKEPO+SO,1
                                            :rem 17
7200 NEXT:POKE36876,0                       :rem 219
7210 RETURN                                 :rem 170
7500 REM OOPS                               :rem 241
7510 PRINT"{CLR}{PUR}"                      :rem 205
7515 SP=SP-5                                :rem 183
7520 PRINT"{10 DOWN}{9 RIGHT}OOPS"          :rem 143
7521 LI=LI-1                                :rem 148
7525 FORDE=1TO20:NEXT                       :rem 47
7530 FORDE=1TO10:POKE36876,DE*10+150:FORQQ=1TO10:N
     EXT:NEXT:POKE36876,0                   :rem 35
7550 FORDE=1TO20:NEXT                       :rem 45
7560 IFLI<1THEN7700                         :rem 96
7599 RETURN                                 :rem 190
7700 REM THE GAMES OVER                     :rem 60
7705 POKE36876,0                            :rem 108
7710 PRINT"{CLR}"                           :rem 51
7715 IF SC>HSTHENHS=SC:GOSUB9000:PRINT"{CLR}{8}"
                                            :rem 43
```

```
7718 PRINT"{2 DOWN}{9 RIGHT}NERM"                    :rem 1
7720 PRINT"{YEL}{4 DOWN}YOUR SCORE ";SC              :rem 31
7730 PRINT"{6}{5 DOWN}HIGH SCORE ";HS                :rem 2
7735 GOSUB 7800                                      :rem 37
7736 PRINT"{WHT}{DOWN}PRESS TRIGGER TO PLAY,Q TO Q
     UIT"                                            :rem 155
7740 FORX=1TO15:POKE36876,PN(X):FORD=1TO100:NEXT:N
     EXT:POKE36876,0                                 :rem 62
7780 GOSUB4000:IFFR<>0THEN5011                       :rem 27
7783 IFPEEK(197)=48THENPOKE198,0:PRINT"{CLR}{BLU}"
     ;:POKE36879,27:POKE36869,240:END                :rem 215
7785 GOTO 7780                                       :rem 234
7800 REM RANK THE GAMER                              :rem 44
7810 PRINT"{CYN}{2 DOWN}{3 SPACES}YOUR NEW RANK IS
      "                                              :rem 99
7820 IFLO=1THENPRINT"{9 SPACES}ZERO"                 :rem 169
7830 IFLO=2THENPRINT"{8 SPACES}ROOKIE"               :rem 52
7840 IFLO=3THENPRINT"{8 SPACES}NOVICE"               :rem 49
7850 IFLO=4THENPRINT"{7 SPACES}AVERAGE"              :rem 106
7860 IFLO=5THENPRINT"{8 SPACES}MASTER"               :rem 61
7870 IFLO=6THENPRINT"{5 SPACES}GRAND MASTER"
                                                     :rem 171
7880 IFLO=7THENPRINT"{8 SPACES}WIZARD"               :rem 70
7890 IFLO=8THENPRINT"{5 SPACES}GRAND WIZARD"
                                                     :rem 180
7900 IFLO=9THENPRINT"{6 SPACES}SUPER STAR"           :rem 57
7910 IFLO>9THENPRINT"{5 SPACES}HALL OF FAME"
                                                     :rem 65
7920 RETURN                                          :rem 178
9000 REM NEW HIGH SCORE                              :rem 51
9002 PRINT"{CLR}"                                    :rem 47
9003 PRINT"{CYN}{6 DOWN}{7 RIGHT}NEW HIGH":rem 119
9004 PRINT"{4 DOWN}{8 RIGHT}SCORE"                   :rem 70
9005 FORY=1TO3                                       :rem 82
9010 FORN=1TO5                                       :rem 69
9020 FORD=1TO5:NEXT                                  :rem 181
9030 POKE36876,N*20+150                              :rem 208
9050 NEXT                                            :rem 13
9060 NEXT:POKE36876,0                                :rem 225
9065 FORD=1TO30:NEXT                                 :rem 236
9070 RETURN                                          :rem 176
9100 REM EXTRA LIFE                                  :rem 82
9110 PRINT"{CLR}"                                    :rem 47
9115 PRINT"{CYN}{10 DOWN}{6 RIGHT}BONUS LIFE"
                                                     :rem 63
9120 FORJ=100TO200                                   :rem 0
9140 POKE36876,J+50                                  :rem 17
9150 NEXT                                            :rem 14
9160 POKE36876,0                                     :rem 105
```

```
9170 EX=EX+3                                        :rem 166
9180 LI=LI+1                                        :rem 149
9190 RETURN                                         :rem 179
10000 DIM PN(15),NN(18),JP(2,2):PRINT"{CLR}{8}"
                                                    :rem 130
10005 SO=38400-VS                                   :rem 170
10010 PRINT"{RED}{6 DOWN}{6 RIGHT}WELCOME TO"
                                                    :rem 162
10020 PRINT"{CYN}{6 DOWN}{4 RIGHT}WORM OF BEMER"
                                                    :rem 130
10030 PRINT"{YEL}{4 DOWN}{RIGHT}HIT TRIGGER TO STA
      RT"                                           :rem 238
10045 GOSUB4000:IFFR<>0THENRETURN                    :rem 88
10060 GOTO10045                                      :rem 42
11100 PRINT"{CLR}{CYN}{10 DOWN}REDEFINING
      {2 SPACES}CHARACTERS"                         :rem 91
11105 FORI=0TO2:FORJ=0TO2:READJP(J,I):NEXTJ,I
                                                    :rem 79
11110 FORI=7168TO7168+64*8:POKEI,PEEK(I+25600):NEX
      TI                                            :rem 46
11180 FORI=0TO39:READA:POKE7168+I+32*8,A:NEXT
                                                    :rem 201
11185 FORI=0TO7:READA:POKE7168+I+42*8,A:NEXT
                                                    :rem 154
11190 POKE36869,255                                   :rem 3
11195 POKE36878,14*16+15                            :rem 243
11200 FORI=1TO18:READNN(I):NEXT                     :rem 163
11210 FORI=1TO15:READPN(I):NEXT                     :rem 163
11230 DATA 10,14,6,11,15,7,9,13,5                    :rem 62
11240 DATA 0,0,0,0,0,0,0,0                          :rem 198
11250 DATA 0,40,170,170,255,60,60,0                 :rem 163
11260 DATA 85,85,85,85,85,85,85,85                  :rem 176
11261 DATA 170,190,190,190,190,190,170,170 :rem 19
11262 DATA 0,60,170,170,170,170,60,0                :rem 214
11263 DATA 0,40,85,85,85,85,40,0                     :rem 39
11270 RETURN                                        :rem 219
12000 DATA 195,209,0,209,215,0,215,219,225,219,225
      ,219,209,0,195,209,0,209                      :rem 52
12100 DATA 209,0,0,195,191,195,201,201,195,0,0,0,2
      07,207,209                                    :rem 108
```

Gotcha!

Doug Smoak

"Gotcha!" will keep you on the run as you scramble for dollars while avoiding the dreaded Collector. It is written for the unexpanded VIC; a joystick is required.

The idea of "Gotcha!" is to get all the money you can lay your hands on while outwitting the Collector at the same time. But you'd better be quick, because he's not that interested in the money itself. He wants to catch *you*.

You begin the game inside a diamond pattern in the middle of the screen, with the Collector hot on your heels. Once outside the diamond, you're free to move up, down, and diagonally in your effort to grab the dollars and elude the Collector. If you get trapped near a side, you can run off the screen and wrap around to the other side. But beware—the Collector knows where you are, and he comes onto the screen headed straight for you. You might not see him in time to escape—and then it's Gotcha!

You have the advantage of being able to move in eight directions, while the Collector can only move in straight lines across the screen. If you survive until all the money is gone, you move on to the next round, where there is more money with a higher score value. The screen changes to a maze after the first round, restricting your movement and making it easier for the Collector to track you down. If you survive 18 rounds (no one ever has), you can take your money and retire or play again.

Gotcha!

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 GOTO56                                              :rem 213
2 K=INT(.5+(ME-7767)/44):J=Ø:ET=K*44+7767:CH=-1:E1
  =2:IFRND(1)<.5THENET=ET-21:CH=1:E1=3                :rem 184
3 IFPEEK(ME)=36THENGOSUB5Ø                            :rem 132
4 IFPEEK(ET)=36THENIT=IT+1                            :rem 122
5 IFME<77Ø2THENME=ME+22                               :rem 129
6 IFME>8163THENME=ME-22                               :rem 136
7 POKEOM,32:POKEME,M1:POKEME+3Ø72Ø,7                  :rem 189
```

```
8 IFPEEK(ET)=M1THEN52                                    :rem 39
9 POKEOT,32:POKEET+30720,2:POKEET,E1                     :rem 199
10 IFIT=>(30+RD*10)THENPRINT"{CLR}":RD=RD+1:MT=0:O
   M=0:OT=0:GOTO72                                       :rem 75
11 POKEDD,127:P=PEEK(P2)AND128:J0=-(P=0)                 :rem 56
12 POKEDD,255:P=PEEK(P1):J1=-((PAND8)=0):J2=-((PAN
   D16)=0):J3=-((PAND4)=0)                               :rem 105
13 IFJ0THENDX=1:M1=1                                     :rem 204
14 IFJ1THENDY=22                                         :rem 220
15 IFJ2THENDX=-1:M1=0                                    :rem 252
16 IFJ3THENDY=-22                                        :rem 13
17 OM=ME:ME=ME+DX+DY:DY=0:DX=0                           :rem 132
18 IFME<7702 THENME=ME+22                                :rem 181
19 IFME>8164THENME=ME-22                                 :rem 189
20 IFPEEK(ME)<>32ANDPEEK(ME)<>36THENME=OM                :rem 48
21 OT=ET:ET=ET+CH:J=J+1:IFJ=>22THEN2                     :rem 245
22 GOTO3                                                 :rem 208
23 POKE36879,8:PRINT"{CLR}{RVS}{WHT}SCORE{OFF}"SC:
   IFRD=0THEN41                                          :rem 194
24 IFRD<19THENGOTO26                                     :rem 1
25 POKE36869,240:PRINT"{CLR}YOU MADE IT!!!!":GOTO2
   5                                                     :rem 21
26 UR$="E F":UL$="H G"                                   :rem 78
27 POKE36879,8:PRINT:FORT=1TO10:PRINT"{PUR}DDDDDDD
   DDDDDDDDDDDDDDD":NEXT                                 :rem 179
28 PRINT"{HOME}{2 DOWN}":OV$="{8 RIGHT}":AP$=""
                                                         :rem 76
29 FORT=1TO4:PRINTOV$UR$AP$UL$                           :rem 192
30 AP$=AP$+"{4 RIGHT}":OV$=OV$+"{2 LEFT}":NEXT
                                                         :rem 24
31 AP$=AP$+"{4 LEFT}":OV$=OV$+"{2 RIGHT}"                :rem 160
32 FORT=1TO4:PRINTOV$UL$AP$UR$                           :rem 186
33 AP$=AP$+"{4 LEFT}":OV$=OV$+"{2 RIGHT}":NEXT
                                                         :rem 27
34 FORT=1TO30+(RD*10)                                    :rem 179
35 SP=RND(1)*398+7744                                    :rem 125
36 IFPEEK(SP)=32THENPOKESP,36:POKESP+30720,5:GOTO3
   8                                                     :rem 187
37 GOTO35                                                :rem 11
38 NEXT                                                  :rem 170
39 DD=37154:P1=37151:P2=37152:ME=7932:V=36878:S=36
   875:IT=0                                              :rem 255
40 FORT=225TO177STEP-4:POKES,T:FORTT=30TO0STEP-1:P
   OKEV,TT/2:NEXTTT:NEXT:GOTO2                           :rem 153
41 PRINT"{RED}{HOME}{4 DOWN}{10 SPACES}FH"               :rem 55
42 PRINT"{DOWN}{8 SPACES}FDDDDH"                         :rem 230
43 PRINT"{DOWN}{6 SPACES}FDDDDDDDDH"                     :rem 247
44 PRINT"{DOWN}{4 SPACES}FDDDDDDDDDDDDH"                 :rem 8
45 PRINT"{DOWN}{4 SPACES}GDDDDDDDDDDDDE"                 :rem 7
```

```
46 PRINT"{DOWN}{6 SPACES}GDDDDDDDDE"          :rem 248
47 PRINT"{DOWN}{8 SPACES}GDDDDE"              :rem 233
48 PRINT"{DOWN}{10 SPACES}GE"                 :rem 218
49 GOTO34                                     :rem 13
50 POKES,235:FORT=1TO5:POKEV,3*T:NEXT:POKEV,0:IT=I
   T+1:SC=SC+10*(1+RD)                        :rem 65
51 PRINT"{HOME}{RVS}{WHT}SCORE{OFF}"SC:RETURN
                                              :rem 31
52 PRINT"{CLR}{WHT}{RVS}{7 SPACES}GOTCHA!!"
                                              :rem 218
53 POKES,0:POKEV,15:FORT=200TO240:POKES-1,T:NEXT:F
   ORT=1TO50:NEXT                             :rem 139
54 FORT=240TO126STEP-1:POKES-1,T:NEXT:POKEV,0
                                              :rem 123
55 RD=0:OM=0:OT=0:FL=1:GOTO70                 :rem 14
56 PRINT"{CLR}{5 RIGHT}{8 DOWN}JUST A MOMENT":POKE
   56,28:POKE52,28:CLR                        :rem 42
57 FORI=0TO511:POKE7168+I,PEEK(32768+I):NEXTI
                                              :rem 216
58 READX:IFX<0THEN65                          :rem 8
59 FORI=XTOX+7:READJ:POKEI,J:NEXTI:GOTO58     :rem 250
60 DATA7168,48,18,156,120,24,40,36,34,7176,24,81,5
   8,28,24,20,36,68                           :rem 250
61 DATA7184,60,230,126,30,30,30,254,124,7192,60,10
   3,126,120,120,120,127,62                   :rem 81
62 DATA7200,255,255,255,255,255,255,255,255
                                              :rem 139
63 DATA7208,255,254,252,248,240,224,192,128,7216,1
   ,3,7,15,31,63,127,255                      :rem 231
64 DATA7224,255,127,63,31,15,7,3,1,7232,128,192,22
   4,240,248,252,254,255,-1                   :rem 110
65 POKE36869,255:POKE36879,110:PRINT"{YEL}{CLR}
   {6 DOWN}{7 SPACES}{RVS}GOTCHA!!"           :rem 86
66 PRINT"{CLR}{RVS}USING THE JOYSTICK{4 SPACES}
   {DOWN}GATHER AS MUCH MONEY{2 SPACES}{DOWN}AS YO
   U CAN WITHOUT"                             :rem 118
67 PRINT"{RVS}{DOWN}BEING GOTTEN BY {OFF}{RED}C
   {RVS}{YEL}."                               :rem 1
68 PRINT"{DOWN}{RVS}YOU ARE {OFF}{YEL}A{RVS}. THE
   {SPACE}NUMBER{DOWN} AND VALUE OF THE {GRN}$
   {YEL}'S {DOWN} INCREASE WITH EACH"         :rem 107
69 PRINT"{RVS}{DOWN}ROUND.":GOTO72            :rem 245
70 PRINT"{RVS}{2 DOWN}SCORE"SC:PRINT"{RVS}{DOWN}HI
   GH"HS:IFSC>HSTHENHS=SC:GOSUB78             :rem 101
71 IFRD=0THENSC=0                             :rem 44
72 FORT=1TO500:NEXTT:PRINT"{RVS}{2 DOWN}PRESS THE
   {SPACE}TRIGGER TO":PRINT"{RVS}PLAY";       :rem 9
73 IFFL=1THENPRINT"{RVS} AGAIN, Q TO QUIT":rem 206
74 P=PEEK(37151):FB=-((PAND32)=0)             :rem 50
```

```
75 IFPEEK(197)=48 THEN POKE198,0:SYS4096    :rem 35
76 IFFB=0THEN74                            :rem 139
77 FL=0:GOTO23                              :rem 69
78 FORT=1TO1000:NEXT:FORCT=1TO3:PRINT"{RVS}
   {2 DOWN}{2 SPACES}A NEW HIGH SCORE!!"    :rem 57
79 POKEV,15:FORT=190TO245:POKES-1,T:NEXTT:FORTT=1T
   O200:NEXTTT:POKES-1,0:NEXTCT             :rem 139
80 RETURN                                   :rem 72
```

Freeway Zapper

Steve Elder

"Freeway Zapper" is a good example of the exciting graphics that good BASIC programming can produce—if you use the right programming techniques. It's an exciting game, too, particularly if you're one of those who can't wait to get to work every day! For the unexpanded VIC.

What a grand and glorious morning!

Eagerly you get out of bed, get dressed, and eat breakfast. With a cheerful spring in your step, you bound down the steps, hop into your car, and head for work. You *love* work, and you can hardly wait to arrive.

But what's this on the radio?

"Attention all motorists!" the announcer says. "A secret group of mad scientists, whose sole ambition is to control the world's freeways, has unleashed an army of sophisticated robots. Those robots are patrolling the interstate at this very minute, chasing all drivers away!"

You're stunned. What horrible news! You can't just sit there and let them take your beloved freeways away from you. That would be terrible—and if they did, *you couldn't get to work!*

Emotion overwhelms you. Oblivious to the danger, you dash onto the freeway, determined to get to work anyway. You know it's dangerous. Even the slightest touch of one of the robots and you've bought the big one for sure. So onward you drive, dodging those mechanical nightmares, determined to arrive on time. You are dogged. You are relentless. Tales of your bravery are sung throughout the land.

You are the symbol of hope for a broken nation.

You are *Freeway Zapper!*

"Freeway Zapper" runs on the unexpanded VIC, requiring only a joystick for steering. It uses multicolored custom characters, in combination with the built-in graphics set, for the robots and for your car.

This game uses several simple programming techniques to achieve speed and smoothness of execution. The greatest gain in speed is realized by placing the main loop at the beginning

of the program; all initialization, definition of variables, and so on is placed at the end and is accessed by a GOTO in the first line.

Speed is further increased by replacing often-used numbers with variables. The computer handles variables more quickly than it does numbers, and the improvement in execution speed can be significant.

There is another trick that you can use, too: Replace all zeros with periods. For example, if you replace A=0 with A=. your program will run still faster.

Freeway Zapper is easy to play. Enter a skill level from one to five (one is the hardest) and steer with your joystick. When you crash (and you will), both the high score and your current score will be displayed. To play again, simply press the joystick button.

Freeway Zapper

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 GOTO5ØØ                                          :rem 255
2Ø S=S+5:IFS>2ØØTHENS=2ØØ                          :rem 226
22 POKEV-4,S:GOTO1ØØ                               :rem 205
1ØØ IF(PEEK(J)AND16)=ØTHEND=-2                     :rem 231
1Ø2 IF(PEEK(J+O)AND128)=ØTHEND=2                   :rem 1Ø6
1Ø4 IFPEEK(P+D+T)<>BTHEN2Ø2                         :rem 87
1Ø6 POKEP,B:P=P+D:PRINT"{RVS}{4 RIGHT}{YEL}--
    {RIGHT}:{RIGHT}:{RIGHT}:{RIGHT}:{RIGHT}--":POK
    EP,Ø:POKEP+C,1Ø:D=Ø                            :rem 218
1Ø8 Q=Q+O:IFRND(O)>O-Q/SKTHENX=INT(RND(O)*9+8126):
    POKEX,O:POKEX+C,1Ø                             :rem 242
11Ø IF(Q+15)/1ØØ=INT((Q+15)/1ØØ)THENPRINT"{UP}
    {RVS}{CYN}"Q+15:GOTO2Ø                         :rem 121
2ØØ GOTO1ØØ                                         :rem 92
2Ø2 IFPEEK(P+D+T)>OTHEND=Ø:GOTO1Ø4                  :rem 76
2Ø4 POKEP,B:P=P+D:PRINT:POKEV-4,Ø                   :rem 76
2Ø6 POKEP,255:POKEP-23,255:POKEP-T,255:POKEP-21,25
    5:POKEP-O,255:POKEP+O,255                      :rem 253
2Ø8 POKEP+21,255:POKEP+T,255:POKEP+23,255          :rem 128
21Ø POKEV-O,2Ø0:FORX=15TOØSTEP-.2:POKEV,XOR112:POK
    E36865,RND(O)*1Ø+2Ø:NEXT                        :rem 56
212 POKEV-O,Ø:POKE36865,25:PRINT"{HOME}{RVS}{CYN}
    {6 RIGHT}SCORE:"Q                              :rem 126
214 IFQ>HSTHENHS=Q                                 :rem 168
216 PRINT"{RVS}{3 RIGHT}HI-SCORE:"HS               :rem 226
218 PRINT"{19 DOWN}{RVS}{2 RIGHT}PRESS FIRE BUTTON
    "                                              :rem 138
```

```
300 IF(PEEK(J)ANDB)THEN300                          :rem 10
302 PRINT"{CLR}":FORX=1TO500:NEXT:D=0:Q=0:GOTO520
                                                    :rem 126
500 POKE36879,11                                    :rem 99
502 PRINT"{CLR}{WHT}{4 DOWN}{3 RIGHT}FREEWAY ZAPPE
    R!{5 DOWN}"                                     :rem 246
504 PRINT"{CYN} STEER LEFT AND RIGHT{3 SPACES}WITH
     THE JOYSTICK."                                 :rem 195
506 PRINT"{YEL}{2 DOWN}{2 SPACES}CHOOSE SKILL LEVE
    L{3 SPACES}{DOWN}1-HARDEST{2 SPACES}5-EASIEST"
                                                    :rem 206
508 GETA$:IFA$=""THEN508                            :rem 93
510 X=ASC(A$)-48:IFX<1ORX>5THEN508                  :rem 23
512 POKE7679,X:PRINT"{CLR}":POKE52,28:POKE56,28:PO
    KE51,0:CLR                                      :rem 220
514 FORX=0TO15:READY:POKEX+7168,Y:NEXT              :rem 190
516 FORX=7424TO7431:POKEX,0:NEXT                    :rem 241
518 DATA40,105,125,60,40,105,105,40,60,60,170,40,4
    0,20,65,65                                      :rem 156
520 V=36878:C=30720:J=37151:P=7800:B=32:S=130:SK=1
    00+200*PEEK(7679):O=1:T=22                      :rem 202
522 POKEV,127:POKEV-9,255:POKEJ+3,127:POKEV-4,S
                                                    :rem 71
524 FORX=0TO21:PRINT"{RVS}{4 RIGHT}{YEL}--{RIGHT}:
    {RIGHT}:{RIGHT}:{RIGHT}:{RIGHT}--":NEXT :rem 1
600 GOTO100                                         :rem 96
```

Wheeler

Phil Callister

If you've ever tried to maneuver a solitary tire across an obstacle-strewn desert, you'll appreciate the problem posed in "Wheeler." For the unexpanded VIC.

"Wheeler" is an exciting game for the unexpanded VIC. The object is to maneuver a wheel across the desert, avoiding pits, rocks, nails, knives, and scissors.

You can avoid the hazards by jumping over them. Press 1, 2, or 3 to jump one, two, or three spaces, or (if things get really bad) press H (for HOPE) to make a random jump of from two to nine spaces.

You get five points every time you jump over an object and ten points every time you make it to the end of a screen. When you have completed five screens, you will move on to level two. There you are presented with more obstacles. The number of obstacles increases with every five screens until you reach level ten.

Start by typing in and saving the program. Then RUN it, and you'll get the title screen. Hit f1 to begin play; to play again after one round is completed, press Y (to return you to the title screen) then f1 (to start a new game).

Roughening the Road

If you ever decide that the course is too easy, it is not hard to add a few more road hazards to your path. Change variables LV and MN to increase the number of pits and obstacles. Similarly, you can change variable RS to speed up or slow down the game. Increasing the value of RS slows the action; decreasing it speeds things up.

Wheeler

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
5 POKE36879,30:PRINT"{CLR}":GOTO600          :rem 171
6 PRINT"{HOME}":LV=1:P=5:SC=0:QW=5           :rem 241
7 PRINT"{HOME}{2 DOWN}{RVS}{BLK}{6 RIGHT}
  {7 SPACES}":PRINT"{HOME}{8 DOWN}{7 SPACES}":PRIN
  T"{HOME}{2 DOWN}{RVS}SCORE"SC:PRINT"{RVS}{DOWN}L
  EVEL"LV                                    :rem 223
```

```
8 MN=3:RS=300:BS=30720:HE=3                                    :rem 49
9 IFSD=1THENSD=0:FORI=8010TO8185:POKEI,10:POKEI+BS
  ,6:NEXT                                                      :rem 117
10 PRINT"{HOME}{17 DOWN}{5 RIGHT}{BLU}{RVS}* WHEEL
   ER *":PRINT"{2 DOWN}{2 RIGHT}{RVS}BY{2 SPACES}P
   HIL CALLISTER"                                              :rem 108
15 GETA$:IFA$="{F1}"THEN20                                     :rem 112
16 GOTO15                                                       :rem 6
20 C=1:V=7900:R=0                                              :rem 169
30 X$="{CYN}@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   @@@@@@@"                                                    :rem 63
35 Z$="{CYN}{UP}@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"                             :rem 215
37 Y$="{22 SPACES}"                                            :rem 104
40 PRINT"{HOME}{9 DOWN}"                                       :rem 225
70 PRINTY$+X$+Z$                                               :rem 193
75 FORI=1TOMN:H=INT(RND(1)*19)+2:POKE7922+H,9:POKE
   7922+H+BS,5:NEXT                                            :rem 249
80 FORG=1TOLV                                                   :rem 73
85 H=INT(RND(1)*19)+2:IFPEEK(H+7922)=9THEN85
                                                               :rem 29
90 Z=INT(RND(1)*5)+1:POKEH+7900,Z                              :rem 143
91 SX=INT(RND(1)*5)+2:POKE7900+H+BS,SX:NEXT
                                                               :rem 108
100 POKEV,32:V=V+1                                              :rem 38
102 IFR=PTHENR=0:LV=LV+1:MN=MN+1:RS=RS-25                       :rem 100
110 IFV>=7921THENR=R+1:V=7900:SC=SC+10:GOTO40
                                                               :rem 212
112 IFLV>10THENLV=10                                           :rem 211
113 IFMN>10THENMN=10                                           :rem 198
114 IFRS<50THENRS=50                                           :rem 225
120 PRINT"{HOME}{2 DOWN}{BLU}{RVS}SCORE"SC:rem 220
121 IFSC>HITHENHI=SC                                            :rem 27
125 PRINT"{HOME}{2 DOWN}{RVS}{13 RIGHT}{PUR}HI"HI:
    PRINT"{DOWN}{RVS}{BLU}LEVEL"LV                             :rem 78
130 IFPEEK(V)<>32THEN500                                       :rem 146
131 QW=QW+1:IFQW=9THENQW=6                                     :rem 175
135 IFPEEK(V+22)=9THENV=V+22:GOTO500                           :rem 168
136 POKEV,QW:POKEV+BS,0:POKE36878,10:POKE36875,190
    :FORT=1TO10:NEXT:POKE36875,0                               :rem 66
137 FORT=1TORS:NEXT:POKE36875,210:FORT=1TO10:NEXT:
    POKE36875,0                                                :rem 108
140 GETA$:IFA$=""THEN100                                        :rem 73
151 IFA$="1"THENHB=1:GOTO158                                   :rem 117
152 IFA$="2"THENHB=2:GOTO158                                   :rem 120
153 IFA$="3"THENHB=3:GOTO158                                   :rem 123
154 IFA$="H"THENHB=INT(RND(1)*8)+2:GOTO158:rem 191
156 GOTO100                                                    :rem 102
```

```
158 POKE36878,15:FORI=1TO20STEP4:POKE36876,160+I:N
    EXT:POKE36876,0                          :rem 182
160 POKEV,32:V=V-22:FORG=1TOHB:POKEV,32:V=V+1:POKE
    V,QW:POKEV+BS,0:FORT=1TORS:NEXT          :rem 69
161 IFV>=7899THENPOKEV,32:SC=SC+10:R=R+1:GOTO40
                                             :rem 155
163 IFPEEK(V+22)<>32ORPEEK(V+44)=9THENPOKEV+22,13:
    POKEV+22+BS,6:SC=SC+5:GOSUB700           :rem 112
164 POKEV+22,32:NEXT                         :rem 185
165 POKEV,32:V=V+22                          :rem 100
170 GOTO100                                   :rem 98
500 POKEV+BS,0:POKEV,11:FORI=10TO250STEP5:POKE3687
    6,250-I:NEXT:POKEV,12:POKE36875,150      :rem 229
501 POKE36876,0:FORT=1TO10:NEXT:POKE36875,0:PRINT"
    {HOME}{17 DOWN}{5 RIGHT}{RVS}{BLU}*GAME OVER*"
                                             :rem 12
510 PRINT"{DOWN}{2 RIGHT}{RVS}{DOWN}PLAY ANOTHER G
    AME?"                                    :rem 117
520 POKE198,0                                :rem 196
530 GETB$:IFB$=""THEN530                      :rem 85
540 IFB$="Y"THEN6                            :rem 205
550 IFB$<>"N"THEN530                          :rem 98
560 PRINT"{CLR}{BLU}":POKE36879,27:POKE36869,240:E
    ND                                       :rem 121
600 POKE52,28:POKE56,28:POKE51,PEEK(55):CLR:A=256*
    28+PEEK(51):FORX=256TO264:POKEA+X,0      :rem 45
610 NEXT:FORX=0TO111:READRT:POKEX+A,RT:NEXT:POKE36
    869,255                                  :rem 167
620 DATA255,255,255,255,255,255,255,255,0,8,8,8,8,
    8,8,28,56,124,110,222,250,110,126,52     :rem 195
625 DATA128,96,80,42,20,14,23,3,129,66,36,24,24,23
    1,165,231,16,72,48,176,88,12,6,3         :rem 253
630 DATA24,60,94,102,102,122,60,24,24,60,122,102,1
    02,94,60,24,24,60,126,78,98,126,60,24    :rem 209
635 DATA0,0,0,0,170,170,255,255,239,251,223,254,24
    7,223,237,127                             :rem 70
640 DATA0,0,60,126,223,231,227,126,0,0,0,0,28,62,9
    9,255                                    :rem 168
645 DATA0,117,69,117,16,117,0,0               :rem 30
650 SD=1:GOTO6                                :rem 73
700 FORI=1TO10STEP2:POKE36876,235+I:NEXT:POKE36876
    ,0:FORI=1TO220:NEXT:RETURN                :rem 70
```

Olympiad

Kevin Woram and Mike Buhidar, Jr.

In this mythical struggle between a magician and a king, you decide the fate of the realm with your joystick. For two players and a VIC with at least 8K expansion.

Long ago, Admar (a magician of great power) served the king of Denbar as an advisor. Through the years, Admar's power grew so that the king began to fear him. Foolishly, the king decided that because of his power, Admar could no longer be trusted, and he plotted to kill the magician.

Admar, actually still loyal to the king, learned of the king's plot. Fearing for his own life, he fled the capital with a legion of his own warriors.

The king followed with an army of his own and attacked Admar's stronghold. There were heavy casualties on both sides. It took time, but finally the king and Admar realized that continued battle would result in nothing more than general bloodshed.

An Enchanted Arena

So it was agreed that an enchanted arena would be built where the king's Black Knights would do mock battle with Admar's Red Knights. Whoever's knights won would claim the kingdom as his own.

You and a friend control the actions of the knights as they fight for their masters. Player 1 controls his knight with the joystick, while player 2 uses the keyboard (I, J, K, and M for up, left, right, and down movements).

The knights have been given 20 magical arrows which stun on contact. Press the fire button (or space bar) to fire an arrow. When your knight has used all of his arrows, your only defense is to run. If both warriors exhaust their arrows, the round will start anew, with each player receiving a fresh supply of 20 arrows.

Teleportation Grids

To add an element of randomness to the battle, three teleportation grids have been added to the arena. A large one is in the center, while the other two are in corners. Either

knight may use any of the three grids. When any warrior steps onto one of these grids, he is instantly teleported to a random position in the arena.

There are also two doors on each side of the arena which allow you to move directly from one side to the other, in effect wrapping around the screen. You can even shoot arrows through these doors; if the opposing knight happens to be standing in front of the door on the left side, for instance, and you fire *through* the door on the right, you can stun him.

Typing In Olympiad

Before loading the game into your VIC (that is, right after the computer is turned on), carefully enter the following lines:

```
POKE43,1:POKE44,32:POKE8192,0:NEW
POKE36869,240:POKE36866,150:POKE648,30
PRINTCHR$(147)
```

"Olympiad" makes extensive use of keyboard graphics in drawing the arena display. To avoid confusion and possible typing errors, please refer to "How to Type In Programs," Appendix B, before you attempt to enter this program. Using "The Automatic Proofreader," Appendix C, will virtually guarantee that you type Olympiad correctly the first time. Make sure you read the explanation and have a copy of the Proofreader program available before you start typing.

Pay close attention to lines 3010–3210. If any spaces are to be left after the characters on one line of the page, the correct number will be indicated in braces at the beginning of the next line. Unless you are specifically instructed to type spaces, do not do so.

Note that spaces are sometimes called for *within* a line. Single spaces are not indicated by braces—there's just a gap. Whenever more than one space is to be inserted, you'll see the number in braces.

Olympiad

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
0 POKE36879,26:GOTO 1000                          :rem 56
1 SCR=256*PEEK(648):A=3072Ø:IFPEEK(648)=16THEN A=3
  3792                                            :rem 35
2 DIM X(50),CS(50),D2(50),C2(80),DX(50)          :rem 194
4 N=15:B=32:V=36878:S1=36874:S4=36877             :rem 52
5 PB=37152:JB=16:HP=102                           :rem 18
```

```
10 RN=1:COL=A:LB=SC+4:FB=SC+10                    :rem 32
15 RO=SC+22:FB=SC+3:FO=SC+15:LB=SC+4:LR=SC+15:GOSU
   B 3010                                         :rem 67
22 D2(0)=0:D2(1)=-22:D2(2)=22:D2(4)=-1:D2(5)=-23:D
   2(6)=21:D2(8)=1:D2(9)=-21                      :rem 246
25 DX(10)=23:CS(25)=192:CS(23)=193:CS(46)=195:CS(2
   )=194:CS(45)=197                               :rem 21
27 D2(10)=23:C2(12)=194:C2(20)=193:C2(44)=192:C2(3
   6)=195                                         :rem 51
30 CS(47)=199:CS(3)=196:CS(1)=198                 :rem 247
34 RESTORE:GOSUB400:XX=0:CS(0)=195:C2(0)=194:FORI=
   1TO160:READPI:NEXT                             :rem 226
36 PRINT"{3 UP}";SPC(JB/2);"{7 SPACES}"           :rem 203
49 OP=SCR+45:O2=SCR+482:UP=OP:U2=O2               :rem 168
50 POKEOP,195:POKEO2,194:POKEOP+CO,0:POKEO2+CO,2
                                                  :rem 216
60 IF NA+AT=0 THEN POKE OP,B:POKEO2,B:RN=RN-1:GOTO
   34                                             :rem 99
61 POKE 37154,127:P=PEEK(37152)AND128:J0=-(P=0):PO
   KE37154,255                                    :rem 110
63 P=PEEK(37151):J1=-((PAND8)=0):J2=-((PAND16)=0):
   J3=-((PAND4)=0)                                :rem 54
64 IF-((PAND32)=0)=1THENGOSUB 100                 :rem 107
65 IF XX=1 THEN 34                                :rem 174
66 VV=(J0-J2)+(J1-J3)*22:IFVV=0THEN75             :rem 117
67 UP=OP+VV:JV=VV+24:CS(0)=CS(JV)                  :rem 46
68 IF(PEEK(UP)<>B)AND(PEEK(UP)<>96)THENGOSUB4000:G
   OTO 75                                         :rem 159
70 POKEOP,B:POKEUP+CO,0:POKEUP,CS(JV):OP=UP:rem 70
75 AA=PEEK(197):IF(AA<>12)AND(AA<>20)AND(AA<>36)AN
   D(AA<>44)THEN95                                :rem 235
76 BB=INT(AA/10):ONBBGOTO80,78,77,79               :rem 73
77 U2=O2+22:GOTO81                                 :rem 30
78 U2=O2-1:GOTO81                                 :rem 238
79 U2=O2+1:GOTO81                                 :rem 237
80 U2=O2-22                                        :rem 62
81 IF(PEEK(U2))<>BAND(PEEK(U2)<>96)THENGOSUB4100:G
   OTO 95                                         :rem 97
82 IF XX=1 THEN 34                                :rem 173
90 POKEO2,B:POKEU2+CO,2:POKEU2,C2(AA):O2=U2:CC=AA
                                                  :rem 244
95 IF PEEK(197)=32THENGOSUB110                    :rem 247
97 GOTO60                                          :rem 15
99 REM SHOOT ARROW                                :rem 110
100 IFNA=0THENRETURN                               :rem 43
101 NA=NA-1:BP=INT(NA/10):IFBP>1THENBP=1           :rem 168
102 PRINT"{HOME}{BLK}";NA:POKESC+2+BP,B:D=DX(JV):J
    C=CS(JV):GOSUB 200                            :rem 119
105 AP=UP+D:C1=0:GOTO115                          :rem 157
```

```
110 IFAT=0THENRETURN                                    :rem 50
111 AT=AT-1:BT=INT(AT/10):IFBT>1THENBT=1                :rem 199
112 PRINT"{HOME}{RED}";SPC(18);AT:POKESC+20+BT,B:D
    =D2(CC):JC=C2(CC):GOSUB200                          :rem 158
114 AP=U2+D:C1=2                                        :rem 119
115 AD=JC+8:IF(PEEK(AP)<>B)AND(PEEK(AP)<>96)THENRE
    TURN                                                :rem 73
120 POKEV,2:POKES4,200:FORA=1TO13:NP=AP+D               :rem 16
125 AC=NP+CO                                            :rem 180
130 IF(PEEK(NP)<>B)AND(PEEK(NP)<>96)THENPOKEV,0:PO
    KES4,0:GOTO300                                      :rem 199
140 POKEAP,B:POKEAC,C1:POKENP,AD:AP=NP:NEXT:POKEAP
    ,B:POKEV,0:POKES4,0:RETURN                          :rem 16
199 REM STILL CHECKER                                   :rem 4
200 IFD<>0THENRETURN                                    :rem 30
210 IFJC=192THEND=1:RETURN                              :rem 131
220 IFJC=193THEND=-1:RETURN                             :rem 178
230 IFJC=195THEND=22:RETURN                             :rem 187
240 IFJC=194THEND=-22:RETURN                            :rem 232
250 IFJC=197THEND=21:RETURN                             :rem 190
260 IFJC=198THEND=-23:RETURN                            :rem 239
270 IFJC=194THEND=-21:RETURN                            :rem 234
280 D=23:RETURN                                         :rem 154
299 REM DEATH                                           :rem 238
300 IFPEEK(NP)<192THENPOKEAP,B:RETURN                   :rem 133
310 IFC1=0THEN330                                       :rem 201
312 POKELB,B:LB=LB+1:GOSUB600                           :rem 0
315 IFLB=SC+7THEN6000                                   :rem 224
317 XX=1:RETURN                                         :rem 211
330 POKELR,B:LR=LR+1:GOSUB610                           :rem 49
335 IFLR=SC+18THEN6010                                  :rem 37
340 XX=1:RETURN                                         :rem 207
400 NA=20:AT=20:PRINT"{HOME}{BLK}";NA;SPC(14);"
    {RED}";AT                                           :rem 234
410 PRINT"{BLU}{HOME}{2 DOWN}{RVS}";SPC(8);"ROUND"
    ;RN;"{OFF}":RN=RN+1:RETURN                          :rem 150
600 DP=UP:OM=U2:GOTO620                                 :rem 177
610 DP=U2:OM=UP                                         :rem 167
620 POKEAP,B:POKEOM,B:FORK=210TO208STEP-1:POKEDP,K
    :FORH=1TO100:NEXT:NEXT                              :rem 189
630 POKEDP,211:GOSUB7000:POKEDP,B:POKEUP,B:POKEU2,
    B:RETURN                                            :rem 252
650 IFJC=196THEND=21:RETURN                             :rem 193
1000 PRINT"{CLR}{BLK}LOADING CHARACTER SET INTO ME
     MORY. . .":PRINTCHR$(142)                          :rem 121
1010 FORI=5120TO7168:POKEI,PEEK(I+27648):NEXT
                                                        :rem 188
1020 POKE 36869,253                                     :rem 200
1045 IFPEEK(13983)=102THEN1060                          :rem 157
```

```
1090 FORNP=6656TO6815:READMD:POKENP,MD:NEXT
                                              :rem 254
1100 GOTO 1                                    :rem 44
1999 REDEFINED CHARACTERS                      :rem 66
2000 DATA102,227,241,159,159,241,227,102       :rem 216
2010 DATA102,199,143,249,249,143,199,102       :rem 235
2020 DATA126,219,153,24,60,231,231,126         :rem 113
2030 DATA126,231,231,60,24,153,219,126         :rem 114
2040 DATA60,6,207,253,201,201,124,60             :rem 6
2050 DATA60,62,147,147,191,243,96,60           :rem 36
2060 DATA60,96,243,191,147,147,62,60           :rem 37
2070 DATA60,124,201,201,253,207,6,60            :rem 9
2082 DATA0,132,66,63,66,132,0,0,0,33,66,252,66,33,
     0,0,16,56,84,16,16,16,40,68               :rem 233
2084 DATA68,40,16,16,16,84,56,16,7,3,5,8,16,224,32
     ,32,4,4,7,8,16,160,192,224                :rem 202
2086 DATA224,192,160,16,8,7,4,4,32,32,224,16,8,5,3
     ,7                                        :rem 39
2088 DATA0,0,8,16,4,16,0,0                     :rem 26
2090 DATA0,0,20,10,32,20,0,0                   :rem 99
2092 DATA68,9,32,132,1,40,130,17               :rem 78
2093 DATA 0,0,0,0,0,0,0,0                      :rem 156
2094 REMDATA126,90,126,60,0,102,24,102         :rem 190
2605 IFJC=198THEND=-23:RETURN                   :rem 36
2999 REM PLAYFIELD                             :rem 91
3010 PRINT"U*{R}****************{R}*I";        :rem 82
3020 PRINT"- -{16 SPACES}-{RED}{+}{BLU}-";     :rem 38
3030 PRINT"-{3 SPACES}U**I U**I U**I{3 SPACES}-";
                                               :rem 230
3040 PRINT"-{3 SPACES}J**K{SHIFT-SPACE}J**K
     {SHIFT-SPACE}J**K{3 SPACES}-";            :rem 12
3050 PRINT"-{20 SHIFT-SPACE}-";                :rem 14
3060 PRINT"- U***I{8 SPACES}U***I{SHIFT-SPACE}-";
                                               :rem 235
3070 PRINT"-{SHIFT-SPACE}J***K{8 SPACES}J***K
     {SHIFT-SPACE}-";                          :rem 122
3080 PRINT"-{2 SHIFT-SPACE}{5 SPACES}U{W}
     {2 SPACES}{Q}I {5 SHIFT-SPACE} -";        :rem 237
3090 PRINT"-{6 SPACES}UK{4 SPACES}JI{2 SPACES}UI
     {2 SPACES}-";                             :rem 99
3100 PRINT"-{2 SPACES}UI{2 SPACES}-{6 SPACES}- UKJ
     I J";                                     :rem 2
3110 PRINT"K UKJI -{2 SPACES}{RED}{+}{BLK}{+}{BLU}
     {2 SPACES}- JIUK{2 SPACES}";              :rem 211
3120 PRINT"{2 SPACES}JIUK -{2 SPACES}{BLK}{+}{RED}
     {+}{BLU}{2 SHIFT-SPACE}-{2 SPACES}JK
     {2 SPACES}U";                             :rem 128
3130 PRINT"I{2 SPACES}JK{2 SPACES}-{6 SHIFT-SPACE}
     -{6 SPACES}-";                            :rem 136
```

```
3140 PRINT"- {SHIFT-SPACE}{4 SPACES}JI
     {4 SHIFT-SPACE}UK{6 SPACES}-";          :rem 225
3150 PRINT"-{7 SPACES}J{W}{2 SPACES}{Q}K{7 SPACES}
     -";                                     :rem 130
3155 PRINT"- U***I{8 SPACES}U***I -";        :rem 80
3160 PRINT"- J***K{8 SPACES}J***K -";{6 SPACES}
                                             :rem 58
3170 PRINT"-{20 SPACES}-";                   :rem 145
3180 PRINT"-{3 SHIFT-SPACE}U**I U**I{SHIFT-SPACE}U
     **I{3 SHIFT-SPACE}-";                   :rem 76
3190 PRINT"-{3 SPACES}J**K{SHIFT-SPACE}J**K
     {SHIFT-SPACE}J**K{3 SPACES}-";          :rem 18
3200 PRINT"-{BLK}{+}{BLU}-{16 SPACES}- -";:rem 154
3210 PRINT"J*{E}**********{E}*";             :rem 126
3220 POKE505+SCR+A,6:POKE505+SCR,75          :rem 31
3225 FORI=0TO2:POKECO+LB+I,0:POKELB+I,195:POKECO+L
     R+I,2:POKELR+I,194                      :rem 211
3226 NEXT                                    :rem 12
3230 RETURN                                  :rem 168
3999 REM HIT DATA                            :rem 193
4000 IFPEEK(UP)<>HPTHENUP=OP:RETURN          :rem 74
4010 RF=INT(RND(1)*482)+RO:IFPEEK(RF)<>BTHEN4010
                                             :rem 87
4020 UP=RF:POKEUP+CO,0:POKEOP,B:MP=UP:JP=JV:OP=UP:
     GOSUB5000:RETURN                        :rem 155
4100 IFPEEK(U2)<>HPTHENU2=O2:RETURN          :rem 241
4110 R2=INT(RND(1)*482)+RO:IFPEEK(R2)<>BTHEN4110
                                             :rem 49
4120 U2=R2:POKEU2+CO,2:POKEO2,B:MP=U2:JP=J2:GOSUB5
     000:O2=U2:RETURN                        :rem 178
5000 FORMN=208TO210:POKEMP,MN:FORW=1TO150:NEXT:NEX
     T:POKEMP,CS(JP):RETURN                  :rem 255
5999 REM END ROUTINE                         :rem 193
6000 WN$=" RED ":LS$=" BLACK ":GOTO6020       :rem 1
6010 WN$=" BLACK ":LS$=" RED "               :rem 199
6020 PRINT"{CLR}{DOWN}[BLK} THE";WN$;"KNIGHTS"
                                             :rem 73
6030 PRINT" DEFEATED THE";:PRINTLS$          :rem 114
6040 PRINT" KNIGHTS IN ";RN-1;" ROUNDS"      :rem 221
6060 PRINT"{3 DOWN}PRESS SPACEBAR TO PLAY":PRINT"
     {DOWN} ANY OTHER KEY TO END"            :rem 96
6063 POKE 198,0                              :rem 252
6065 GETI$:IFI$=""THEN6065                   :rem 213
6067 IF I$<>" "THEN END                      :rem 147
6070 CLR:GOTO1                               :rem 82
6999 REM DEATH SOUND                         :rem 180
7000 POKEV,12:POKES4,150:FORI=12TO1STEP-1:FORJ=1TO
     30                                      :rem 228
7010 NEXT J:POKEV,I:NEXTI:POKES4,0:RETURN    :rem 173
```

The Frantic Fisherman

David Lacey

Idly floating in your boat, waiting for the fish to bite, is a fine way to relax. In this game, however, an angler's dream becomes a nightmare when sharks get the notion that you're the bait and the thunderclouds threaten you with gargantuan raindrops. It's good you remembered to bring your shark swatter and an umbrella. For the unexpanded VIC.

The fish are biting, and you've managed to catch a few. But suddenly you notice the sky is clouding over, and to make things worse, ravenous sharks begin to circle your boat.

The object of "Frantic Fisherman" is to survive. You score points by clubbing the sharks with your bat and blocking raindrops with your umbrella. You start with three fishermen. Each time a shark or raindrop hits the boat, you lose the boat and one fisherman. However, a new fisherman is awarded for every 2000 points.

Three keys are used to control movement. To move back and forth, use the less than (<) and greater than (>) keys. The space bar serves two functions. When the sharks approach, it controls the club. If a raindrop is falling, it controls the umbrella. You can use the shark swatter as many times as you like. The umbrella, though, can be lifted only three times for each raindrop.

If you think the game is too fast or slow, you can make the fisherman more (or less) frantic. Alter the speed by changing the variable DE in line 30 of Program 2. To add more fishermen, increase the value of GL in line 100.

Loading the Programs

The game runs on an unexpanded VIC, but it is made up of two programs. The first redefines the character set, while the second is the main program.

First enter Program 1. If you are using a disk drive, add the following lines:

```
700 PRINT"LOAD"CHR$(34)"FRANTIC2"CHR$(34)",8"
710 POKE198,4:FORT=631TO633:POKET,145:NEXT:POKE634
    ,13:END
```

Cassette users should add this line:

```
700 POKE198,1:POKE631,131:END
```

Next, type in Program 2 and SAVE it as "FRANTIC2". To play the game, LOAD and RUN Program 1, and it will load and run Program 2 automatically.

Frantic Fisherman, Program 1

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 PRINT"{CLR}{3 DOWN}{2 SPACES}FRANTIC{2 SPACES}FI
  SHERMAN"                                  :rem 108
2 PRINT"{4 DOWN} LOADING CHARACTERS.."      :rem 129
10 FORT=7168TO7168+62*8-1:READA:POKET,A:NEXT
                                            :rem 181
20 DATA14,62,254,62,14,2,6,6                :rem 131
30 DATA14,14,14,22,22,22,54,54              :rem 222
40 DATA118,118,246,254,246,246,246,246      :rem 141
50 DATA0,0,0,1,1,3,7,7                       :rem 70
60 DATA246,246,246,254,246,246,246,246      :rem 147
70 DATA15,31,63,127,255,255,255,255         :rem 242
80 DATA0,0,0,0,0,3,7,31                     :rem 116
90 DATA246,246,6,254,254,6,6,15              :rem 45
100 DATA255,255,8,255,255,0,0,0              :rem 23
110 DATA127,127,64,255,255,0,0,0             :rem 70
120 DATA255,255,127,127,63,31,15,7          :rem 183
130 DATA170,255,85,0,255,85,0,170           :rem 132
140 DATA255,255,254,254,252,248,240,224     :rem 182
150 DATA6,12,24,240,192,0,0,0               :rem 165
160 DATA0,0,0,192,240,24,12,6               :rem 166
170 DATA96,48,24,15,3,0,0,0                  :rem 80
180 DATA0,0,0,3,15,24,48,96                  :rem 81
190 DATA40,40,170,60,28,252,12,60           :rem 122
200 DATA12,40,40,248,248,40,20,40           :rem 113
210 DATA40,40,170,60,52,63,48,60             :rem 73
220 DATA48,40,40,47,47,40,20,40              :rem 22
230 DATA32,48,190,187,252,255,190,128        :rem 88
240 DATA4,12,125,221,63,255,125,1           :rem 118
250 DATA0,0,0,0,0,0,7,195                   :rem 219
260 DATA0,0,0,0,1,13,109,255                :rem 113
270 DATA0,0,0,0,128,224,248,248              :rem 22
280 DATA0,0,0,0,1,15,15,15                   :rem 11
```

```
290 DATA7,3,15,63,255,255,255,255          :rem 146
300 DATA199,255,255,255,255,255,255,255    :rem 200
310 DATA255,255,255,255,255,255,255,255    :rem 194
320 DATA0,192,240,240,224,248,252,248       :rem 72
330 DATA255,7,31,127,3,24,31,31             :rem 22
340 DATA0,0,0,0,0,0,0,0                    :rem 101
350 DATA255,255,255,255,255,127,255,255,0,0,0,0,0,
    0,0,0                                  :rem 164
360 DATA255,255,255,255,252,255,255,224    :rem 192
370 DATA255,192,240,252,0,0,128,0          :rem 122
380 DATA0,0,63,0,0,0,0,0                   :rem 162
390 DATA3,3,193,1,1,1,0,0                  :rem 224
400 DATA0,0,0,0,0,3,12,0                   :rem 152
410 DATA0,0,0,48,192,0,0,0                  :rem 11
420 DATA127,127,63,63,31,15,7,3             :rem 30
430 DATA1,2,4,8,16,32,64,128               :rem 133
440 DATA0,0,0,0,4,4,8,8                    :rem 126
450 DATA255,127,31,7,0,0,0,4               :rem 124
460 DATA16,16,32,32,0,0,0,0                 :rem 64
470 DATA4,4,4,4,4,4,4,0                    :rem 133
475 DATA255,199,189,207,245,143,255,255    :rem 211
480 DATA255,129,145,169,169,169,145,255    :rem 210
490 DATA255,129,145,177,145,145,185,255    :rem 202
500 DATA255,129,153,165,137,145,189,255    :rem 195
510 DATA255,129,185,137,153,137,185,255    :rem 195
520 DATA255,129,153,169,189,137,137,255    :rem 202
530 DATA255,129,189,161,185,133,185,255    :rem 199
540 DATA255,129,157,161,185,165,153,255    :rem 195
550 DATA255,129,189,133,137,145,145,255    :rem 196
560 DATA255,129,153,165,153,165,153,255    :rem 192
570 DATA255,129,153,165,157,133,185,255    :rem 197
580 DATA20,58,28,119,8,54,8,54               :rem 1
590 DATA8,8,28,20,58,62,62,28              :rem 207
600 DATA28,62,127,73,8,8,40,16             :rem 245
610 DATA16,2,32,136,80,42,116,56            :rem 78
620 REM LOWER MEMORY 512 BYTES             :rem 253
630 POKE52,PEEK(52)-2:POKE56,PEEK(56)-2    :rem 215
```

Frantic Fisherman, Program 2

```
10 CL$="{WHT}WXY{5 LEFT}{DOWN}Z[ff]↑{5 LEFT}
   {DOWN}←1#$"                             :rem 213
20 DEF FNRN(X)=INT(RND(1)*X)               :rem 111
30 V=36878:NO=V-1:S=V-2:S2=V-3:S3=V-4:CO=30720:EG=
   2000:TT=22:T6=256:Z=32:DE=29             :rem 63
```

```
40 GOTO7000                                          :rem 100
100 GL=3:SC=.                                        :rem 207
105 EG=2000                                           :rem 33
110 POKE36869,255                                    :rem 153
120 PRINT"{CLR}{11 RIGHT}{18 DOWN}{GRN}@{LEFT}
    {DOWN}{WHT}A{2 LEFT}{DOWN}CB{3 LEFT}{DOWN}FED
    {3 LEFT}{DOWN}IHG{4 LEFT}{DOWN}{RED}JKKKL";
                                                     :rem 84
130 POKE646,10:PRINT"{4 LEFT}KKK":PRINT"{CYN}]]]]]
    ]]]]]]]]]]]]]]]]{LEFT}{INST}]{HOME}{3 DOWN}":P
    OKEV,15+16*9                                     :rem 181
140 POKE36879,238                                    :rem 158
150 PRINT"{HOME}{6 DOWN}{2 RIGHT}"CL$"{7 UP}
    {4 RIGHT}"CL$"{3 DOWN}{3 RIGHT}"CL$;             :rem 235
160 PRINT"{YEL}{8 UP}%&]]{4 LEFT}{DOWN}'()]
    {3 LEFT}{DOWN}*+,{2 LEFT}{DOWN}-."               :rem 252
170 PRINT"{HOME}{BLK}/"SC;:POKE646,8:PRINT"{HOME}
    {DOWN}";:IF GL>1THENFORT=1TOGL-1:PRINT"S";:NEX
    T                                                :rem 44
180 LO=8128:POKELO+CO,10:POKELO,18:POKELO-TT,17:PO
    KELO-TT+CO,10:POKELO-1,15                        :rem 175
190 POKELO-1+CO,.:CL=LO-1:POKE8132+CO,10:POKE8110+
    CO,10:POKE8133+CO,.                              :rem 37
200 TY=FNRN(2)+1:ONTYGOTO210,300                     :rem 158
210 X=FNRN(2)+1:ONXGOTO220,230                       :rem 248
220 BC=8142:EC=8149:SP=1:DD=21:GOTO240               :rem 176
230 BC=8163:EC=8155:SP=-1:DD=22                      :rem 214
240 FORDL=BCTOECSTEPSP:POKEDL,DD:POKEDL+CO,.
                                                     :rem 150
250 GOSUB1000:POKEDL,Z:NEXT:IFSD<>196THEN3000
                                                     :rem 60
260 SD=.:GOTO200                                     :rem 159
300 NU=.:Y=59:C=6:X=FNRN(2)+1:ONXGOTO310,320
                                                     :rem 110
310 B=7776:E=8084:GOTO330                            :rem 135
320 B=7758:E=8088                                    :rem 131
330 FORDL=BTOESTEPTT:POKEDL+CO,C:POKEDL,Y:GOSUB100
    0                                                :rem 118
340 POKEDL,Z:NEXT:IFSP<>T6THEN3000                   :rem 183
350 SP=.:GOTO200                                     :rem 171
1000 CK=PEEK(197):IFCK=64THENFORR=0TODE:NEXT:RETUR
     N                                                :rem 53
1010 IFCK=29ANDLO=8132THEN1500                       :rem 245
1020 IFCK=37ANDLO=8128THEN1750                        :rem 1
1030 IFCK=ZTHENONTYGOTO2000,3500                     :rem 224
1040 RETURN                                          :rem 165
1500 POKELO,Z:POKELO-TT,Z:POKECL,Z:LO=8128:CL=LO-1
                                                     :rem 215
```

```
1510 POKELO,18:POKELO-TT,17:POKECL,15:FORSD=130TO1
     50STEP2:POKES2,SD:NEXT:POKES2,.          :rem 191
1520 RETURN                                    :rem 168
1750 POKELO,Z:POKELO-TT,Z:POKECL,Z:LO=8132:CL=LO+1
                                               :rem 215
1760 POKELO,20:POKELO-TT,19:POKECL,13          :rem 29
1770 FORSD=150TO130STEP-2:POKES2,SD:NEXT:POKES2,.:
     RETURN                                    :rem 126
2000 POKECL,PEEK(CL)+1:FORSD=250TO200STEP-10:POKEN
     O,SD:NEXT:IFPEEK(CL+TT)=DDTHEN2100        :rem 50
2010 POKECL,PEEK(CL)-1:POKENO,.:RETURN         :rem 138
2100 SC=SC+75:PRINT"{HOME}{BLK}/"SC:GOSUB4000
                                               :rem 180
2110 POKEDL,58:FORSD=254TO198STEP-2:POKEDL+CO,FNRN
     (8):POKENO,SD:NEXT                        :rem 109
2120 POKENO,.:GOTO2010                         :rem 33
3000 GL=GL-1:FORT=130TO254STEP2:POKES,T:POKENO,T:P
     OKEV,15+FNRN(16)*16:NEXT                  :rem 170
3010 POKELO-TT,218:FORT=15TO0STEP-.2:POKES,.:POKEN
     O,160:POKEV,T+FNRN(16)*16:NEXT            :rem 29
3020 IFGL=.THEN7000                            :rem 72
3030 POKENO,.:GOTO120                          :rem 242
3500 IFNU>2THENRETURN                          :rem 121
3510 NU=NU+1:POKELO-44,60:POKELO-44+CO,4:FORSD=150
     TO180STEP10:POKES2,SD:NEXT:POKES2,.       :rem 95
3520 IFPEEK(LO-66)=59THEN3600                  :rem 166
3530 POKE(LO-44),Z:RETURN                      :rem 27
3600 SC=SC+50:PRINT"{HOME}{BLK}/"SC:GOSUB4000:POKE
     LO-66,61                                  :rem 227
3610 FORSP=200TO254STEP2:POKES,SP:NEXT:POKES,.
                                               :rem 233
3620 POKELO-66,Z:RETURN                        :rem 206
4000 IFSC>=EGTHENGL=GL+1:DE=DE-4:EG=EG+2000:POKE77
     00+GL,19:POKE7700+CO+GL,8:GOTO4020        :rem 30
4010 RETURN                                    :rem 165
4020 FORT=130TO230STEP10:FORR=T+10TOTSTEP-1:POKES,
     T:NEXTR,T:POKES,.:RETURN                  :rem 127
7000 POKE36869,240:PRINTCHR$(8):IFSC>HSTHENHS=SC
                                               :rem 238
7010 POKE36879,8:POKE646,10:PRINT"{CLR}{3 SPACES}
     {A}**{S}":PRINT"{3 SPACES}-{A}*{X}"       :rem 147
7020 PRINT"{3 SPACES}-{Z}{R}*{R}*{R}**{R}**{2 R}*
     {S}{6 SPACES}-{A}{W}{A}{W}{D}-{A}{S}{Q}{S}{A}
     {W}-{A}{X}{6 SPACES}-----{SHIFT-SPACE}-------
     -{Z}{S}"                                  :rem 2
7030 PRINT"{A}**+{X}{Q}{W}{Z}{2 E}{X}{Z}{X}{Z}{X}
     {Z}{E}*{X}":PRINT"-{A}*{X} --"            :rem 11
```

```
7040 PRINT"-{Z}{2 R}*{W}{Z}*{R}*{R}*{R}**{R}*{R}**
     {S} -{A}{W}-*{W}{A}{S}-{V}-{A}{W}{2 SPACES}-
     {D}-{A}{S}-"                                  :rem 60
7050 PRINT"---{Q}*----*{W}----- ---- {Z}{X}{Z}{E}*
     {E}{X}{Z}{E}*{E}{X}{Z}{5 E}{X}{Z}{X}" :rem 22
7060 PRINT"{DOWN}{RED}{2 SPACES}LAST SCORE:"SC:PRI
     NT"{DOWN}{GRN}{2 SPACES}HIGH SCORE:"HS
                                                   :rem 246
7070 PRINT"{PUR}{DOWN}{2 SPACES}HIT A KEY TO PLAY"
                                                   :rem 52
7080 PRINT"{RVS}{WHT}{7 SPACES}CONTROLS{7 SPACES}
     {OFF}{PUR}{4 SPACES}<-LEFT":PRINT"{GRN}
     {4 SPACES}>-RIGHT"                            :rem 255
7090 PRINT"{RVS}{BLU}SPACE{OFF}-CLUB OR UMBRELLA":
     POKENO,.                                      :rem 120
7100 POKE36878,(FNRN(14)+2)*16:IFPEEK(197)=64THEN7
     100                                           :rem 218
7110 GOTO100                                       :rem 147
```

3
Educational Games

Educational Games

Your VIC-20 never tires of quizzes or drills, and that makes it an excellent teacher. The teaching games in this chapter turn your VIC into a sophisticated educational tool, making learning more fun than ever before.

For example, Janet Arnold's "Tree Tutor for Tots" uses lively, colorful graphics displays to introduce young children to addition. Soori Sivakumaran's "Snertle," aimed at elementary-age students, also teaches subtraction and multiplication. It features an easy-to-use menu, making it easy for children to control the program themselves.

"Alpha-Shoot," by Neil T. Capaldi, turns learning the alphabet into an adventure in galactic defense. There are several play options, so the game will hold a child's attention through repeated sessions of play.

Mike Salman's "Word Scramble" lets two players (or teams) compete to unscramble words of up to ten letters. It's an excellent way for anyone, young or old, to learn new words.

Your VIC can help you learn new skills, too. For instance, "Typing Derby," by Carlos Esteves, lets you develop touch-typing skills as you race toward a finish line.

Tree Tutor for Tots

Janet Arnold

This educational program uses custom characters and lively graphics to teach addition to young children. Correct answers are rewarded; there are no penalties for guessing wrong. It is written for the unexpanded VIC.

Arithmetic is for the birds—but only if your youngster plays "Tree Tutor for Tots." This math program is suitable for small children who are just learning to add. It is a tutor, not simply a drill, since it illustrates addition concepts using colorful, attention-getting graphics.

The screen shows a tree, some apples in the tree, and other apples on the ground. The child adds the apples hanging in a tree to those scattered on the ground, typing in the correct answer. The problem is also shown in more traditional form to the right of the tree.

A correct answer brings a bird swooping from the sky to pluck an apple from the tree. The bird then drops it into a basket and flies off the screen. After ten right answers—that is, when ten apples have been stacked in the basket—the game ends.

Choosing Levels of Play

LOAD the program and RUN it. After a short wait, the title appears and you are asked to "Choose highest sum (2–9)." Hitting a 7, for instance, generates problems with answers no higher than seven. Choose 2 at first, proceeding to the harder problems as the easier ones are mastered.

Next, you are given a choice of options for displaying the fruit. At first, you should select option 1; this tells the computer to show the apples when the problem is first printed. Selecting option 2 makes the fruit appear only if the child gives a wrong answer.

Once the tree and the problem are displayed, guide your child to discover the correct answer by saying something like, "There are two apples in the tree and one more on the ground. See this problem? It says 2 plus 1. How much is two plus one? Let's count the apples and find out."

Point out that the number of apples in the tree is the same as the top number of the combination, and that the number of apples on the ground matches the bottom number. Your child will learn that the apples are a *picture* of the addition problem.

When you think your youngster is ready, suggest trying to answer without counting the apples (that is, using option 2). If the given answer is wrong, the apples will appear on the screen and your child can count them to discover the correct sum.

An apple is dropped into the basket for every right answer, even after several incorrect guesses, as an incentive to keep trying. After collecting ten apples, you receive a message stating the total tries (although a preschooler probably won't care). The child will, however, enjoy seeing the bird fly down to land on the message, which is a further incentive to complete ten problems.

Incorrect Keys Are Ignored

Because tots often hit the keyboard accidentally, lines 10, 14, and 78 accept only numerals within the stated range. All other keys will be unresponsive (except for the RUN/STOP key). The program uses a GET statement, so the child need not hit RETURN after entering an answer. Line 76 resets the number of characters in the keyboard buffer to zero, in case a key was pressed between problems.

Here is a brief description of how Tree Tutor works.

Line(s)	Description
2–6	Title, custom characters created, variables set.
8–14	GET highest number desired; GET fruit option.
16	POKE basket.
18	Main loop—count ten correct answers.
20–22	Choose problem (see paragraph following).
24	Erase former tree, problem, and message.
26–38	PRINT tree and problem.
40–74	POKE fruit.
76–80	GET and judge answer.
82–84	Routine for wrong answer.
86–106	Reward correct answer.
108–122	Reward ten correct answers; "play again" option.
124–126	Subroutine for falling apple.
128–138	Data for custom characters.

When the computer chooses an addition problem in lines 20–22, it first generates a random top number anywhere from one to the highest number family (F) selected by the user. The bottom number is never greater than F minus the top number, so the sum will never be greater than F. T1 and B1 hold the values of T and B, the top and bottom addends, from the last displayed problem. This is to insure that an identical problem does not follow immediately.

One oddity you will notice—my children discovered it right away—is that the apples in the tree are different from the apples elsewhere on the screen. The program POKEs the tree apples in multicolor mode, which causes some loss of horizontal resolution. This results in a boxier-looking apple, but it does fill in the empty spaces around the apples with green, the border color, rather than with white, the screen color.

My older son strongly dislikes seeing two shapes of apples. If this bothers you, too, change the first eight numbers of line 128 to read 240, 60, 255, 255, 255, 255, 255, and 60.

This program uses up most of the memory in an unexpanded VIC, so don't add any unnecessary spaces.

Tree Tutor for Tots

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
2 PRINT"{CLR}{8 DOWN}{RVS}{5 RIGHT}*{UP}{LEFT}*TRE
  E{RIGHT}TUTOR*{DOWN}{LEFT}*{DOWN}{12 LEFT}*
  {RIGHT}FOR{RIGHT}TOTS{RIGHT}*"              :rem 121
4 POKE36869,255:POKE52,28:POKE56,28:CLR:FORI=7168T
  O7679:POKEI,PEEK(I+25600):NEXT              :rem 97
6 FORI=7168TO7263:READN:POKEI,N:NEXT:POKE36879,29:
  V=36878:M=36876:C=30720                     :rem 173
8 X=0:PRINT"{CLR}{BLU}{RVS}{2 SPACES}CHOOSE HIGHES
  T SUM"SPC(10)"(2-9)"                        :rem 83
10 GETF$:F=VAL(F$):IFF<2ORF>9THEN10           :rem 113
12 PRINT"{CLR}{RVS}WHEN DO YOU WANT TO{3 SPACES}SE
   E FRUIT?{2 DOWN}{4 LEFT}(1) ALWAYS"SPC(12)"
   {DOWN}(2) IF WRONG"                        :rem 149
14 GETI$:I=VAL(I$):IFI<1ORI>2THEN14           :rem 128
16 PRINT"{CLR}":POKE8183,205:POKE8184,228:POKE8185
   ,206:FORB=38903TO38905:POKEB,10:NEXT       :rem 216
18 FORL=1TO10:Q=240:IFI$="2"THENI=2           :rem 60
20 T=INT(RND(.)*F)+1                          :rem 94
```

```
22 B=INT(RND(.)*((F+1)-T)):IFT=T1ANDB=B1THEN20
                                                :rem 166
24 PRINT"{HOME}{2 DOWN}";:FORY=1TO20:PRINT"
   {18 SPACES}":NEXT:PRINT"{9 SPACES}";     :rem 94
26 FORZ=38796TO38883:POKEZ,2:NEXT           :rem 66
28 PRINT"{HOME}{2 DOWN}{GRN}{3 SPACES}HFHFHF":PRIN
   T"{2 SPACES}HJJJJJJF":PRINT" HJJJJJJJJF":PRINT"
    GJJJJJJJJI"                             :rem 97
30 PRINT" HJJJJJJJJF":PRINT" GJJJJJJJJI":PRINT" HJ
   JJJJJJJF":PRINT" GJJJJJJJJI"             :rem 209
32 PRINT"{2 SPACES}GJJJJJJI":PRINT"{3 SPACES}GJJJJ
   I":PRINT"{4 SPACES}GJJI":PRINT"{5 SPACES}{BLK}J
   J{DOWN}{2 LEFT}JJ{DOWN}{2 LEFT}JJ{DOWN}{2 LEFT}
   JJ";                                     :rem 123
34 PRINT"{DOWN}{2 LEFT}JJ{DOWN}{2 LEFT}JJ{DOWN}
   {3 LEFT}HJJF"                             :rem 0
36 X=X+1:PRINT"{HOME}{8 DOWN}{16 RIGHT}{BLK}"T"
   {4 LEFT}{2 DOWN}+"B"{4 LEFT}{DOWN}{RVS}***":PRI
   NTSPC(17)"?{2 LEFT}";                    :rem 204
38 IFI=2THEN76                               :rem 78
40 POKE7751,11:POKE7751+C,10:IFT=1THEN58     :rem 84
42 POKE7860,11:POKE7860+C,10:IFT=2THEN58     :rem 89
44 POKE7885,11:POKE7885+C,10:IFT=3THEN58    :rem 106
46 POKE7775,11:POKE7775+C,10:IFT=4THEN58    :rem 105
48 POKE7815,11:POKE7815+C,10:IFT=5THEN58     :rem 98
50 POKE7820,11:POKE7820+C,10:IFT=6THEN58     :rem 84
52 POKE7903,11:POKE7903+C,10:IFT=7THEN58     :rem 91
54 POKE7840,11:POKE7840+C,10:IFT=8THEN58     :rem 94
56 POKE7928,11:POKE7928+C,10                 :rem 64
58 IFB=0THEN76                               :rem 71
60 POKE8086,0:IFB=1THEN76                   :rem 220
62 POKE8078,0:IFB=2THEN76                   :rem 224
64 POKE8123,0:IFB=3THEN76                   :rem 218
66 POKE8150,0:IFB=4THEN76                   :rem 221
68 POKE8106,0:IFB=5THEN76                   :rem 225
70 POKE8146,0:IFB=6THEN76                   :rem 223
72 POKE8126,0:IFB=7THEN76                   :rem 224
74 POKE8152,0                               :rem 198
76 POKE198,0                                :rem 154
78 GETA$:AN=VAL(A$):IFAN<1ORAN>9THEN78       :rem 93
80 PRINTAN:FORZ=1TO500:NEXT:IFAN=T+BTHEN86  :rem 17
82 PRINT"{RVS}{9 DOWN}TRY AGAIN";:POKEV,5:POKEM,23
   1:FORZ=1TO200:NEXT:POKEM,225              :rem 42
84 FORZ=1TO200:NEXT:POKEV,0:I=0:GOTO36      :rem 176
86 PRINT"{RVS}{9 DOWN}HOORAY!{2 SPACES}"SPC(7)L;:T
   1=T:B1=B:A=7700                          :rem 212
88 PRINT"{HOME}{2 SPACES}";:FORB=1TO3:PRINT"{BLK}
   {OFF}CAE{3 LEFT}";:FORZ=1TO75:NEXT:PRINT"BAD
   {3 LEFT}";                               :rem 147
```

```
90 FORZ=1TO75:NEXT:PRINT"{3 SPACES}{2 LEFT}{DOWN}"
   ;:NEXT                                          :rem 170
92 FORB=1TO2:PRINT" {RED}{UP}@{UP}{2 LEFT}{BLK}CAE
   {3 LEFT}";:IFB=2THENPOKE7730,6:POKE7730+C,5:GOT
   O96                                             :rem 251
94 POKE7751,10:POKE7751+C,5:POKE7728,6:POKE7728+C,
   5:POKE7729,8:POKE7729+C,5                       :rem 115
96 FORZ=1TO75:NEXT:PRINT"BAD{3 LEFT}";:FORZ=1TO75:
   NEXT:PRINT"{3 SPACES}{DOWN}{2 LEFT}";:NEXT:rem 5
98 FORB=1TO13:PRINT" {RED}@{UP}{3 LEFT} {BLK}CAE
   {3 LEFT}";:FORZ=1TO75:NEXT:PRINT"BAD{3 LEFT}";
                                                   :rem 158
100 FORZ=1TO75:NEXT:PRINT"{3 SPACES}{DOWN}{2 LEFT}
    ";:NEXT                                        :rem 210
102 PRINT"{UP}{LEFT}";:PRINT" CA{2 LEFT}";:GOSUB12
    4:PRINT"BA{2 LEFT}";:GOSUB124:PRINT" C{LEFT}";
    :GOSUB124                                      :rem 237
104 PRINT"B{LEFT}";:GOSUB124:PRINT" ";              :rem 23
106 FORB=1TO(15-L):POKEM,Q:POKEA,32:A=A+22:POKEA,0
    :POKEA+C,2:Q=Q-5:NEXT:POKEV,0:NEXT             :rem 251
108 PRINT"{HOME}{8 DOWN}{11 RIGHT}{11 SPACES}
    {DOWN}{10 LEFT}{RVS}{BLK}YOU GOT 10{DOWN}
    {10 LEFT}APPLES IN"                            :rem 240
110 PRINT"{RVS}{11 RIGHT}"X"TRIES.{DOWN}{4 LEFT}
    {2 SPACES}":FORZ=1TO300:NEXT                    :rem 34
112 PRINT"{HOME}{21 RIGHT}{BLK}U{LEFT}";:FORZ=1TO7
    5:NEXT:PRINT"B{LEFT}";:FORZ=1TO75:NEXT:rem 157
114 PRINT" {DOWN}{2 LEFT}CA{2 LEFT}";:FORZ=1TO75:N
    EXT:PRINT"BA{2 LEFT}";:FORZ=1TO75:NEXT:rem 144
116 FORB=1TO7:PRINT"{3 SPACES}{4 LEFT}{DOWN}CAE
    {4 LEFT}";:FORZ=1TO75:NEXT:PRINT"BAD{3 LEFT}";
    :FORZ=1TO75:NEXT                               :rem 215
118 NEXT:PRINT"{RVS}{6 DOWN}{LEFT}{BLU}HIT *{DOWN}
    {5 LEFT}TO PLAY{DOWN}{7 LEFT}AGAIN."            :rem 12
120 GETP$:IFP$<>"*"THEN120                         :rem 206
122 GOTO8                                           :rem 6
124 POKEV,9:FORB=1TO2:POKEM,Q:POKEA,32:A=A+22:POKE
    A,0:POKEA+C,2:Q=Q-5:FORZ=1TO15:NEXT             :rem 127
126 NEXT:RETURN                                    :rem 242
128 DATA24,8,106,255,255,255,126,52,60,126,187,199
    ,239,126,40,40                                 :rem 147
130 DATA0,0,0,15,31,48,96,192,240,120,12,7,3,0,0,0
    ,0,0,0,240,248,12,6,3                          :rem 137
132 DATA15,30,48,224,192,0,0,0,192,240,248,252,252
    ,254,255,255                                   :rem 18
134 DATA255,255,127,127,63,31,15,3,3,7,15,31,63,63
    ,127,255                                       :rem 84
136 DATA255,254,254,252,252,248,224,192,255,255,25
    5,255,255,255,255,255                          :rem 254
138 DATA245,105,170,170,170,170,170,105            :rem 169
```

Snertle

Soori Sivakumaran

By making simple selections from a menu, a child can change this arithmetic drill to fit his or her own tutoring needs. Written for the unexpanded VIC, it features a smiling turtle and bold graphics sure to catch the young child's eye.

"Snertle" is designed to help teach children the fundamentals of addition, subtraction, and multiplication. A turtle named Snertle is drawn on the screen to give encouragement and assistance to the player.

An Individual Challenge

Snertle allows children to tailor math problems to fit their individual abilities and weaknesses. Snertle first asks the child to select addition, subtraction, or multiplication problems. If addition or subtraction is selected, the child is then asked to choose the largest and smallest numbers to be used in creating the problems. The largest number that can be chosen is 99, and the smallest number is 0.

If multiplication is chosen, the child can decide to practice a specific multiplication table or solve problems created randomly using numbers from 0 through 14. For example, if the 12-times table is selected, one number in each question created will always be 12. The other number will be randomly selected from the range 0–14.

If the child chooses to attempt random multiplication problems, he or she must define the range of numbers (within the limits of 0 and 14) from which the problems can be created.

Creating the Screen

Once the necessary information is entered, the turtle's image is POKEd onto the screen. The two numbers used in the problem are chosen in lines 305, 315, and 1070. The numbers are then displayed on the screen, each digit being four regular characters high and three wide. The large character set is created in a series of subroutines in lines 500–990.

The larger number is always displayed above the smaller number to avoid negative answers to subtraction problems.

The appropriate sign for addition, subtraction, or multiplication is drawn on the screen by a subroutine beginning at line 6000. Next, a horizontal line is drawn under the numbers.

Line 394 contains a FOR-NEXT loop that clears the keyboard buffer. This prevents the child from accidentally entering data while the turtle and the problem are being put on the screen.

Another FOR-NEXT loop in lines 395–420 enters the user's response to the problem. Because a GET statement is used, the RETURN key does not have to be pressed when entering the response. An arrow will appear at the bottom of the screen to prompt for each digit of the response. Enter your answers with the ones digit first, followed by the tens digit, then the hundreds digit, just as though you were doing the calculation on a piece of paper.

The Smiling Turtle

Once a response is entered, Snertle checks it against the correct answer. If the child's response is correct, the turtle will smile, GOOD! will appear on its shell, and a high beep will sound. If the response is incorrect, Snertle's head will disappear into his shell, and the message TRY AGAIN will appear on his side.

The child then gets a second chance. If the new response is correct, Snertle will poke his head out from his shell. If the answer is again incorrect, the correct answer will be displayed on the screen.

The program will keep producing problems until the X key is pressed in response to a problem. The percentage of correctly answered questions is then calculated (in line 410) and displayed on the screen. The percentage only includes problems answered correctly on the first attempt. Snertle then returns to the menu, where the child may END the program or select more problems.

Snertle

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
100 A$=CHR$(147):B$=CHR$(17):C$=CHR$(29):D$=CHR$(1
    8):E$=CHR$(146):Y=160:LL=36876          :rem 62
110 PRINTA$SPC(5)B$B$"**SNERTLE**":POKELL+2,15
                                            :rem 181
120 PRINTB$B$B$B$C$C$ D$"SELECT ONE:"E$     :rem 119
130 PRINTB$"1) ADDITION"                    :rem 113
140 PRINTB$"2) SUBTRACTION"                 :rem 117
150 PRINTB$"3) MULTIPLICATION"               :rem 87
155 PRINTB$"4) END PROGRAM"                  :rem 30
160 PRINTB$"(ENTER 1,2,3 OR 4)";:INPUTQ:IFQ>4ORQ<0
    THEN160                                 :rem 102
185 C=14:IFQ=1ORQ=2THENC=99                 :rem 141
187 IFQ=3THEN1000                           :rem 224
188 IFQ=4THENEND                            :rem 248
190 PRINTA$B$B$"ENTER LARGEST VALUE"        :rem 169
200 PRINT"(MIN.:0 MAX.:";C;")";:INPUTR:IFR<0ORR>CT
    HEN200                                  :rem 142
230 PRINTB$B$"ENTER SMALLEST VALUE"         :rem 146
240 PRINT"(MIN.:0 MAX.:";R;")";:INPUTS:IFS<0ORS>RT
    HEN240                                  :rem 183
263 PRINTA$B$"PRESS "D$"X"E$" RETURN TO MENU":FORI
    =1TO750:NEXTI                            :rem 6
265 PRINTA$                                 :rem 143
270 Z=0:ZZ=0:GOSUB2000                       :rem 55
275 GOSUB1100:GOSUB1170:GOSUB1230:GOSUB1260
                                            :rem 102
301 TR=0:ZZ=ZZ+1                            :rem 226
305 L=INT(RND(1)*(R-S+1))+S                 :rem 234
310 IFQ=3ANDT=1THEN320                       :rem 61
315 K=INT(RND(1)*(R-S+1))+S                 :rem 234
320 F$=STR$(K):W=0                          :rem 243
325 IFK<LTHENW=110                           :rem 81
330 GOSUB3000                               :rem 217
335 W=110                                   :rem 193
337 IFL>KTHENW=0                            :rem 244
340 F$=STR$(L)                              :rem 248
345 GOSUB3000                               :rem 223
346 ONQGOSUB6000,6000,6004                  :rem 185
350 IFQ=1THENM=K+L                           :rem 97
355 IFQ=2ANDK>=LTHENM=K-L                    :rem 78
360 IFQ=2ANDK<LTHENM=L-K                     :rem 11
365 IFQ=3THENM=K*L                          :rem 104
380 GOSUB740:MM=1:IFM>9THENMM=2             :rem 189
385 IFM>99THENMM=3                          :rem 101
390 GOSUB740                                :rem 183
```

```
393 V=0:GOSUB1100                                   :rem 222
394 FORI=631TO640:POKEI,0:NEXTI                     :rem 180
395 FORJ=0 TO MM-1                                  :rem 218
397 POKE8177-(4*J),30                               :rem 94
400 GETH$                                           :rem 224
405 IFH$=""THEN400                                  :rem 216
407 IFH$="X"ANDZZ=1THEN100                          :rem 36
410 IFH$="X"THENPRINTA$"PERCENTAGE:";INT(Z/(ZZ-1)*
    100):GOTO120                                    :rem 10
412 FORO=8164TO8168:POKEO,32:NEXTO                  :rem 104
415 P=VAL(H$)                                       :rem 199
420 V=V+(P*10↑J):X=8110-(4*J):GOSUB480:NEXTJ
                                                    :rem 86
450 IFM=VTHEN470                                    :rem 210
451 POKELL,160:FORI=1TO500:NEXTI:POKELL,0           :rem 83
452 FORI=8098TO8186:POKEI,32:NEXTI                  :rem 96
456 IFTR=1THEN460                                   :rem 11
458 TR=1:GOSUB1500:GOSUB770:GOTO393                 :rem 159
460 M$=STR$(M)                                      :rem 3
461 FORI=1TO22-MM:READA:NEXTI                       :rem 96
462 FOROO=1TOMM                                     :rem 204
464 P=VAL(MID$(M$,(OO+1),1))                        :rem 243
465 READX:GOSUB480:NEXTOO:RESTORE                   :rem 222
470 GOSUB1230:IFTR=0THENGOSUB2500:GOSUB755:Z=Z+1:G
    OSUB6500                                        :rem 154
471 GOSUB2225:GOTO301                               :rem 238
480 IFP=0THENGOSUB720                               :rem 48
485 ONPGOSUB 500,525,555,585,610,633,660,680,700:R
    ETURN                                           :rem 254
500 FORI=0TO66STEP22:POKEX+I+1,Y:NEXTI:RETURN
                                                    :rem 211
525 GOSUB990:GOSUB980:POKEX+44,Y:GOSUB970:RETURN
                                                    :rem 102
555 GOSUB990:GOSUB980:POKEX+46,Y:GOSUB970:RETURN
                                                    :rem 107
585 POKEX,Y:POKEX+22,160                            :rem 193
595 FORI=44TO46:POKEI+X,Y:NEXTI                     :rem 1
600 POKEX+23,118:POKEX+67,118:RETURN                :rem 172
610 GOSUB990                                        :rem 185
620 POKEX+22,Y:POKEX+23,98:POKEX+24,98:POKEX+46,Y:
    GOSUB970:RETURN                                 :rem 95
633 GOSUB990                                        :rem 190
640 POKEX+22,Y:POKEX+23,98:POKEX+24,98              :rem 18
645 POKEX+44,Y:POKEX+46,Y:GOSUB970:RETURN           :rem 141
660 GOSUB990                                        :rem 190
670 POKEX+24,Y:POKEX+45,Y:POKEX+46,97:POKEX+67,Y:R
    ETURN                                           :rem 254
680 GOSUB525                                        :rem 186
690 POKEX+22,Y:POKEX+46,Y:RETURN                    :rem 47
700 GOSUB680:POKEX+44,32:RETURN                     :rem 180
```

```
720 GOSUB680:POKEX+23,32:RETURN                    :rem 179
740 FORI=8080TO8093:POKEI,64:NEXTI:RETURN          :rem 115
755 POKE7753,7:POKE7754,15:POKE7755,15:POKE7756,4:
    POKE7757,33                                    :rem 37
760 POKE7753,7:POKE7754,15:POKE7755,15:POKE7756,4:
    POKE7757,33:RETURN                             :rem 59
770 POKE7732,20:POKE7733,18:POKE7734,25            :rem 209
780 POKE7753,1:POKE7754,7:POKE7755,1:POKE7756,9:PO
    KE7757,14:POKE7758,33                          :rem 147
785 FORI=1TO750:NEXTI:RETURN                       :rem 93
960 FORI=0TO66STEP22:POKE I+X,160:NEXTI:RETURN
                                                   :rem 191
970 FORI=0TO2:POKEI+66+X,160:NEXTI:RETURN          :rem 125
980 POKEX+22,98:POKEX+23,98:POKEX+24,160:RETURN
                                                   :rem 113
990 FORI=0TO2:POKEX+I,160:NEXTI:RETURN             :rem 232
1000 PRINTA$B$B$SPC(2)"DO YOU WISH TO:"            :rem 212
1010 PRINTB$SPC(3)"1) PRACTICE TIMES"              :rem 138
1015 PRINT"TABLES"                                 :rem 83
1020 PRINTB$SPC(3)"2) RANDOM NUMBERS"              :rem 156
1030 PRINT"(ENTER 1 OR 2)";:INPUTT:IFT<0ORT>2THEN1
     030                                           :rem 162
1050 IFT=2THENGOTO190                              :rem 26
1060 PRINTA$B$B$SPC(2)"ENTER TIMES TABLE"          :rem 154
1070 PRINTB$SPC(3)"(1-14)";:INPUTK:IFK<1ORK>14THEN
     1070                                          :rem 212
1090 S=0:R=14:GOTO263                              :rem 198
1100 FORI=7702TO7790STEP22                         :rem 25
1110 READA:READB                                   :rem 184
1120 FORJ=1TOB                                     :rem 72
1130 POKE (I+A+J),102                              :rem 46
1140 NEXTJ:NEXTI:RESTORE:RETURN                    :rem 137
1170 FORI=1TO11                                    :rem 108
1180 POKE(7815+I),120                              :rem 82
1190 NEXTI                                         :rem 83
1200 POKE7793,74                                   :rem 99
1210 RETURN                                        :rem 164
1230 FORI=1TO10:READA:NEXTI                        :rem 193
1232 FORI=7724TO7768STEP 22                        :rem 40
1234 FORJ=15TO17                                   :rem 169
1235 READA:POKEI+J,A:NEXTJ:NEXTI:RESTORE:RETURN
                                                   :rem 185
1260 FORI=1TO2                                     :rem 60
1270 POKE7817+I,Y:POKE7821+I,Y:NEXTI               :rem 191
1300 FORI=1TO3                                     :rem 56
1310 POKE7839+I,Y                                  :rem 200
1320 POKE7843+I,Y                                  :rem 196
1330 NEXTI:RETURN                                  :rem 105
1500 FORI=7724TO7768STEP 22                        :rem 38
```

```
1510 FORJ=15TO17:POKEI+J,32:NEXTJ:NEXTI:RETURN
                                              :rem 253
2000 FORI=38400TO38575                        :rem 221
2001 POKEI,5:NEXTI                             :rem 94
2003 POKE38482,6:FORI=38576TO38905:POKEI,1+Q:NEXTI
     :RETURN                                   :rem 38
2225 FORI=7878TO8185:POKEI,32:NEXTI:RETURN:rem 174
2500 POKE7785,202:RETURN                      :rem 171
3000 IFLEN(F$)>2THEN3030                       :rem 81
3015 P=VAL(MID$(F$,2,1))                      :rem 254
3020 X=7890+W:GOSUB480                         :rem 10
3025 RETURN                                   :rem 170
3030 P=VAL(MID$(F$,2,1))                      :rem 251
3035 X=7886+W:GOSUB480                         :rem 21
3040 P=VAL(MID$(F$,3,1))                      :rem 253
3045 X=7890+W:GOSUB 480                        :rem 17
3050 RETURN                                   :rem 168
5000 DATA 6,5,5,7,4,9,3,11,3,11,233,160,160,160,10
     8,160,160,160,160,8102,8106,8110         :rem 159
6000 POKE8015,Y:POKE8036,Y:POKE8037,Y:POKE8038,Y:P
     OKE8059,Y                                 :rem 76
6002 IFQ=2THENPOKE8015,32:POKE8059,32         :rem 164
6003 RETURN                                   :rem 169
6004 POKE8014,Y:POKE8016,Y:POKE8037,Y:POKE8058,Y:P
     OKE8060,Y:RETURN                          :rem 97
6500 POKELL,207:FORI=1TO150:NEXTI:POKELL,215:FORI=
     1TO175:NEXTI:POKELL,0:RETURN              :rem 64
```

Alpha-Shoot

Neil T. Capaldi

Educational games should, by definition, be educational. But they should also be challenging, visually stimulating, and fun. "Alpha-Shoot" is just such a game, designed to teach the alphabet and letter recognition to young children. Written for the unexpanded VIC, it can be used with either joystick or keyboard control.

"Alpha-Shoot" is a game I wrote to help my son learn and recognize the letters of the alphabet.

The object of the game is to line up the heart-shaped character at the bottom of the screen with the letter displayed above. The heart can be moved left or right with the C and B keys or with the joystick. Pressing the space bar or joystick fire button launches an arrow toward the top of the screen.

Whenever a letter is hit, it explodes and is placed in alphabetic order at the bottom of the screen. When all the letters in the alphabet have been captured in this way, the game then displays the complete alphabet to the familiar children's tune of "Twinkle Twinkle Little Star."

Four Games in One

Alpha-Shoot offers four possible play options. When you first run the program, it asks you to choose one of the four. Option 1 displays the letters of the alphabet at random, while in Option 2, letters are displayed in alphabetic order beginning with A. Option 3 displays a letter selected from the keyboard. Option 4 moves random letters across the screen.

Parents should select the variation they want and have the child name each letter as it appears on the screen. Children may also want to sing along with the alphabet song (to the tune of "Twinkle Twinkle Little Star") to further reinforce the learning.

Alpha-Shoot

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
5 PRINT"{CLR}":Y=7900:SV=36878:SS=36876:CL=36879:P
  OKECL,78:POKESV,14                              :rem 225
7 H=8108:CL=30720:J=37137:POKE650,128:POKE651,1
                                                  :rem 130
8 DIM AB(26):FORX=1TO26:AB(X)=32:NEXTX             :rem 25
10 GOSUB200                                       :rem 115
12 GOSUB40                                         :rem 71
14 GOSUB215:FORX=8120TO8141:POKEX,67:POKEX+30720,3
   :NEXTX                                         :rem 225
16 GOSUB250:GOSUB265                              :rem 213
30 GOSUB435:GOTO16                                 :rem 89
40 PRINT"{CLR}":RESTORE:Y=7900                    :rem 198
42 READL:IFL=0THEN90                              :rem 232
43 POKE Y,L                                       :rem 103
50 POKEY,L:READ P:POKESS,P:READ D                  :rem 48
60 FORX=1TOD:NEXTX:POKESS,0                        :rem 54
70 Y=Y+1:FORX=1TO10:NEXTX:IFL=32THENY=Y-1         :rem 161
80 IFY=7922THENY=7952                             :rem 253
85 GOTO 42                                         :rem 12
90 READA$:IFA$="0"THEN100                         :rem 137
92 READP:READD                                    :rem 113
95 PRINT"{4 RIGHT}"A$:POKESS,P                    :rem 163
97 FORX=1TOD:NEXTX:POKESS,0:FORX=1TO10:NEXTX:PRINT
   "{HOME}":GOTO90                                :rem 246
100 FORX=1TO1500:NEXTX:RETURN                     :rem 146
170 DATA 1,135,310,2,135,310,3,175,310,4,175,310,5
    ,183,310,6,183,310                            :rem 239
173 DATA 7,175,615,8,163,310,9,163,310,10,159,310,
    11,159,310                                    :rem 131
175 DATA 12,147,120,13,147,120,14,147,120,15,147,1
    20,16,135,602                                  :rem 5
177 DATA17,175,310,18,175,310,19,163,601,20,159,31
    0                                              :rem 0
178 DATA21,159,310,22,147,601,23,175,121          :rem 221
179 DATA 32,175,121,32,175,231,24,163,601,25,159,3
    10,32,159,310                                 :rem 26
181 DATA 26,147,605,0                              :rem 7
183 DATA"NOW",135,310,"{4 RIGHT}I",135,310,"
    {6 RIGHT}KNOW",175,310                        :rem 11
184 DATA"{11 RIGHT}MY",175,310                    :rem 105
185 DATA"{DOWN}{4 RIGHT}A",183,310,"{DOWN}
    {5 RIGHT}B",183,310,"{DOWN}{6 RIGHT}C'S",175,6
    10                                            :rem 158
```

```
187 DATA"{2 DOWN}NEXT",163,310,"{2 DOWN}{5 RIGHT}T
    IME",163,310,"{2 DOWN}{10 RIGHT}WON'T",159,310
                                          :rem 107
189 DATA"{3 DOWN}YOU",159,310,"{3 DOWN}{4 RIGHT}SI
    NG",147,310,"{3 DOWN}{9 RIGHT}WITH",147,310
                                          :rem 247
190 DATA"{5 DOWN}{5 RIGHT}ME",135,630,"Ø" :rem 154
200 PRINTTAB(5):PRINT"ALPHA-SHOOT"        :rem 167
202 LE=-1:KR=Ø                             :rem 4
204 PRINT"{2 DOWN}WHICH GAME- 1,2,3,OR 4"  :rem 50
205 POKE198,Ø:WAIT198,1:GETA$:             :rem 235
206 IFA$="1"THEN211                        :rem 1
207 IFA$="2"THENLE=Ø:GOTO 211              :rem 116
208 IFA$="3"THENLE=1:GOTO 211              :rem 119
209 IFA$="4"THENLE=2:GOTO 211              :rem 122
210 GOTO205                                :rem 99
211 RETURN                                 :rem 116
215 R$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"        :rem 110
225 PRINT"{CLR}":POKEH,83:POKE36879,10:GOSUB228:PO
    KE7703+V,R:RETURN                      :rem 245
228 R=INT(LEN(R$)*RND(1)+1):P=ASC(MID$(R$,R,1))-64
                                           :rem 31
229 IFLE=1THEN232                          :rem 241
230 R$=LEFT$(R$,R-1)+RIGHT$(R$,LEN(R$)-R)  :rem 31
232 R=P:V=INT(RND(1)*350+1)                 :rem 3
236 IFLE=ØTHENKR=KR+1:R=KR                 :rem 144
237 IFLE=1THENWAIT198,1:GETB$:R=ASC(B$)-64:rem 251
238 IFR>26ORR<1THENR=1                     :rem 115
239 RETURN                                 :rem 126
250 POKE37139,Ø:X=(PEEK(37137)AND6Ø)/4      :rem 96
252 POKE37154,127:J=PEEK(37152)AND128:POKE37154,25
    5                                      :rem 110
255 IFX=11THEND=-1:GOSUB275                :rem 136
257 IFJ=Ø THEND= 1:GOSUB275                 :rem 29
259 IFX=7THENGOSUB3ØØ                      :rem 61
260 RETURN                                 :rem 120
265 GETA$:IFA$=""THENGOTO27Ø               :rem 146
266 IFA$="C"THEND=-1:GOSUB275              :rem 188
267 IFA$="B"THEND=+1:GOSUB275              :rem 186
268 IFA$=" "THENGOSUB3ØØ                   :rem 87
270 RETURN                                 :rem 121
275 X=H+D:IFX<8098ORX>8119THENRETURN        :rem 57
276 POKESS,130:POKEH,32:POKEX,83:H=X       :rem 229
278 POKESS,Ø:RETURN                        :rem 236
300 G=H:FORU=1TO19:G=G-22:IFPEEK(G)<>32THENPOKEG,3
    2:POKEG+22,32:GOTO350                  :rem 92
305 POKESS,U+220:POKEG,30:IFU>1THENPOKEG+22,32
                                           :rem 62
306 GOSUB435:NEXTU:POKESS,Ø:POKEG,32:RETURN:rem 73
```

```
350 POKESS,0:POKE36877,220:FORL=13TO0STEP-1:POKE36
    878,L:POKE36879,40:GOSUB375              :rem 208
355 NEXTL:POKE36877,0:POKE36878,14:GOSUB390:rem 92
357 POKE36879,10:GOSUB228:POKE7703+V,R:RETURN
                                              :rem 21
375 POKEG,90:POKEG+22,42:POKEG-22,42:POKEG+1,42:PO
    KEG-1,42                                 :rem 140
377 POKEG+23,77:POKEG-23,77:POKEG-21,78:POKEG+21,7
    8                                        :rem 209
379 POKEG,32:POKEG+22,32:POKEG-22,32:POKEG-1,32:PO
    KEG+1,32                                 :rem 136
381 POKEG-23,32:POKEG+23,32:POKEG-21,32:POKEG+21,3
    2:RETURN                                 :rem 192
390 AB(R)=R:FORX=1TO22:POKE8141+X,AB(X):POKE8141+X
    +30720,7                                 :rem 149
392 NEXTX:FORX=23TO26:POKE8150+X,AB(X):POKE8150+X+
    30720,7:NEXT                              :rem 42
394 FORX=1TO26:IFAB(X)=32THENRETURN          :rem 254
395 NEXTX:FORX=1TO26:AB(X)=32:NEXTX:POKE36879,78:F
    ORW=1TO1000:NEXTW:GOSUB40:RUN            :rem 127
435 IFLE<2THENRETURN                          :rem 57
436 Q=V+7703:IFPEEK(162)<41 THEN RETURN        :rem 5
440 IFQ>8074THENPOKEQ,32:V=2:RETURN          :rem 226
442 POKEQ,32:POKEQ+1,R:V=V+1:POKE162,0        :rem 28
445 RETURN                                   :rem 125
```

Word Scramble

Mike Salman

Who would have ever thought that scrambled words could be so much fun? "Word Scramble" lets you match wits with an opponent as you play against time. For two players and an unexpanded VIC.

"Word Scramble" is written for two players. The computer first asks you for the names of the players. It then instructs player one to enter a common word (maximum ten letters).

A Three-Minute Puzzle

When the word has been scrambled, player two presses the space bar to see the scrambled letters. The game allows three minutes to discover the word.

Elapsed time is shown at the top of the screen; the scrambled letters appear below it. Below the scrambled word is a horizontal bar, on which you type the first letter of the word. If you type the wrong letter you hear a buzz. Type the right one, however, and you'll hear a beep. The letter will also appear on the screen.

A Ten-Point Penalty

If you unscramble the letters within the allotted time and have made no wrong guesses, you are awarded 50 points. For every wrong guess that you make, you lose ten points. A scoreboard is displayed every second turn, so you'll be able to tell when both players have played an equal number of rounds.

Word Scramble

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
10 PRINT"{CLR}":POKE36879,8:PRINT"{RED}{7 DOWN}
   {5 RIGHT}WORD SCRAMBLE":POKE36878,15     :rem 98
20 GOSUB1000:POKE36879,27:PRINT"{CLR}"      :rem 80
25 PRINT"{RED}EACH PLAYER TAKING":PRINT"TURNS ENTE
   RS A COMMON"                             :rem 114
30 PRINT"{RED}WORD (MAX.10 LETTERS).";      :rem 222
35 PRINT"{RED}THE COMPUTER WILL THEN";:PRINT"SCRAM
   BLE THE WORD AND"                        :rem 139
40 PRINT"{RED}PRINT IT."                    :rem 169
```

```
45 PRINT"{RED}YOU HAVE THREE MINUTES";:PRINT"TO FI
   ND IT."                                    :rem 233
50 PRINT"{RED}IF FOUND WITHIN THE":PRINT"ALLOTTED
   {SPACE}TIME,YOU WILL";                     :rem 64
55 PRINT"{RED}BE GIVEN 50 POINTS.":PRINT"EVERY WRO
   NG GUESS THAT";                            :rem 221
60 PRINT"{RED}YOU MAKE WILL COST":PRINT"YOU 10 POI
   NTS.{BLU}"                                 :rem 114
65 PRINT:PRINT:PRINT:PRINT"{3 RIGHT}{RVS}{PUR}PRES
   S SPACE BAR{OFF}"                          :rem 246
70 GETA$:IFA$=""THEN70:C=0                    :rem 219
80 PRINT"{CLR}{4 DOWN}{GRN}PLAYER # 1'S NAME{BLU}"
   :INPUTP$(0)                                :rem 200
85 PRINT:PRINT:PRINT"{RED}PLAYER # 2'S NAME{BLU}":
   INPUTP$(1)                                 :rem 132
90 PRINT:PRINT"{DOWN}{PUR}"P$(C)",":PRINT"{RVS}
   {RED}ENTER WORD TO BE":PRINT"{RVS}SCRAMBLED:
   {OFF}{BLU}"                                :rem 216
92 W$="":INPUTW$:IFW$=""THENPRINT"{UP}";:GOTO92
                                              :rem 27
95 IFLEN(W$)>10THENPRINT"{RVS}{GRN}MORE THEN 10 LE
   TTERS!{OFF}{BLU}{7 UP}"::GOTO90            :rem 60
100 GOSUB200                                  :rem 163
110 GOSUB300                                  :rem 165
120 T(C)=T(C)+S(C)                            :rem 178
130 GOSUB400:FORI=1TO10:B$(I)="":NEXT         :rem 184
140 GOTO90                                     :rem 55
200 FORI=1TOLEN(W$)                           :rem 126
210 A$(I)=MID$(W$,I,1)                        :rem 107
220 NEXT                                      :rem 211
230 C$="":FORI=1TOLEN(W$)                     :rem 163
240 R=INT(RND(1)*LEN(W$)+1)                   :rem 248
250 IFB$(R)<>""THEN240                        :rem 178
260 B$(R)=A$(I)                               :rem 221
270 NEXT                                      :rem 216
271 FORI=1TOLEN(W$):C$=C$+B$(I):NEXT          :rem 111
272 IFC$=W$ANDLEN(W$)<>1THENFORI=1TOLEN(W$):B$(I)=
    "":NEXT:GOTO230                           :rem 201
275 PRINT"{CLR}{5 DOWN}{RVS}{7 RIGHT}{RED}WORD HAS
    {11 RIGHT}BEEN SCRAMBLED{OFF}{BLU}"       :rem 255
280 PRINT"{6 DOWN}{3 RIGHT}{GRN}PRESS SPACE BAR
    {9 RIGHT}WHEN READY{BLU}"                 :rem 223
285 GETC$:IFC$=""THEN285                      :rem 101
290 PRINT"{CLR}{5 DOWN}{7 RIGHT}";             :rem 90
295 FORI=1TOLEN(W$):PRINT"{RED}";B$(I);:NEXT
                                              :rem 162
298 RETURN                                    :rem 131
300 X=51:S(C)=50                               :rem 81
310 TI$="000000"                              :rem 246
```

```
320 PRINT:PRINT:PRINT:PRINT                          :rem 119
325 SC=7885:CC=38605                                  :rem 76
330 FORI=1TOLEN(W$)                                  :rem 130
335 POKESC,99:POKECC,2                                :rem 75
340 GETC$                                            :rem 222
350 PRINT"{HOME}{RVS}"MID$(TI$,4,1)" {OFF}MINUTES
    {2 SPACES}{RVS}"RIGHT$(TI$,2)" {OFF}SECONDS"
                                                      :rem 95
355 IFTI$="000300"THENGOSUB500:GOTO390               :rem 228
360 IFC$=""THEN340                                   :rem 214
365 PRINT"{4 DOWN}"                                  :rem 179
370 IFC$=A$(I)THENPRINTTAB(X)A$(I);:POKE36875,200:
    FORT=1TO100:NEXT:POKE36875,0:GOTO380             :rem 230
375 IFS(C)<10THENGOSUB550:GOTO390                     :rem 10
378 IFC$<>A$(I)THENS(C)=S(C)-10:POKE36877,220:FORT
    =1TO100:NEXT:POKE36877,0:GOTO335                 :rem 131
380 X=X+1:SC=SC+1:CC=CC+1:NEXT                        :rem 59
390 RETURN                                           :rem 124
400 IFC<>1THENC=1:RETURN                              :rem 11
410 PRINT"{CLR}{5 DOWN}{7 RIGHT}{RED}{RVS}SCORES
    {OFF}{BLU}"                                      :rem 199
420 PRINT"{7 RIGHT}[6 T]"                              :rem 4
430 PRINT"{DOWN}"P$(0),P$(1)                         :rem 144
440 PRINTT(1),T(0)                                   :rem 252
450 C=0:RETURN                                        :rem 99
500 PRINT"{CLR}{4 DOWN}{3 RIGHT}{RVS}{RED}YOUR TIM
    E IS UP.{OFF}{BLU}"                               :rem 96
510 PRINT"{2 DOWN}{2 RIGHT}WORD WAS:"W$".":S(C)=0
                                                      :rem 159
520 FORT=1TO5000:NEXT:RETURN                          :rem 59
550 PRINT"{RVS}{RED}{2 DOWN}YOU RAN OUT OF POINTS.
    {OFF}{BLU}"                                      :rem 226
560 PRINT"{2 DOWN}{PUR}WORD WAS:{BLU}"W$"."
                                                      :rem 151
570 FORT=1TO2000:NEXT                                 :rem 35
580 RETURN                                           :rem 125
1000 FORS=250TO235STEP-1:POKE36874,S:POKE36878,S-2
     35:FORT=1TO100:NEXTT,S                          :rem 188
1010 POKE36874,0:POKE36878,15:RETURN                 :rem 126
```

Typing Derby

Carlos Esteves

If your keyboard style is hunt and peck, you need "Typing Derby." Its exciting drills make a game out of learning the typewriter keyboard, and it can help make you a smooth touch-typist. For the VIC with at least 3K memory expansion.

You can acquire lots of good software by typing in program listings from books and magazines. However, for us two-fingered typists, typing in a long program can be a slow process. That's why I decided to enlist the help of the computer to improve my typing. Having three children who are already dealing with the keyboard and who will eventually work with word processors or typewriters gave me another reason to write a typing tutor. But it also called for a program with game-style features that would appeal to them and take some of the drudgery out of typing practice. That's how "Typing Derby" came to be.

Racing the Computer

In Typing Derby, players race a red horse against the computer's black horse by correctly typing—without looking at the keyboard—the exercises displayed at the bottom of the screen. Each finger is assigned a range of keys, and there are 13 levels of difficulty. When you have won against the black horse 23 times, earning 230 points, you move up to the next level.

At first the pace is slow, allowing each finger to get the feel of the keys. But every time your red horse wins, the black horse runs faster in the next race. While it is possible to type faster, make mistakes, and still win handily, it is better to win a close race with no mistakes. At the end of each level you will need the typing speed but cannot afford the mistakes.

Brief Program Description

Here's what the different lines do:

Line(s)	Description
2–21	Contain initialization, opening, and closing routines.
22–80	Set the screen for the beginning of each race, including the "call to the gate" and the text of the

	corresponding exercise. (The horses and their colors are POKEd while everything else, except colons and commas, is PRINTed).
90–170	Control the development of the race.
200–290	DATA statements. Each line corresponds to a level of difficulty and contains the text of an exercise.
300–390	The sound subroutine.
401–432	Instructions.

The number of points required to move on to the next level (230) is, of course, arbitrary. Since the purpose of Typing Derby is to practice at the keyboard, it does not seem to be too high. However, it can be changed by varying the value 220 in line 21 and by adjusting the value of R (lines 10 and 21), which controls the speed of the black horse.

The number of exercises is limited only by the available memory. Any book on touch-typing can be used to provide exercises. Just remember that colons and commas cannot be part of items in the DATA statements. They have to be POKEd directly into screen memory (lines 52–55).

Typing Derby

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
2 PRINTCHR$(147)"{5 RIGHT}{5 DOWN}{RVS}{RED}TYPING
   DERBY{OFF}":PRINT"{6 DOWN}{RIGHT}INSTRUCTIONS?
  {SPACE}(Y=YES)                                  :rem 32
4 GETA$:IFA$=""THEN4                             :rem 139
6 IFA$="Y"THENGOSUB401:PRINT"{CLR}"              :rem 230
8 PRINT"{HOME}{15 DOWN}{RIGHT}ENTER LEVEL:(1TO13)"
  :INPUTL1:IFL1<1ORL1>13THEN4                     :rem 14
10 DIMC(3):C(0)=38488:C(1)=38554:C(2)=38620:H=7768
   :H1=7790:J=8010:R=32                           :rem 235
11 L=L1:S=(220*(L1-1))+(10*-(L1>1)):GOTO20:rem 179
16 PRINT"{CLR}{3 DOWN}{3 RIGHT}CONT'(Y/N)?":rem 22
17 GETX$:IFX$=""THEN17                           :rem 33
18 IFX$="N"THENEND                               :rem 73
20 IFS=2880THENPRINT"{4 RIGHT}THE END":END:rem 144
21 M=0:N=0:N1=0:IFS>220*LTHENL=L+1:L1=L1+1:R=32
                                                 :rem 247
22 PRINTCHR$(147):POKE36879,219:PRINT"{BLK}SCORE:
   {RVS}"S"{OFF} LEVEL{RVS}"L1"{OFF}"            :rem 138
30 FORI=0TO3:PRINT"{DOWN}{22 R}":NEXT:PRINT"{3 UP}
   {2 LEFT}{RED}{*}{DOWN}{LEFT}{M}"              :rem 255
```

```
40 PRINT"{10 DOWN}{5 RIGHT}{RVS}TYPING DERBY{OFF}
   {BLK}";                                          :rem 48
50 FORI=1TOL:READD$:NEXT:RESTORE:PRINT"{HOME}
   {15 DOWN}{BLU}"D$                                :rem 43
52 IFL=6THENPOKE8015,44:POKE8067,44                 :rem 71
53 IFL=7THENPOKE8023,44                             :rem 118
54 IFL=8THENFORI=0TO6STEP2:POKE8010+I,58:NEXT
                                                    :rem 132
55 IFL=8THENPOKE8031,44:POKE8070,58:POKE8072,58
                                                    :rem 28
56 IFL=9THENPOKE8046,58                             :rem 133
60 FORI=0TO2:FORT=0TO21:POKEC(I)+T,2:NEXTT:NEXTI:P
   OKEH,94                                          :rem 188
70 FORI=0TO2:FORT=0TO21:POKEC(I)+22+T,0:NEXTT:NEXT
   I:POKEH1,94                                      :rem 123
80 IFN=0ANDM=0ORS=(220*L)+10THENGOSUB300            :rem 153
90 IFM=21ORM=87THENM1=M:M=M+44                      :rem 90
100 IFPEEK(H1+M+1)<>32THEN16                        :rem 88
110 IFTI>T+RTHENPOKEH1+M1,32:POKEH1+M,32:M=M+1:POK
    EH1+M,94:T=TI                                   :rem 145
120 GETA$:IFA$=""THEN90                             :rem 31
130 IFASC(A$)=PEEK(J+N1)THEN150                     :rem 28
140 IFASC(A$)<>PEEK(J+N1)+64THEN90                  :rem 194
150 POKE38730+N1,2:N1=N1+1:IFN=21ORN=87THENN2=N:N=
    N+45                                            :rem 214
160 POKEH+N,32:POKEH+N2,32:N=N+1:IFPEEK(H+N)<>32TH
    ENS=S+10:R=R-1:GOTO16                           :rem 68
170 POKEH+N,94:GOTO90                               :rem 253
200 DATAFRF FTF FGF FBF FVF FRF FTF FGF FBF FVF FR
    F FTF FGF FBF FVF FR                            :rem 7
210 DATADED DCD FRF FTF FGF FBF FVF DED DCD FRF FT
    F FGF FBF FVF DED DC                            :rem 179
220 DATASWS SXS DED DCD FRF FTF FGF FBF FVF SWS SX
    S DED DCD FRF FTF FG                            :rem 96
230 DATAAQA AZA SWS SXS DED DCD FRF FTF FGF AQA AZ
    A SWS SXS DED DCD FR                            :rem 92
240 DATAJUJ JYJ JHJ JNJ JMJ AQA AZA SWS SXS DED DC
    D JUJ JYJ JHJ JNJ JM                            :rem 146
250 DATAKIK KIK JUJ JYJ JHJ JNJ JMJ AQA AZA SWS SX
    S DED DCD KIK KIK FR                            :rem 124
260 DATALOL L.L KIK KIK JUJ JYJ JHJ JNJ JMJ AQA AZ
    A SWS SXS FTF LOL L.                            :rem 125
270 DATA;P; ;/; LOL L.L KIK KIK JUJ JYJ JHJ JNJ JM
    J AQA AZA SWS SXS ;P                            :rem 51
280 DATAA11 S22 D33 F44 F55 J66 J77 K88 L99 ;00 Z1
    1 X22 C33 V44 V55 N6                            :rem 187
281 DATAIF IF IF{2 SPACES}IT IT IT{2 SPACES}IS IS
    {SPACE}IS TIME TIME TIME IF IT IS TIME IF I
                                                    :rem 105
```

```
282 DATAWE WE WE{2 SPACES}CAN CAN CAN{2 SPACES}PLA
    Y PLAY PLAY WE WE WE CAN CAN CAN PLA  :rem 201
287 DATATHAT LITTLE BROWN FOX QUICKLY RUNS AND JUM
    PS OVER THE LAZY DOG                  :rem 50
290 DATATHIS RACE WILL END THE GAME; IF YOUR TYPIN
    G DOES NOT FAIL. BYE                  :rem 76
300 V=36878:S2=36875:POKEV,15:POKES2,173:FORX=1TO1
    50:NEXT:POKEV,0                       :rem 249
330 POKEV,15:POKES2,194:FORX=1TO150:NEXT:POKEV,0
                                          :rem 25
340 POKEV,15:POKES2,206:FORX=1TO150:NEXT:POKEV,0
                                          :rem 20
350 FORI=0TO2:POKEV,15:POKES2,214:FORX=1TO150:NEXT
    :POKEV,0:NEXT                         :rem 57
360 FORI=0TO2:POKEV,15:POKES2,206:FORX=1TO150:NEXT
    :POKEV,0:NEXT                         :rem 59
370 POKEV,15:POKES2,194:FORX=1TO150:NEXT:POKEV,0
                                          :rem 29
380 POKEV,15:POKES2,206:FORX=1TO150:NEXT:POKEV,0
                                          :rem 24
385 POKEV,15:POKES2,194:FORX=1TO150:NEXT:POKEV,0
                                          :rem 35
390 POKEV,15:POKES2,173:FORX=1TO1800:NEXT:POKEV,0:
    RETURN                                :rem 105
401 PRINTCHR$(14)CHR$(147)"{5 RIGHT}{RVS}{RED}TYPI
    NG DERBY{OFF}{BLU}"                   :rem 204
402 PRINT"{2 RIGHT}BASIC TOUCH TYPING{4 SPACES}TUT
    OR":PRINT"{RVS}{DOWN}INSTRUCTIONS{OFF}:"
                                          :rem 174
403 PRINT"{RVS}{DOWN}1{OFF}.LEARN FINGERS' RANGE O
    N THE KEYBOARD."                      :rem 13
404 PRINT"{RVS}2{OFF}.PLACE FINGERS ON THE'HOME KE
    YS'.WRISTS LE-VEL,FINGERS SLIGHTLY";  :rem 181
405 PRINT"{2 SPACES}ARCHED,PALMS OFF VIC." :rem 37
406 PRINT"{RVS}3{OFF}.TYPE THE EXERCISES{2 SPACES}
    WITHOUT LOOKING AT THE KEYBOARD."     :rem 250
407 PRINT"{RVS}4{OFF}.AT FIRST,ACCURACY IS BETTER
    {SPACE}THAN SPEED."                   :rem 178
408 PRINT"{RVS}5{OFF}.BEAT THE BLACK HORSE 23 TIME
    S AND MOVE ON TO THE NEXT LEVEL."     :rem 198
409 PRINT"{DOWN}PRESS A KEY TO GO ON "    :rem 140
410 GETA$:IFA$=""THEN410                  :rem 77
411 PRINTCHR$(147)CHR$(142)"{RVS}{RED}{5 RIGHT}
    {DOWN}TYPING DERBY{BLK}{OFF}{BLU}"    :rem 160
412 POKE36879,232                         :rem 154
414 PRINT"{DOWN}{3 RIGHT}{WHT}{RVS}Q{13 RIGHT}Q":P
    RINT"{RVS}{2 RIGHT}Q3Q{11 RIGHT}Q8Q"   :rem 73
415 PRINT"{RVS}{2 RIGHT}2E4{11 RIGHT}6I9"  :rem 89
```

```
416 PRINT"{RIGHT}{RVS}Q{RED}D{WHT}5{11 RIGHT}7
    {RED}K{WHT}O{RVS}Q":PRINT"{RIGHT}{RVS}I{RED}S{WHT}C
    R{11 RIGHT}Y{RED}L{WHT}Ø                    :rem 18
418 PRINT"{RIGHT}{RVS}QX TQ{9 RIGHT}QU .P":PRINT"
    {RIGHT}{RVS}{RED}A{WHT}{2 SPACES}{RED}F{WHT}
    {9 RIGHT}{BLK}S{WHT}H{2 SPACES}{RED}:{WHT}"
                                               :rem 167
420 PRINT"{RIGHT}{RVS}Z{2 SPACES}G{WHT} {9 RIGHT}
    {BLK}P{RED}J{WHT}{2 SPACES}/":PRINT"{RIGHT}
    {RVS}{3 SPACES}V {9 RIGHT}{BLK}C{WHT}M
    {3 SPACES}"                                :rem 117
422 PRINT"{RIGHT}{RVS}{3 SPACES}B {9 RIGHT}{BLK}E
    {WHT}N{3 SPACES}":PRINT"{RIGHT}{RVS}{4 SPACES}
    {OFF}£{9 RIGHT}{*}{RVS}{4 SPACES}"    :rem 114
424 PRINT"{RIGHT}{RVS}{H}{3 SPACES}{11 RIGHT}
    {3 SPACES}{N}"                             :rem 55
426 PRINT"{RIGHT}{DOWN}{RVS}LEFT{10 RIGHT}RIGHT":P
    RINT"{RIGHT}FINGER RANGE":PRINT"{RIGHT}{RVS}
    {BLK}SPACE BAR"                            :rem 211
428 PRINT"{RIGHT}{RVS}{RED}HOME KEYS{BLU}{OFF}":PR
    INT"{DOWN}PRESS ANY KEY TO GO ON";         :rem 91
430 GETA$:IFA$=""THEN430                       :rem 81
432 RETURN                                    :rem 121
```

4

Brain Games

Brain Games

Are computers smart? The games in this section may make you think so.

Start with something simple like "Checkers." This version, written by Fred Hambrecht, allows you to play against (and sometimes defeat) your computer.

Then there's "Poker," by August Kwitowski, in which you sit down at the gaming table with your stony-faced monitor. Watch the screen carefully—was that the hint of a grin?

For an intellectual challenge, try Sean Puckett's "Quatrainment," where the computer innocently challenges you to match a simple geometric pattern. It's hard to beat—in more ways than one.

Or there's "Mind Boggle," by James E. Rylee. How many ways can you arrange five different colors? You only need to find *one* of those arrangements, but (you guessed it) the task may boggle your mind.

After all of that, you're probably going to be worn out—and a little "Therapy" may be just what you need. Programmer Steven Rubio developed this program to turn your VIC into a solid-state psychiatrist, and it's just the thing after a hard session with the games in this chapter. It's also a fascinating demonstration game (and the perfect answer to the question, "But what can your computer *do*?).

Checkers

Fred Hambrecht

In "Checkers," you match wits with an opponent who rarely makes mistakes: your computer. For the unexpanded VIC.

This computer version of "Checkers" plays just like the traditional game. The same rules apply; you can double- (or even triple-) jump, and you can win kings. Because it uses most of the memory on an unexpanded VIC, screen instructions are not included. However, if you have a VIC with expanded memory, there is plenty of room to add instructions at the beginning of the program if you wish.

The computer always makes the first move. When it's your turn, decide on the checker you want to move, then identify it first by column, then row. These are labeled next to the checkerboard. Be sure to enter the column number first, then the row number. Don't press RETURN. Before taking its turn, the program automatically moves your checker for you.

To jump one of the computer's checkers, you must press RETURN after entering the coordinates. In the case of a double-jump, enter the second set of coordinates after the prompt "+TO", then press RETURN. For a triple-jump, enter three sets of coordinates, etc.

You'll find that the computer plays a conservative game, but what it lacks in strategic imagination it makes up for by making few careless errors. Also, it does not require you to jump the opponent's checker, and it takes advantage of this tactic.

You'll have to play within the rules for checkers, since complete error checking was not possible in the unexpanded VIC. For instance, as the program is written, you'll find you can cheat the computer by jumping your own checker or by moving backwards. There are only about 100 bytes free in the VIC, which is not enough room to program the necessary checks for every possible illegal move.

Also, if you lose to the computer (you probably won't), there is no routine that sends you back to the start. Just enter RUN if you want to play another game.

Checkers

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
100 DIMX(4),S(7,7):G=-1:X(0)=-99:PRINT"{BLK}{CLR}"
                                                :rem 235
102 POKE36879,40:GOTO110                        :rem 104
110 DATA1,0,1,0,0,0,-1,0,0,1,0,0,0,-1,0,-1,15
                                                 :rem 95
120 A$="[19 SPACES}":B$="{HOME}{12 DOWN}"       :rem 121
130 FORX=0TO7:FORY=0TO7:READJ:IFJ=15THEN150
                                                :rem 246
140 S(X,Y)=J:GOTO160                            :rem 167
150 RESTORE:READS(X,Y)                          :rem 145
160 NEXTY,X:PRINT"{CLR}";                       :rem 140
170 FORX=0TO7:FORY=0TO7:IFS(X,Y)>-1THEN200:rem 134
180 IFS(X,Y)=-1THENFORA=-1TO1STEP2:B=G:GOSUB210:NE
    XTA                                         :rem 127
190 IFS(X,Y)=-2THENFORA=-1TO1STEP2:FORB=-1TO1STEP2
    :GOSUB210:NEXTB,A                            :rem 47
200 NEXTY,X:GOTO370                             :rem 187
210 U=X+A:V=Y+B:IFU<0ORU>7ORV<0ORV>7THEN260      :rem 7
220 IFS(U,V)=0THENGOSUB270:GOTO260               :rem 94
230 IFS(U,V)<0THEN260                           :rem 210
240 U=U+A:V=V+B:IFU<0ORV<0ORU>7ORV>7THEN260      :rem 4
250 IFS(U,V)=0THENGOSUB270                       :rem 86
260 RETURN                                      :rem 120
270 IFV=0ANDS(X,Y)=-1THENQ=Q+2                   :rem 69
280 IFABS(Y-V)=2THENQ=Q+5                         :rem 9
290 IFY=7THENQ=Q-2                              :rem 100
300 IFY=0ORU=7THENQ=Q+1                         :rem 188
310 FORC=-1TO1STEP2:IFU+C<0ORU+C>7ORV+G<0THEN350
                                                 :rem 8
320 IFS(U+C,V+G)<0THENQ=Q+1:GOTO350              :rem 96
330 IFU-C<0ORU-C>7ORV-G>7THEN350                :rem 216
340 IFS(U+C,V+G)>0AND(S(U-C,V-G)=0OR(U-C=XANDV-G=Y
    ))THENQ=Q-2                                 :rem 203
350 NEXTC:IFQ>X(0)THENX(0)=Q:X(1)=X:X(2)=Y:X(3)=U:
    X(4)=V                                      :rem 135
360 Q=0:RETURN                                  :rem 113
370 IFX(0)=-99THEN1040                          :rem 210
380 GOSUB1060:PRINT"ME"X(1);","X(2)"TO"X(3)","X(4)
    :X(0)=-99                                   :rem 222
390 FORXX=1TO400:NEXTXX                          :rem 1
400 IFX(4)=0THENS(X(3),X(4))=-2:GOTO420         :rem 202
410 S(X(3),X(4))=S(X(1),X(2))                   :rem 224
420 S(X(1),X(2))=0:IFABS(X(1)-X(3))<>2THEN510
                                                :rem 204
```

```
430 S((X(1)+X(3))/2,(X(2)+X(4))/2)=0          :rem 252
440 X=X(3):Y=X(4):IFS(X,Y)=-1THENB=-2:FORA=-2TO2ST
    EP4:GOSUB480                               :rem 65
450 IFS(X,Y)=-2THENFORA=-2TO2STEP4:FORB=-2TO2STEP4
    :GOSUB480:NEXTB                            :rem 210
460 NEXTA:IFX(0)<>-99THENPRINT"TO"X(3)","X(4);:X(0
    )=-99:GOTO400                              :rem 210
470 GOTO510                                    :rem 106
480 U=X+A:V=Y+B:IFU<0ORU>7ORV<0ORV>7THEN500:rem 13
490 IFS(U,V)=0ANDS(X+A/2,Y+B/2)>0THENGOSUB270
                                               :rem 185
500 RETURN                                     :rem 117
505 FORI=1TO25:PRINT:NEXT                      :rem 130
510 PRINT"{BLK}{HOME} ROW":PRINT"{BLK}{2 SPACES}
    {D}{8 I}{F}":FORY=7TO0STEP-1:PRINTY;"{LEFT}
    {RVS}{K}{OFF}";:FORX=0TO7                  :rem 235
520 IFS(X,Y)=0THENIF(X+Y)/2=INT((X+Y)/2)THENPRINT"
    {RVS} {OFF}";:GOTO580                       :rem 86
530 IFS(X,Y)=0THENPRINT" ";                     :rem 80
540 IFS(X,Y)=1THENPRINT"{RVS}Q{OFF}";:GOTO580
                                               :rem 215
550 IFS(X,Y)=-1THENPRINT"{RVS}W{OFF}";:GOTO580
                                                :rem 11
560 IFS(X,Y)=-2THENPRINT"*";:GOTO580           :rem 188
570 IFS(X,Y)=2THENPRINT"{RVS}*{OFF}";           :rem 36
580 NEXTX:PRINT"{K}":NEXTY:PRINT"{2 SPACES}{C}
    {RVS}{8 I}{OFF}{V}":PRINT"{3 SPACES}01234567 C
    OL"                                        :rem 112
590 GOSUB1060:PRINT"FROM";                      :rem 95
600 GETG$:IFG$=""THEN600                        :rem 91
610 IFG$<"0"ORG$>"7"THEN590                    :rem 211
620 E=VAL(G$):PRINTE;",";                      :rem 171
630 GETG$:IFG$=""THEN630                        :rem 97
640 IFG$<"0"ORG$>"7"THEN590                    :rem 214
650 H=VAL(G$):PRINTH                           :rem 206
660 X=E:Y=H:IFS(X,Y)<=0THEN590                  :rem 78
670 PRINT"TO";                                  :rem 76
680 GETG$:IFG$=""THEN680                       :rem 107
690 IFG$<"0"ORG$>"7"THEN670                    :rem 218
700 A=VAL(G$):PRINTA;",";                      :rem 162
710 GETG$:IFG$=""THEN710                        :rem 95
720 IFG$<"0"ORG$>"7"THEN670                    :rem 212
730 B=VAL(G$):PRINTB                           :rem 193
740 X=A:Y=B                                    :rem 131
750 IFS(X,Y)=0ANDABS(A-E)<=2ANDABS(A-E)=ABS(B-H)TH
    EN770                                       :rem 6
760 GOTO590                                    :rem 116
770 I=46                                       :rem 142
780 S(A,B)=S(E,H):S(E,H)=0:IFABS(E-A)<>2THEN910
                                               :rem 168
```

```
790 S((E+A)/2,(H+B)/2)=0                                    :rem 167
800 PRINT"+TO";                                             :rem 114
810 GETG$:IFG$=""THEN810                                     :rem 97
820 IFG$=CHR$(13)THEN910                                     :rem 80
830 IFG$<"0"ORG$>"7"THEN810                                 :rem 210
840 A1=VAL(G$):PRINTA1;",";                                   :rem 9
850 GETG$:IFG$=""THEN850                                    :rem 105
860 IFG$=CHR$(13)THEN910                                     :rem 84
870 IFG$<"0"ORG$>"7"THEN850                                 :rem 218
880 B1=VAL(G$):PRINTB1                                       :rem 41
890 IFS(A1,B1)<>0ORABS(A1-A)<>2ORABS(B1-B)<>2THEN8
    00                                                       :rem 0
900 E=A:H=B:A=A1:B=B1:I=I+15:GOTO780                        :rem 95
910 IFB=7THENS(A,B)=2                                       :rem 208
920 PRINT"{HOME}{11 DOWN}{3 RIGHT}01234567 COL"
                                                            :rem 11
930 PRINT"{2 UP}{2 SPACES}{C}{RVS}{8 I}{OFF}{V}
    {2 UP}"                                                 :rem 223
940 FORY=0TO7:PRINTY;"{LEFT}{RVS}{K}{OFF}";:FORX=0
    TO7                                                     :rem 160
950 IFS(X,Y)=0THENIF(X+Y)/2=INT((X+Y)/2)THENPRINT"
    {RVS} {OFF}";:GOTO1010                                  :rem 130
960 IFS(X,Y)=0THENPRINT" ";:GOTO1010                        :rem 140
970 IFS(X,Y)=1THENPRINT"{RVS}Q{OFF}";:GOTO1010
                                                             :rem 3
980 IFS(X,Y)=-1THENPRINT"{RVS}W{OFF}";:GOTO1010
                                                            :rem 55
990 IFS(X,Y)=-2THENPRINT"*";:GOTO1010                       :rem 232
1000 IFS(X,Y)=2THENPRINT"{RVS}*{OFF}";                       :rem 73
1010 NEXTX:PRINT"{K}{2 UP}":NEXTY                           :rem 249
1020 PRINT"{HOME} ROW":PRINT"{2 SPACES}{D}{8 I}{F}
     {2 UP}"                                                 :rem 67
1030 GOTO170                                                :rem 149
1040 GOSUB1060:FORI=1TO40:PRINT"Z";:FORJ=1TO50:NEX
     T:NEXT                                                 :rem 222
1050 PRINT"YOU WIN":END                                     :rem 147
1060 PRINTB$                                                :rem 186
1070 FORXX=1TO8:PRINTA$:NEXTXX:PRINTB$:RETURN
                                                            :rem 68
```

Poker

August J. Kwitowski

"Poker" is an original color and sound version of the classic card game of draw poker. The format and style of play are similar to those of commercial poker machines. For the VIC with at least 8K memory expansion.

"Poker" opens with a dynamic introduction featuring color, sound, and horizontal text scrolling. The number of each round is announced, and five cards are dealt at random. You build your hand by choosing which cards to keep or exchange (up to three cards can be drawn). The computer ranks your hand and announces the payoff, if any. Your cumulative winnings (or losses) are displayed at the top of the screen. The higher the hand, the more you win. For example, you break even on a pair of jacks or better, but a royal flush brings you $250.

Program Features

The program contains several interesting features:

- The short routine in lines 230 and 240 scrolls single lines of text horizontally across the screen.
- To conserve memory, lines of text used in the introduction are reused in the routine that announces the rank and value of the hand.
- A machine language (ML) routine POKEd into the cassette buffer is used to create a colorful border. The routine is accessed by the SYS 828 statement in line 350.
- A hand's rank and value are determined by using ML and IF-THEN statements in lines 2110–2210. The machine language performs a bubble sort (ranking) of the card values and determines which cards are duplicates (two kings, three jacks, etc.). The ML routines are POKEd into the cassette buffer and are accessed in lines 2020 and 2130.

How It Works

Line(s)	Description
20	POKE machine language in buffer.
30–210	Define constants and DIMension variables.
220–300	Scroll lines of text with sound.
310	Set text lines to null strings if they're not used again.

350–370	Hand number routines; create card screen.
500–630	Select cards; determine display characters and colors.
640–800	Deal cards.
810–1550	Keep or change each of the five cards.
2000–2170	Determine rank of hand.
2180–2220	Determine value and correct line of text.
3030–3100	Display determination with sound.
4030–4050	Subroutine for hand number.
5000	Subroutine to flash border, colors.
5050	Subroutine to display winnings.

Poker

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
20 POKE36879,200:PRINT"{CLR}":FORA=828TO998:READB:
   POKEA,B:NEXT                                  :rem 181
30 CL=33792:WK=4096                               :rem 35
60 DIMJ%(13,4):DIMG$(20):S1=36875:S2=S1+1:VL=S1+3:
   D1=0:SC=0:HD=0                                 :rem 97
70 G$(4)="{2 SPACES}*** POKER{2 SPACES}256 ***
   {2 SPACES}"                                    :rem 242
80 G$(7)=" IT'S YOU AGAINST VIC"                  :rem 217
100 G$(9)="{2 SPACES}YOU WIN AS FOLLOWS: ":G$(10)=
    "{3 SPACES}ROYAL FLUSH-$250{2 SPACES}":rem 175
120 G$(11)=" STRAIGHT FLUSH-$100{2 SPACES}":G$(12)
    ="{3 SPACES}4 OF A KIND-$20{4 SPACES}":rem 185
140 G$(13)="{4 SPACES}FULL HOUSE-$10{4 SPACES}":G$
    (14)="{7 SPACES}FLUSH-$8{7 SPACES}"           :rem 134
160 G$(15)="{5 SPACES}STRAIGHT-$5{5 SPACES}":G$(16
    )="{4 SPACES}3 OF A KIND-$4{4 SPACES}":rem 184
180 G$(17)="{6 SPACES}2 PAIR-$3{7 SPACES}":G$(18)=
    "{2 SPACES}PAIR, JACKS & UP-$1 "              :rem 17
200 G$(20)="{2 SPACES}EACH HAND COSTS $1.":N$="
    {HOME}{22 DOWN}"                              :rem 34
210 B$=LEFT$(N$,20):JW$=LEFT$(N$,10)              :rem 177
220 A=4:MM=220:G=50:PRINT"{BLK}":POKEVL,15:D1=0
                                                  :rem 98
230 FORB=1TO22:PRINTLEFT$(N$,A)RIGHT$(G$(A),B):POK
    ES1,MM                                        :rem 138
240 FORC=1TOG:NEXT:POKES1,0:NEXT:FORB=1TOD1:NEXT:I
    FA=20THENPOKEVL,0:GOTO310                     :rem 193
250 IFA=18THENA=20:MM=220:PRINT"{WHT}":G=50:D1=150
    0                                             :rem 166
260 IFA>8ANDA<18THENA=A+1                         :rem 225
270 IFA=7THENA=9:PRINT"{BLU}":G=40:D1=600         :rem 145
```

```
280 IFA=5THENGOSUB5000:FORA=1TO600:NEXT:A=7:PRINT"
    {WHT}":MM=238:G=40:D1=600                    :rem 35
290 IFA=4THENA=5:PRINT"{RED}":MM=226            :rem 144
300 GOTO230                                      :rem 97
310 G$(4)="":G$(5)="":G$(7)="":G$(10)="":G$(20)=""
                                                 :rem 16
320 AK$="{DOWN}{GRN}{RVS} {DOWN}{LEFT} {DOWN}
    {LEFT} ":D$=B$+"{21 SPACES}"                 :rem 23
340 E$=LEFT$(N$,15):F$=E$+"{21 SPACES}":X=RND(-TI)
                                                :rem 104
350 HD=HD+1:GOSUB4030:POKE36879,31:PRINTCHR$(147):
    SYS828                                      :rem 100
360 PRINTLEFT$(N$,5)SPC(4)"{BLU}HIT {RVS}K{OFF} TO
     KEEP"                                       :rem 3
370 PRINTLEFT$(N$,7)SPC(3)"HIT {RVS}C{OFF} TO CHAN
    GE":GOSUB5050                               :rem 227
500 X=INT(RND(1)*13)+1:Y=INT(RND(1)*4)+1:IFJ%(X,Y)
    =1THEN500                                   :rem 122
510 J%(X,Y)=1:K=K+1                              :rem 10
520 E=32:IFY=1THENG=88:H=0                       :rem 32
530 IFY=2THENG=83:H=2                           :rem 254
540 IFY=3THENG=65:H=0                           :rem 254
550 IFY=4THENG=90:H=2                            :rem 0
560 IFX=10THENE=49:F=48:GOTO620                 :rem 114
570 IFX>1ANDX<10THENF=X+48                       :rem 91
580 IFX=11THENF=10                               :rem 54
590 IFX=12THENF=17                               :rem 63
600 IFX=13THENF=11                               :rem 50
610 IFX=1THENX=14:F=1                            :rem 3
620 IFK>5THENRETURN                             :rem 244
630 IFX=1THENX=14:F=1                            :rem 5
640 IFK=1THENCD=WK+199:PT(1)=X:ST(1)=G           :rem 33
650 IFK=2THENCD=WK+203:PT(2)=X:ST(2)=G           :rem 23
660 IFK=3THENCD=WK+207:PT(3)=X:ST(3)=G           :rem 31
670 IFK=4THENCD=WK+211:PT(4)=X:ST(4)=G           :rem 30
680 IFK=5THENCD=WK+215:PT(5)=X:ST(5)=G:GOSUB700:PO
    KEVL,5:Z=250:GOTO810                         :rem 87
690 GOSUB700:GOTO500                            :rem 190
700 POKECD,112:POKECD+CL,0:POKECD+1,64:POKECD+1+CL
    ,0:POKECD+2,110:POKECD+2+CL,0               :rem 197
710 FORA=(CD+24)TO(CD+68)STEP22:POKEA,93:POKEA+CL,
    0:NEXT                                      :rem 149
720 FORA=(CD+22)TO(CD+66)STEP22:POKEA,93:POKEA+CL,
    0:NEXT                                      :rem 146
730 POKECD+88,109:POKECD+88+CL,0:POKECD+89,64:POKE
    CD+89+CL,0:POKECD+90,125                     :rem 94
740 POKECD+90+CL,0:LF=1:WB=230                  :rem 218
750 E1=E:F1=F:G1=G:H1=H:E=160:F=160:G=160:H=0
                                                :rem 33
```

```
760 LF=LF+1:POKES1,WB:POKEVL,14                    :rem 168
770 POKECD+23,E:POKECD+23+CL,H:POKECD+45,F:POKECD+
    45+CL,H:POKECD+67,G:POKECD+67+CL,H              :rem 92
780 FORB=1TO100:NEXT:POKEVL,0:POKES1,0:IFLF=0THENR
    ETURN                                          :rem 106
790 IFLF=4THENLF=0:E=E1:G=G1:H=H1:F=F1:GOTO770
                                                   :rem 69
800 H=H+3:WB=WB+5:GOTO760                          :rem 220
810 POKE198,0:PRINTD$:PRINTB$;:PRINTTAB(3)CHR$(28)
    "KEEP OR CHANGE?":CT=0                         :rem 104
820 PRINTE$SPC(2)CHR$(30)"?":POKES1,Z              :rem 164
830 FORA=1TO100:NEXT:PRINTE$SPC(2)" ":POKES1,0:FOR
    A=1TO50:NEXT                                    :rem 85
840 GETH$:IFH$=""THEN820                           :rem 103
850 IFH$="C"ORH$="K"THEN870                          :rem 3
860 GOTO820                                        :rem 113
870 IFH$="K"THEN900                                 :rem 46
880 IFH$="C"THENCT=CT+1:GOSUB500:PT(1)=X:ST(1)=G:E
    (1)=E:F(1)=F:G(1)=G:H(1)=H                     :rem 152
890 PRINTJW$SPC(2)AK$                               :rem 12
900 PRINTE$SPC(6)"?":FORA=1TO100:NEXT:PRINTE$SPC(6
    )" ":FORA=1TO50:NEXT                            :rem 46
910 GETI$:IFI$=""THEN900                           :rem 102
920 IFI$="C"ORI$="K"THEN940                          :rem 1
930 GOTO900                                        :rem 110
940 IFI$="K"THEN970                                 :rem 52
950 CT=CT+1:GOSUB500:PT(2)=X:ST(2)=G:E(2)=E:F(2)=F
    :G(2)=G:H(2)=H                                 :rem 174
960 PRINTJW$SPC(6)AK$                               :rem 14
970 PRINTE$SPC(10)"?":FORA=1TO100:NEXT:PRINTE$SPC(
    10)" ":FORA=1TO50:NEXT                         :rem 139
980 GETJ$:IFJ$=""THEN970                           :rem 118
990 IFJ$="C"ORJ$="K"THEN1020                         :rem 48
1000 GOTO970                                       :rem 154
1020 IFJ$="K"THEN1050                              :rem 129
1030 CT=CT+1:GOSUB500:PT(3)=X:ST(3)=G:E(3)=E:F(3)=
     F:G(3)=G:H(3)=H                               :rem 218
1040 PRINTJW$SPC(10)AK$:FORA=1TO1000:NEXT:IFCT=3TH
     EN1500                                        :rem 209
1050 PRINTE$SPC(14)"?":FORA=1TO100:NEXT:PRINTE$SPC
     (14)" ":FORA=1TO50:NEXT                       :rem 185
1060 GETK$:IFK$=""THEN1050                         :rem 196
1070 IFK$="K"ORK$="C"THEN1090                        :rem 95
1080 GOTO1050                                      :rem 200
1090 IFK$="K"THEN1120                              :rem 135
1100 CT=CT+1:GOSUB500:PT(4)=X:ST(4)=G:E(4)=E:F(4)=
     F:G(4)=G:H(4)=H                               :rem 222
1110 PRINTJW$SPC(14)AK$:IFCT=3THEN1500               :rem 38
```

```
1120 PRINTE$SPC(18)"?":FORA=1TO100:NEXT:PRINTE$SPC
     (18)" ":FORA=1TO50:NEXT                    :rem 191
1130 GETL$:IFL$=""THEN1120                      :rem 194
1140 IFL$="C"ORL$="K"THEN1160                    :rem 93
1150 GOTO1120                                   :rem 196
1160 IFL$="K"THEN1500                           :rem 136
1170 CT=CT+1:GOSUB500:CD=WK+215:PT(5)=X:ST(5)=G:E(
     5)=E:F(5)=F:G(5)=G:H(5)=H                   :rem 78
1180 PRINTJW$SPC(18)AK$                         :rem 108
1500 FORTV=1TO5:IFTV>5THEN1560                  :rem 126
1510 IFE(TV)>0THEN1530                          :rem 252
1520 NEXTTV:IFTV=5THEN1560                      :rem 145
1530 E=E(TV):F=F(TV):G=G(TV):H=H(TV)            :rem 139
1540 CD=WK+195+TV*4:IFCD>WK+215THEN1560          :rem 27
1550 GOSUB700:IFTV<5THENNEXTTV                  :rem 222
1560 FORA=1TO5:E(A)=0:F(A)=0:G(A)=0:H(A)=0:NEXTA
                                                :rem 242
2000 PRINTD$F$:FORA=1TO5:POKE(1015+A),PT(A):NEXT
                                                :rem 145
2010 FORA=1TO5:POKE(1015+A),PT(A):NEXT          :rem 249
2020 SYS908:FORA=1TO5:PT(A)=PEEK((1015+A)):NEXT
                                                :rem 44
2110 YY=0:IFPT(4)-PT(3)=1THENIFPT(3)-PT(2)=1THENIF
     PT(2)-PT(1)=1THENYY=1                      :rem 53
2115 IFYY=1THENIF(PT(5)-PT(4)=1)OR(PT(1)+PT(5)-15=
     1)THENSS=1                                 :rem 38
2120 IFST(1)=ST(2)THENIFST(2)=ST(3)THENIFST(3)=ST(
     4)THENIFST(4)=ST(5)THENFL=1                 :rem 9
2130 SYS960:XE=PEEK(1011):ZQ=PEEK(1012)         :rem 13
2160 IFPT(1)=PT(2)THENIFPT(1)=PT(3)THENIFPT(1)=PT(
     4)THENFR=1                                 :rem 170
2170 IFPT(5)=PT(4)THENIFPT(5)=PT(3)THENIFPT(5)=PT(
     2)THENFR=1                                 :rem 183
2180 IFSS=1THENIFFL=1THENIFPT(5)=14THENSC=SC+249:Z
     $=G$(10):GOTO3030                          :rem 99
2190 IFSS=1THENIFFL=1THENSC=SC+99:Z$=G$(4):GOTO303
     0                                          :rem 128
2200 IFFR=1THENSC=SC+19:Z$=G$(12):GOTO3030:rem 211
2210 IFZQ=4THENIFFR<>1THENSC=SC+9:Z$=G$(13):GOTO30
     30                                         :rem 187
2220 IFFL=1THENSC=SC+7:Z$=G$(14):GOTO3030 :rem 158
2230 IFSS=1THENSC=SC+4:Z$=G$(15):GOTO3030 :rem 177
2240 IFZQ=3THENSC=SC+3:Z$=G$(16):GOTO3030 :rem 185
2250 IFZQ=2THENSC=SC+2:Z$=G$(17):GOTO3030 :rem 185
2260 IFZQ=1ANDXE>=11THENZ$=G$(18):GOTO3030  :rem 7
2270 SC=SC-1:Z$="{6 SPACES}LOUSY HAND!{5 SPACES}":
     QP=1                                       :rem 236
3030 GOSUB5050:PRINTCHR$(156):IFQP=1THENPRINTCHR$(
     144)                                       :rem 68
```

```
3040 FORA=1TO5:PRINTB$;Z$:UA=20:FORB=135TO243STEP1
     2                                        :rem 158
3050 IFQP=1THENFORB=243TO135STEP-12:UA=32  :rem 14
3060 POKEVL,15:POKES1,B:POKES2,B:FORC=1TOUA:NEXT:N
     EXT                                      :rem 224
3070 POKEVL,0:POKES1,0:POKES2,0:PRINTD$:FORD=1TO10
     0:NEXT:NEXT                              :rem 178
3080 FORX=1TO13:FORY=1TO4:J%(X,Y)=0:NEXT:NEXT:K=0
                                              :rem 102
3090 FORA=1TO5:PT(A)=0:ST(A)=0:NEXT:SS=0:FL=0:ZQ=0
     :FR=0:K=0:XE=0:QP=0                       :rem 5
3100 FORA=1TO1500:NEXT:GOTO350                :rem 71
4030 POKE36879,120:PRINT"{CLR}"LEFT$(N$,11)SPC(8)"
     {BLK}{RVS}HAND";HD                       :rem 59
4040 D=231:POKEVL,15:FORA=1TO3:FORB=120TO127:POKE3
     6879,B:POKES1,D                          :rem 215
4050 POKES2,D:FORC=1TO40:NEXT:D=D+1:NEXT:NEXT:POKE
     S1,0:POKES2,0:RETURN                     :rem 57
5000 FORA=1TO3:FORB=200TO207:POKE36879,B:FORC=1TO5
     0:NEXT:NEXT:NEXT:POKE36879,200:RETURN :rem 30
5050 PRINTLEFT$(N$,3)SPC(4)CHR$(28)CHR$(18)"WINNIN
     GS:"CHR$(146)"$";SC;"{2 SPACES}":RETURN
                                              :rem 84
6000 DATA160,5,162,22,169,160,157,255,15,157,227,1
     7,136,208,3,32,131,3,152,157,255,147 :rem 188
6010 DATA157,227,149,202,208,232,160,5,162,220,169
     ,160,157,22,16,157,43,16,157,8,17,157:rem 245
6020 DATA29,17,136,208,3,32,131,3,152,157,22,148,1
     57,43,148,157,8,149,157,29,149,32,134:rem 254
6030 DATA3,208,218,96,160,7,96,138,56,233,22,170,9
     6,162                                    :rem 214
6040 DATA4,142,246,3,174,246,3,160,0,140,247,3,185
     ,249,3,217,248,3,176,16,72,185,248    :rem 98
6050 DATA3,153,249,3,104,153,248,3,169,1,141,247,3
     ,200,202,208,228,173,247,3,240,5,206 :rem 171
6060 DATA246,3,208,210,96,162                 :rem 198
6070 DATA 0,142,245,3,172,245,3,185,248,3,217,249,
     3,208,4,232,141,243,3,200,192,4,208   :rem 74
6080 DATA242,238,245,3,173,245,3,201,4,208,226,142
     ,244,3,96                                :rem 138
6090 DATA836,29,839,31,849,149,852,151,864,30,867,
     30                                       :rem 103
```

Quatrainment

Sean Puckett

Fast thinking and logic are required for "Quatrainment," a game in which you race the clock and plan your moves to match a master pattern. You'll need at least 3K expansion, as well as a joystick.

The object of "Quatrainment" is to match a pattern generated by the computer, using the fewest moves possible and finishing in the shortest amount of time. As the game begins, your game board is drawn at the left of the screen, and the master pattern is displayed at the right. A timer and move counter are also displayed.

A cursor appears in one of the squares on the game board. To change your pattern, use the joystick to move the cursor onto the square you want. Then press the joystick button. Part of your pattern will toggle from on to off, or from off to on, depending on whether you are in the middle, in a corner, or at an edge of the board. The different ways the pattern can change are shown in examples displayed on the screen.

When you match the pattern, your weighted score will be displayed, based on elapsed time and the number of moves you made. The lower your score, the better.

Quatrainment

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
8 SR=4*(PEEK(36866)AND128)+64*(PEEK(36869)AND12Ø):
  CO=(37888+4*(PEEK(36866)AND128))-SR      :rem 191
1Ø PRINT"{CLR}":POKE214,1Ø:PRINT:POKE211,5:A$="QUA
   TRAINMENT":POKE646,Ø                    :rem 149
15 DF=37154:PA=37151:PB=37152               :rem 69
2Ø FORT1=1TOLEN(A$):PRINTMID$(A$,T1,1);:FORT=1TO2Ø
   Ø:NEXT:NEXT:FORT=1TO5ØØ:NEXT            :rem 214
3Ø RN=16:REMFOR RANDOM INITIAL GRID CHANGE LINE 3Ø
    TO RN=RND(Ø)*15+1                      :rem 159
4Ø PRINT"{CLR}";TAB(1Ø);"TIME:"             :rem 116
45 PRINT"{2 DOWN}{12 LEFT}MOVES:";MO         :rem 83
5Ø PRINT"{2 DOWN}{12 LEFT}EDGES {D}{B}"     :rem 135
53 PRINT"{2 DOWN}{12 LEFT}CORNERS{2 SPACES}{RVS}
   {OFF}{V}{DOWN}{2 LEFT}{V}"               :rem 62
```

```
54 PRINT"{DOWN}{12 LEFT}CENTERS {UP}{D}{DOWN}
   {LEFT}{RVS}{F}{OFF}{V}"                          :rem 195
55 PRINT: POKE214,19:PRINT:POKE211,0:REM{2 SPACES}
   PRINT"HIT {RVS}FIRE{OFF} IF YOU MATCH"           :rem 68
100 GOTO140                                         :rem 95
110 FORL=1TO4:FORB=1TO4:D(L,B)=PEEK(C(L,B)):NEXTB:
    NEXTL:RETURN                                    :rem 214
115 FORT=1TO500:NEXT                                :rem 241
120 FORL=1TO4:FORB=1TO4:IFB(L,B)<>D(L,B)THENRETURN
                                                    :rem 161
130 NEXT:NEXT:SC=VAL(TI$)/16+MO/5:PRINT"{HOME}
    {10 DOWN}":PRINT"{RIGHT}MATCHED!!{2 DOWN}"
                                                    :rem 89
135 PRINT" SCORE:":PRINT INT(SC);"{DOWN}"           :rem 73
136 PRINTTAB(5)"{3 DOWN}AGAIN Y OR N?"              :rem 226
137 IFPEEK(197)<>11ANDPEEK(197)<>28THEN137          :rem 181
138 IF PEEK(197)=11THENRUN                           :rem 5
139 END                                             :rem 116
140 POKE36879,24                                    :rem 103
150 FORJ=0TO8STEP2                                  :rem 125
160 FORT=SRTOSR+176STEP22:POKET+J,93                 :rem 47
170 POKET+CO+J,6:NEXT:NEXT                          :rem 161
180 FORJ=0TO8STEP2                                  :rem 128
190 FORT=SRTOSR+8:POKET+J*22,67:POKET+CO+J*22,6
                                                    :rem 154
200 NEXT:NEXT                                        :rem 74
210 FORJ=0TO8STEP2                                  :rem 122
215 FORT=SR+230TOSR+230+176STEP22:POKET+J,93
                                                    :rem 176
220 POKET+CO+J,6:NEXT:NEXT                          :rem 157
225 FORJ=0TO8STEP2                                  :rem 128
230 FORT=SR+230TOSR+8+230:POKET+J*22,67:POKET+CO+J
    *22,6                                            :rem 21
235 NEXT:NEXT                                        :rem 82
280 FORU=1TO4:FORT=1TO4:C(T,U)=SR+207+2*T+44*U:NEX
    TT:NEXTU                                        :rem 166
300 FORT=1TO4:A(T,1)=SR+21+2*T:A(T,2)=SR+65+2*T
                                                    :rem 172
305 A(T,3)=SR+109+2*T:A(T,4)=SR+153+2*T:NEXT
                                                    :rem 217
310 GOSUB570:X=1:Y=1:GOSUB500                         :rem 0
315 TI$="000000"                                    :rem 251
320 POKEDF,127:GP=PEEK(PB)AND128:JO=-(GP=0)*4:POKE
    DF,255:GP=PEEK(PA):IFJOTHEN340                  :rem 128
321 JO=-((GPAND8)=0)*2:IFJOTHEN340                  :rem 206
322 JO=-((GPAND4)=0):IFJOTHEN340                    :rem 111
324 JO=-((GPAND16)=0)*3                             :rem 217
325 IFJO=0THENJO=5                                  :rem 105
340 ONJOGOSUB390,410,450,430,470                    :rem 249
```

```
350 IF-((GPAND32)=0)=0THEN375                    :rem 109
360 GOSUB910:MO=MO+1                             :rem 190
370 GP=PEEK(PA):IF-((GPAND32)=0)=1 THEN370:GOSUB11
    0:GOSUB120                                   :rem 25
375 PRINT"{HOME}";TAB(15);RIGHT$(TI$,5);"{2 DOWN}
    {4 LEFT}";MO                                 :rem 136
380 GOTO320                                      :rem 105
390 IFY-1<=0THEN480                               :rem 86
400 Y=Y-1:GOSUB 500:RETURN                        :rem 74
410 IFY+1=5THEN480                                :rem 22
420 Y=Y+1:GOSUB500:RETURN                         :rem 74
430 IFX+1=5THEN480                                :rem 23
440 X=X+1:GOSUB500:RETURN                         :rem 74
450 IFX-1<=0THEN480                               :rem 82
460 X=X-1:GOSUB500:RETURN                         :rem 78
470 GOSUB500:RETURN                              :rem 202
480 RETURN                                       :rem 124
490 GOTO320                                      :rem 107
500 P1=PEEK(A(X,Y))                               :rem 56
510 POKEA(X,Y),81                                :rem 201
520 POKEA(X,Y)+CO,2:FORT=1TO50:NEXT              :rem 180
530 POKEA(X,Y),P1                                :rem 227
535 P1=0                                         :rem 139
540 GOSUB110:GOSUB120:RETURN                      :rem 18
570 WE=INT(RND(0)*8)+1:FORJ=1TOWE*RN:READ Q:NEXT
                                                 :rem 214
580 FORY=1TO4:FORX=1TO4:READQ:IFQ=0THEN600:rem 211
590 GOSUB610                                     :rem 181
600 NEXTX:NEXTY:GOSUB640:GOSUB680:RETURN         :rem 197
610 POKEA(X,Y),86                                :rem 207
620 POKEA(X,Y)+CO,2                               :rem 81
630 GOSUB110,120:RETURN                          :rem 132
640 FORX=1TO4:FORY=1TO4:B(X,Y)=PEEK(A(X,Y))
                                                 :rem 169
670 NEXTY:NEXTX:RETURN                            :rem 32
680 FORY=1TO4:FORX=1TO4:READP                    :rem 135
690 IFPTHENPOKEC(X,Y),86:POKEC(X,Y)+CO,0         :rem 218
700 NEXTX:NEXTY:RETURN                            :rem 26
710 DATA1,1,1,1, 1,0,0,1, 1,0,0,1, 1,1,1,1       :rem 82
720 DATA0,0,0,0, 0,1,1,0, 0,1,1,0, 0,0,0,0       :rem 75
730 DATA0,1,1,0, 1,0,0,1, 1,0,0,1, 0,1,1,0       :rem 80
740 DATA1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1       :rem 89
750 DATA1,0,0,1, 0,1,1,0, 0,1,1,0, 1,0,0,1       :rem 82
760 DATA1,1,1,1, 0,0,0,0, 0,0,0,0, 1,1,1,1       :rem 83
770 DATA0,0,0,1, 0,0,0,1, 0,0,0,1, 0,0,0,1       :rem 80
775 DATA1,0,0,1, 0,0,0,0, 0,0,0,0, 1,0,0,1       :rem 85
780 DATA0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0       :rem 77
790 DATA0,0,0,0, 1,0,0,1, 1,0,0,1, 0,0,0,0       :rem 82
800 REM REVERSE                                  :rem 152
```

```
810 POKEA(C,D),118-PEEK(A(C,D))                    :rem 187
830 POKEA(C,D)+CO,2                                 :rem 42
840 P1=0                                           :rem 138
860 RETURN                                         :rem 126
870 REM SET DATA POINTER                           :rem 170
910 REM{2 SPACES}WHICH ONES{2 SPACES}TO CHANGE
                                                   :rem 111
920 IFX+Y<>2THEN950                                :rem 127
930 FORC=2TO3:D=1:GOSUB810:NEXT:FORD=1TO3:C=1:GOSU
    B810:NEXT                                       :rem 35
940 D=2:C=2:GOSUB810:RETURN                        :rem 169
950 IF X+Y<>8THEN980                               :rem 139
960 FORC=3TO2STEP-1:D=4:GOSUB810:NEXT:FOR D=4TO2ST
    EP-1:C=4:GOSUB810:NEXT                          :rem 98
970 C=3:D=3:GOSUB810:RETURN                        :rem 174
980 IF X+Y<>5THEN1020                              :rem 173
990 IF X<>4THEN 1020                                :rem 41
1000 FORC=3TO2STEP-1:D=1:GOSUB810:NEXT:FORD=1TO3:C
     =4:GOSUB810:NEXT                              :rem 229
1010 C=3:D=2:GOSUB810:RETURN                       :rem 207
1020 IFX+Y<>5THEN1060                              :rem 211
1030 IFX<>1THEN1060                                 :rem 76
1040 FORC=2TO3:D=4:GOSUB810:NEXT:FORD=4TO2STEP-1:C
     =1:GOSUB810:NEXT                              :rem 235
1050 C=2:D=3:GOSUB810:RETURN                       :rem 211
1060 REM CHECK EDGES                               :rem 113
1070 IF(X>1ANDX<4)AND(Y=1ORY=4)THENC=X-1:D=Y:GOSUB
     810:C=X+1:GOSUB810:GOSUB1100                  :rem 105
1080 IF(Y>1ANDY<4)AND(X=1ORX=4)THEND=Y-1:C=X:GOSUB
     810:D=Y+1:GOSUB810:GOSUB1100                  :rem 108
1090 GOTO1160                                      :rem 203
1100 IFY=1THEND=Y+1:C=X:GOSUB810                   :rem 226
1110 IFY=4THEND=Y-1:C=X:GOSUB810                   :rem 232
1120 IFX=4THENC=X-1:D=Y:GOSUB810                   :rem 232
1130 IFX=1THENC=X+1:D=Y:GOSUB810                   :rem 228
1140 RETURN                                        :rem 166
1150 REM CHECK CENTERS                              :rem 29
1160 IF (X=1)OR(Y=1)OR(X=4)OR(Y=4)THEN 1200
                                                   :rem 144
1170 D=Y+1:C=X:GOSUB810:C=X-1:D=Y:GOSUB810         :rem 59
1180 D=Y-1:C=X:GOSUB810:C=X+1:D=Y:GOSUB810         :rem 60
1190 C=X:D=Y:GOSUB810                               :rem 10
1200 RETURN                                        :rem 163
```

Mind Boggle

James E. Rylee

"Mind Boggle" is a game of logic based on the popular game "Master Mind." You can play alone or against others, trying to solve the puzzle in the fewest moves. For the unexpanded VIC.

"Mind Boggle" begins by selecting four colors out of six possible choices and arranging them in a random sequence. Then it's up to you to find the correct colors and arrange them in the correct order, using clues given by the program. Each color has a musical sound associated with it. Your selection is displayed on the left side by number, and your clues are on the right.

Guess the Colors and Sequence

When the computer asks SELECT COLORS, enter your guesses for the colors and their sequence by entering the appropriate numeric values and pressing RETURN. Any entry other than 1–6, or any more than the four required digits, will result in an ILLEGAL INPUT message and ask you to again SELECT COLORS. The computer then analyzes your guess and gives you the results.

A black dot indicates that you have guessed a correct color in the correct position. A white dot indicates that you have guessed a correct color only. The position of a clue does not correspond directly to any one color or correct position. You must move the colors around and analyze the clues to determine which are the correct colors and positions.

For example, if you guess 1234 and the computer responds with two white dots, you know two of the colors are correct but in the wrong place. If your next guess, 3214, gains two black dots, you can deduce that colors 3 and 1 were correct and that the hidden code is 3x1x (where x is an unknown).

If you win (that is, if you guess the colors in ten turns) you are rated by the computer. If not, the colors are displayed for you. In either case, you can choose to play again.

Making It Harder

If Mind Boggle doesn't provide sufficient challenge, a few simple changes will produce a more difficult version. As the game is written, each of the four positions will contain a different one of the six possible colors. If you allow the same color to appear in more than one position, the number of possible sequences soars. Game play remains the same, except that a color may now appear two, three, or even four times. To accomplish this, change these lines:

```
1 PRINT"{CLR}{5 RIGHT}{9 DOWN}MIND BOGGLE":CLR

9 A$="123456":GOSUB13:A1$=R$:A1=VAL(A1$)
10 GOSUB13:A2$=R$:A2=VAL(A2$)
11 GOSUB13:A3$=R$:A3=VAL(A3$)
12 GOSUB13:A4$=R$:A4=VAL(A4$)
13 R=INT(RND(1)*6)+1:R$=MID$(A$,R,1):RETURN
```

If you're having trouble telling the colors apart (perhaps you're using a black-and-white TV set), the following changes will cause the numeric value for the color to be displayed:

```
51 PRINT"{BLK}{RVS}{2 SPACES}1{OFF} ";:POKES2,135:
   GOTO57
52 PRINT"{WHT}{RVS}{2 SPACES}2{OFF} ";:POKES2,159:
   GOTO57
53 PRINT"{RED}{RVS}{2 SPACES}3{OFF} ";:POKES2,175:
   GOTO57
54 PRINT"{BLU}{RVS}{2 SPACES}4{OFF} ";:POKES2,191:
   GOTO57
55 PRINT"{PUR}{RVS}{2 SPACES}5{OFF} ";:POKES2,201:
   GOTO57
56 PRINT"{GRN}{RVS}{2 SPACES}6{OFF} ";:POKES2,209
```

You might have problems if you use any of the cursor keys or cause the screen to scroll, since the playing screen could be changed. The game will continue, but you won't be able to see the entries which have scrolled off the screen.

Mind Boggle

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 PRINT"{CLR}{5 RIGHT}{10 DOWN}MIND BOGGLE":CLR
                                              :rem 195
2 FOR T= 1 TO 2000:NEXT                       :rem 185
3 DIMC(4),G(4)                                :rem 205
```

```
4 S1=36875:S2=S1+1:POKES1+3,15:POKES1+4,110:X=0
                                              :rem 196
5 PRINT"{CLR}{19 DOWN}"                       :rem 220
6 PRINT"{2 SPACES}{BLK}{RVS} 1{OFF} {WHT}{RVS} 2
  {OFF} {RED}{RVS} 3{OFF} {CYN}{RVS} 4{OFF} {PUR}
  {RVS} 5{OFF} {GRN}{RVS} 6{OFF}{BLK}"        :rem 174
7 PRINT"{BLK}I CHOOSE 4 COLORS NOW":FORL=1TO100:PO
  KES2,INT(RND(1)*128)+128:FORM=1TO10         :rem 210
8 NEXTM:NEXTL:POKES2,0:GOSUB106               :rem 36
9 A$="123456":R=INT(RND(1)*6)+1:A1$=MID$(A$,R,1):A
  1=VAL(A1$)                                  :rem 202
10 R=INT(RND(1)*6)+1:A2$=MID$(A$,R,1):IFA1$=A2$THE
   N10                                        :rem 174
11 A2=VAL(A2$)                                :rem 221
12 R=INT(RND(1)*6)+1:A3$=MID$(A$,R,1):IFA1$=A3$ORA
   2$=A3$THEN12                               :rem 193
13 A3=VAL(A3$)                                :rem 225
14 R=INT(RND(1)*6)+1:A4$=MID$(A$,R,1):IFA1$=A4$ORA
   2$=A4$ORA3$=A4$THEN14                      :rem 215
15 A4=VAL(A4$)                                :rem 229
16 X$="":POKES1,135:FORL=1TO100:NEXTL:POKES1,0:INP
   UT"SELECT COLORS";X$                       :rem 138
17 IFLEN(X$)<>4THENGOSUB106:GOTO93            :rem 118
18 FORE=1TO4:V=VAL(MID$(X$,E,1))              :rem 116
19 IFV<1ORV>6THENGOSUB106:GOTO93               :rem 76
20 NEXTE                                      :rem 230
21 X=X+1:B=0:W=0:AA$=A1$+A2$+A3$+A4$          :rem 143
22 FORJ=1TO4                                  :rem 218
23 G(J)=VAL(MID$(X$,J,1))                      :rem 86
24 C(J)=VAL(MID$(AA$,J,1))                    :rem 125
25 IFG(J)=C(J)THENB=B+1:G(J)=0:C(J)=0          :rem 77
26 NEXTJ                                      :rem 241
27 FORJ=1TO4:IFC(J)=0THEN33                   :rem 136
28 H=0:FORK=1TO4                              :rem 208
29 IFC(J)=0THEN32                             :rem 217
30 IFC(J)<>G(K)THEN32                         :rem 193
31 H=1:G(K)=0:C(J)=0                           :rem 41
32 NEXTK:W=W+H                                :rem 135
33 NEXTJ                                      :rem 239
34 ONXGOTO35,36,37,38,39,40,41,42,43,44        :rem 49
35 PRINT"{HOME} 1 ";:GOTO45                   :rem 148
36 PRINT"{HOME}{2 DOWN} 2 ";:GOTO45           :rem 184
37 PRINT"{HOME}{4 DOWN} 3 ";:GOTO45           :rem 220
38 PRINT"{HOME}{6 DOWN} 4 ";:GOTO45             :rem 0
39 PRINT"{HOME}{8 DOWN} 5 ";:GOTO45            :rem 36
40 PRINT"{HOME}{10 DOWN} 6 ";:GOTO45           :rem 63
41 PRINT"{HOME}{12 DOWN} 7 ";:GOTO45           :rem 99
42 PRINT"{HOME}{14 DOWN} 8 ";:GOTO45          :rem 135
43 PRINT"{HOME}{16 DOWN} 9 ";:GOTO45          :rem 171
```

```
44 PRINT"{HOME}{18 DOWN}1Ø ";                          :rem 26
45 X1=VAL(LEFT$(X$,1)):X2=VAL(MID$(X$,2,1)):X3=VAL
   (MID$(X$,3,1)):X4=VAL(RIGHT$(X$,1))                :rem 87
46 P=Ø:T=Ø                                             :rem 34
47 P=P+1:ONX1GOTO51,52,53,54,55,56                     :rem 140
48 P=P+1:ONX2GOTO51,52,53,54,55,56                     :rem 142
49 P=P+1:ONX3GOTO51,52,53,54,55,56                     :rem 144
5Ø P=P+1:ONX4GOTO51,52,53,54,55,56                     :rem 137
51 PRINT"{BLK}{RVS}{2 SPACES}{OFF} ";:POKES2,135:G
   OTO57                                               :rem 56
52 PRINT"{WHT}{RVS}{2 SPACES}{OFF} ";:POKES2,159:G
   OTO57                                               :rem 180
53 PRINT"{RED}{RVS}{2 SPACES}{OFF} ";:POKES2,175:G
   OTO57                                               :rem 202
54 PRINT"{CYN}{RVS}{2 SPACES}{OFF} ";:POKES2,191:G
   OTO57                                               :rem 76
55 PRINT"{PUR}{RVS}{2 SPACES}{OFF} ";:POKES2,2Ø1:G
   OTO57                                               :rem 66
56 PRINT"{GRN}{RVS}{2 SPACES}{OFF} ";:POKES2,2Ø9:G
   OTO57                                               :rem 205
57 FORL=1TO99:NEXTL:POKES2,Ø:FORL=1TO25Ø:NEXTL:ONP
   GOTO48,49,5Ø                                         :rem 1
58 ONTGOTO79,8Ø,81,82                                  :rem 198
59 PRINT" ";:IFB=ØTHEN66                               :rem 141
6Ø GOSUB91                                             :rem 8Ø
61 ONBGOTO62,63,64,65                                  :rem 169
62 PRINT"{BLK}Q";:GOTO66                               :rem 180
63 PRINT"{BLK}QQ";:GOTO66                              :rem 134
64 PRINT"{BLK}QQQ";:GOTO66                             :rem 88
65 PRINT"{BLK}QQQQ";:GOTO95                            :rem 44
66 IFW=ØTHEN73                                         :rem 88
67 GOSUB92                                             :rem 88
68 ONWGOTO69,7Ø,71,72                                  :rem 198
69 PRINT"{BLK}{RVS}Q{OFF}":GOTO73                      :rem 34
7Ø PRINT"{BLK}{RVS}QQ{OFF}":GOTO73                     :rem 235
71 PRINT"{BLK}{RVS}QQQ{OFF}":GOTO73                    :rem 189
72 PRINT"{BLK}{RVS}QQQQ{OFF}"                          :rem 178
73 PRINT"{BLK}{HOME}{2Ø DOWN}":PRINT"{21 SPACES}"
                                                       :rem 61
74 PRINT"{HOME}{2Ø DOWN}"                              :rem 163
75 IFX<>1ØTHEN16                                       :rem 196
76 FORL=1TO15:FORM=2ØØTO22Ø+L*2:POKES2,M:NEXTM:NEX
   TL:POKES2,Ø                                         :rem 154
77 PRINT"{HOME}{19 DOWN}":PRINT"{3 SPACES}";
                                                       :rem 219
78 T=T+1:ONA1GOTO51,52,53,54,55,56                     :rem 129
79 T=T+1:ONA2GOTO51,52,53,54,55,56                     :rem 131
8Ø T=T+1:ONA3GOTO51,52,53,54,55,56                     :rem 124
81 T=T+1:ONA4GOTO51,52,53,54,55,56                     :rem 126
```

```
82 PRINT"{4 SPACES}":PRINT"{BLK}CORRECT COLORS RET
   URN"                                          :rem 154
83 IFPEEK(197)<>15THEN83                         :rem 139
84 GETT$:PRINT"{WHT}{CLR}{2 DOWN} TOO BAD YOU MISS
   ED!":PRINT"{2 DOWN} {WHT}10 TRIES IS ENOUGH."
                                                 :rem 207
85 FORL=1TO6:POKES2,160:FORM=1TO400:NEXTM:POKES2,0
   :FORM=1TO400:NEXTM:NEXTL                      :rem 89
86 GETT$:PRINT"{CLR}{2 DOWN}{BLK}{2 SPACES}WANT TO
    PLAY AGAIN?":PRINT:PRINT:PRINT"{5 SPACES}YES O
   R NO?"                                        :rem 207
87 IFPEEK(197)=11THENGETT$:GOTO4                 :rem 226
88 IFPEEK(197)=28THENPOKE36879,27:GOTO90         :rem 157
89 GOTO87                                        :rem 25
90 GETT$:PRINT"{CLR}{9 DOWN}{7 SPACES}{YEL}{RVS}CH
   ICKEN!!":PRINT:PRINT:END                      :rem 126
91 FORL=200TO254:POKES2,L:NEXTL:POKES2,0:RETURN
                                                 :rem 57
92 FORL=200TO128STEP-1:POKES1,L:NEXTL:POKES1,0:RET
   URN                                           :rem 210
93 PRINT"ILLEGAL INPUT!":POKES1+2,200:FORL=1TO500:
   NEXTL:POKES1+2,0                              :rem 110
94 FORL=1TO999:NEXTL:GOSUB106:GOTO16             :rem 76
95 PRINT:PRINT:PRINT"{GRN}{2 SPACES}YOU W I N !! R
   ETURN"                                        :rem 248
96 FORM=250TO240STEP-1:POKES2,M:NEXTM:FORM=240TO25
   0:POKES2,M:NEXTM:POKES2,0                     :rem 105
97 IFPEEK(197)<>15THEN96                         :rem 148
98 PRINT"{CLR}{5 DOWN}"                          :rem 42
99 IFX=1THENPRINT"{YEL}{5 SPACES}LUCKY GUESS!":GOT
   O104                                          :rem 157
100 IFX=2ORX=3THENPRINT"{GRN}{6 SPACES}EXPERT!!!":
    GOTO104                                      :rem 177
101 IFX=4ORX=5ORX=6THENPRINT"{WHT}{4 SPACES}PRETTY
     GOOD!":GOTO104                               :rem 0
102 IFX=7ORX=8THENPRINT"{WHT}{9 SPACES}SO SO!":GOT
    O104                                         :rem 206
103 PRINT"{WHT} YOU BARELY GOT IT!"              :rem 206
104 FORL=1TO50:FORM=248TO253:POKES2,M:NEXTM:FORM=2
    53TO248STEP-1:POKES2,M:NEXTM:NEXTL           :rem 3
105 POKES2,0:GOTO86                              :rem 135
106 PRINT"{2 UP}":PRINT"{21 SPACES}":PRINT"{2 UP}"
    :RETURN                                      :rem 220
```

Therapy

Steven Rubio

Interactive games have always had a certain appeal, and "Therapy" is no exception. It'll never replace Freud, but it may just cure your blues. For the expanded VIC-20.

Eliza, the computer psychotherapist, is probably the most famous of all programs dealing with artificial intelligence. Written in LISP by Joseph Weizenbaum in 1966, *Eliza* has run on computers of all sizes and types, including home computers programmed in BASIC, in the ensuing years. The off-the-wall pronouncements of *Eliza* often elicit laughter and vexation.

There is something fascinating about carrying on a seemingly reasonable conversation with a machine. I still remember the thrill when I first learned my VIC could ask me a question (what is your name?) and remember the answer. This thrill is what prompted me to write "Therapy."

A Smarter Therapist

Why another version of *Eliza?* Mainly because *Eliza*, when written in BASIC, is extremely slow and takes as much as ten seconds to respond to your comments. It seemed to me that for a therapist, *Eliza* was a bit stand-offish and rather dumb.

The problem in BASIC is that *Eliza* tries for too much. Searches of fifty keywords and a hundred responses slow *Eliza* down, and in its attempt to give meaningful comments to *all* the user's statements, it consumes a lot of time for only occasional, if spectacular, success.

This is all right, since Weizenbaum never intended the program to substitute for actual therapy. But when showing off your computer to friends at your next get-together, it might be fun to have a program to demonstrate your machine's "intelligence."

Therapy will run on any expanded VIC.

Therapy

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
100 PRINTCHR$(142)CHR$(8)CHR$(30):POKE36879,8:GOSU
    B1230:POKE198,0                              :rem 11
105 Q=0:QD=0                                     :rem 144
110 PRINTCHR$(147);"HELLO. I'M DR. ROM.{3 SPACES}W
    HAT'S YOUR NAME?"                            :rem 40
115 GOSUB1160:A$=P1$:PRINT                       :rem 39
120 PRINT"IN ONE WORD, ";A$;",":PRINT"WHAT IS YOUR
     PROBLEM?":GOSUB1160:B$=P1$                  :rem 14
130 PRINT:PRINTB$;"...?":PRINT:PRINT"CAN YOU TELL
    {SPACE}ME MORE?"                             :rem 108
140 GOSUB1160:GOSUB900                           :rem 48
150 PRINT:PRINT"I UNDERSTAND ";B$:PRINT"IS DIFFICU
    LT FOR YOU."                                 :rem 99
160 GOSUB1160:IFP1$="NO"THENPRINT"MAYBE I'M NOT QU
    ITE{3 SPACES}UNDERSTANDING..."               :rem 111
170 PRINT:PRINT"CAN YOU BE MORE{7 SPACES}SPECIFIC?
     HOW IS":PRINTB$;" A PROBLEM?"                :rem 5
180 GOSUB1160:GOSUB900                           :rem 52
190 PRINT:PRINT"HOW DOES THIS MAKE YOUFEEL, ";A$;"
    ?":GOSUB1160:C$=P1$                          :rem 73
200 PRINTCHR$(147)                               :rem 13
205 PRINT"SO WHAT YOU'RE SAYING,";A$;", IS THAT YO
    UR":PRINT"PROBLEM WITH ";B$                  :rem 80
210 PRINT"IS MAKING YOU FEEL ":PRINTC$;".":GOSUB11
    60                                           :rem 45
220 PRINT:PRINT"CAN YOU ELABORATE ON{2 SPACES}YOUR
     FEELINGS?":GOSUB1160:GOSUB900               :rem 215
230 PRINT:PRINT"HAS THIS BEEN A{7 SPACES}PROBLEM F
    OR YOU BEFORE? (YES OR NO)":GOSUB1160        :rem 133
240 IFP1$<>"NO"THEN260                           :rem 236
250 PRINT"I SEE. THEN THIS NEW{2 SPACES}SITUATION
    {SPACE}MUST BE{5 SPACES}DIFFICULT FOR YOU.":GO
    TO320                                        :rem 81
260 PRINT:PRINT"DID YOU ALSO FEEL":PRINTC$;" THEN?
    "                                            :rem 100
270 GOSUB1160:PRINT"TELL ME MORE."               :rem 16
280 GOSUB1160:GOSUB900                           :rem 53
290 PRINTCHR$(147)"I THINK WE HAVE{7 SPACES}SOMETH
    ING HERE. DO{4 SPACES}YOU SEE A PATTERN?"
                                                 :rem 236
300 GOSUB1160:PRINT:PRINT"GO ON..."              :rem 106
310 GOSUB1160:PRINT"THIS SOUNDS DIFFICULT FOR YOU.
    ":GOSUB1160                                  :rem 41
320 PRINT:PRINT"DO YOU HAVE A PLAN TO DEAL WITH TH
    IS CURRENTSITUATION?";                       :rem 215
```

```
330 PRINT" (YES OR NO)":GOSUB1160          :rem 105
340 IFP1$<>"YES"THEN350                     :rem 65
343 PRINT"DO YOU THINK THIS PLANWILL BE SUCCESSFUL
    ?":GOTO360                              :rem 230
350 PRINT:PRINT"WHY DON'T YOU MAKE A{2 SPACES}LIST
     OF POSSIBLE{6 SPACES}SOLUTIONS, THEN."
                                            :rem 107
360 GOSUB1160:GOSUB900                      :rem 52
370 FORT=1TO500:NEXTT:PRINTCHR$(147)        :rem 253
380 PRINT"OKAY, WHAT SINGLE WORDBEST DESCRIBES";
                                            :rem 251
385 PRINT" HOW YOUARE FEELING RIGHT NOW?"   :rem 223
390 GOSUB1160:D$=P1$::PRINT:PRINTD$;"...?":rem 224
400 GOSUB1160:GOSUB900:PRINT                :rem 246
410 PRINT"I'M THINKING OF DOING SOMETHING HERE.":P
    RINT"LET'S TRY SOME WORD"               :rem 94
430 PRINT"ASSOCIATION AND SEE{3 SPACES}WHERE IT LE
    ADS US."                                :rem 172
440 PRINT"WHAT DO YOU THINK{5 SPACES}(YES OR NO)?"
    :GOSUB1160                              :rem 236
450 IFP1$="YES"THEN490                      :rem 11
460 PRINT:PRINT"YOU SEEM TO BE HAVING SOME PROBLEM
    S WITH{4 SPACES}THIS."                  :rem 122
470 PRINT"CAN YOU TELL ME ABOUT IT?":GOSUB1160:IFP
    1$="NO"THEN840                          :rem 46
480 PRINT:PRINT"I REALLY THINK A WORD ASSOCIATION
    {SPACE}WOULD BE{2 SPACES}USEFUL RIGHT NOW."
                                              :rem 4
490 PRINT:PRINT"LET'S DO IT."               :rem 242
500 PRINT"I'LL SAY A WORD. YOU{2 SPACES}SAY THE FI
    RST WORD{4 SPACES}THAT COMES TO YOUR{4 SPACES}
    MIND."                                  :rem 133
510 REM ***WORD ASSOCIATION***              :rem 239
520 FORT=1TO5000:NEXTT:PRINTCHR$(147);"DOG":PRINT:
    GOSUB1160                               :rem 204
530 PRINT:PRINT"DRINK":PRINT:GOSUB1160      :rem 241
540 PRINT:PRINT"HOME":PRINT:GOSUB1160:E$=P1$
                                            :rem 40
550 PRINT:PRINTB$:PRINT:GOSUB1160:F$=P1$    :rem 35
560 PRINT:PRINT"FEELINGS":PRINT:GOSUB1160   :rem 201
570 PRINT:PRINT"FUN":PRINT:GOSUB1160:G$=P1$
                                            :rem 237
580 PRINT:PRINT"MOM":PRINT:GOSUB1160:I$=P1$
                                            :rem 240
590 PRINT:PRINTC$:PRINT:GOSUB1160:J$=P1$    :rem 44
600 FORT=1TO1000:NEXTT:PRINTCHR$(147)       :rem 37
610 PRINT"I NOTICED WHEN I SAID HOME THAT YOU SAID
    ":PRINTE$;"."                           :rem 168
```

```
620 PRINT"DOES THIS SOMEHOW{5 SPACES}REFLECT HOW Y
    OU FEEL{2 SPACES}ABOUT YOURSELF?"       :rem 45
630 PRINT"YES OR NO":GOSUB1160:IFP1$<>"YES"THEN650
                                            :rem 2
640 PRINT:PRINT"IN WHAT WAY?":GOSUB1160:GOSUB900
                                            :rem 2
650 PRINT:PRINT"HOW DOES THIS RELATE{2 SPACES}TO Y
    OUR PROBLEM WITH":PRINTB$               :rem 173
660 GOSUB1160:GOSUB900:PRINT:PRINT"WHEN I SAID ";B
    $:PRINT"YOU SAID ";F$                   :rem 20
670 PRINT"WHAT DO YOU THINK THISMEANS?":GOSUB1160:
    GOSUB900                                :rem 112
680 PRINT:PRINT"ARE YOU DISTRESSED? DOYOU WANT A K
    LEENEX?":GOSUB1160                      :rem 28
690 IFP1$<>"YES"THEN710                     :rem 73
700 PRINT"HERE.":FORT=1TO1000:NEXTT         :rem 206
710 PRINT:PRINT"IT'S INTERESTING THAT WHEN I SAID
    {SPACE}FUN,{2 SPACES}YOU SAID ";G$      :rem 57
720 GOSUB1160:GOSUB900:PRINTCHR$(147);"HMMMM..."
                                            :rem 110
730 PRINT:PRINT"IT SEEMS TO ME, ";A$;","    :rem 248
735 PRINT"THAT THIS ALL TIES IN TO YOUR PROBLEM"
                                            :rem 129
740 PRINT"WITH ";B$                         :rem 73
750 GOSUB1160:GOTO770                       :rem 245
760 REM ***DREAMS***                        :rem 57
770 PRINT:PRINT"LET'S TRY A DIFFERENT":PRINT"APPRO
    ACH,";A$                                :rem 145
780 PRINT"TELL ME ABOUT ONE OF{2 SPACES}YOUR DREAM
    S.":GOSUB1160:GOSUB1040:IFQD=1THEN840 :rem 246
790 PRINT:PRINT"HOW WOULD YOU DESCRIBEYOUR FEELING
    S IN THE{2 SPACES}DREAM?":GOSUB1160     :rem 45
800 PRINT:PRINT"DID THE DREAM HAVE{4 SPACES}ANYTHI
    NG TO DO WITH":PRINTI$                  :rem 119
810 GOSUB1160:FORT=1TO1000:NEXTT            :rem 245
820 REM ***ALL DONE***                      :rem 121
830 PRINT:PRINT"{CLR}I THINK WE'RE MOVING
    {2 SPACES}IN A GOOD DIRECTION.":PRINT   :rem 78
840 PRINT"WE'VE DISCUSSED YOUR{2 SPACES}PROBLEM WI
    TH":PRINTB$:PRINT"AND HOW THIS MAKES YOU"
                                            :rem 173
850 PRINT"FEEL ";C$;";"                     :rem 230
860 PRINT"AND DISCUSSED SOME{4 SPACES}POSSIBLE SOL
    UTIONS."                                :rem 124
870 PRINT:PRINT"I SEE YOUR TIME IS UP.SEE YOU NEXT
     WEEK."                                 :rem 189
880 END                                     :rem 119
890 REM ***KEYWORDS***                      :rem 249
900 IFQ>0THENRETURN                         :rem 246
```

```
910 FORJ=1TOLEN(P1$)-5                          :rem 19
920 IFMID$(P1$,J,5)<>" FUN "THEN930             :rem 103
925 PRINT:PRINT"WHAT ARE YOUR{9 SPACES}FEELINGS AB
    OUT FUN?":GOTO950                           :rem 148
930 NEXTJ                                       :rem 37
940 RETURN                                      :rem 125
950 GOSUB1160:Q=1:PRINT:PRINT"THESE FEELINGS SEEM
    {3 SPACES}IMPORTANT."                       :rem 141
960 GOSUB1160:RETURN                             :rem 1
1040 REM ***DREAM KEYWORD SEARCH***             :rem 233
1050 FORJ=1TOLEN(P1$)-7                         :rem 65
1060 IFMID$(P1$,J,7)=" DON'T "THEN1120          :rem 243
1070 NEXTJ                                      :rem 81
1080 FORJ=1TOLEN(P1$)-6                         :rem 67
1090 IFMID$(P1$,J,6)=" DONT "THEN1120           :rem 206
1100 NEXTJ                                      :rem 75
1110 RETURN                                     :rem 163
1120 PRINTCHR$(147)"WHY DO YOU SUPPOSE THAT IS?":G
     OSUB1160:GOSUB900                          :rem 27
1130 PRINT"THIS MAY BE SOMETHING THAT WE'LL WANT"
                                                :rem 176
1140 PRINT"TO DISCUSS LATER. WE MAY FIND THAT IT"
                                                :rem 112
1150 PRINT"RELATES TO YOUR PROBLEM WITH ";B$:QD=1:
     RETURN                                     :rem 223
1160 REM ***COMMODORE PUNCTUATION INPUT***      :rem 55
1170 P1$=""                                     :rem 239
1180 GETP2$:IFP2$=""THEN1180                    :rem 57
1190 PRINTP2$;                                  :rem 57
1200 IFP2$=CHR$(13)THENRETURN                   :rem 250
1210 P1$=P1$+P2$                                :rem 28
1220 GOTO1180                                   :rem 200
1230 REM ***INTRODUCTION***                     :rem 72
1240 PRINTCHR$(147);TAB(6)"THERAPY"             :rem 60
1250 PRINT:PRINT"WOULD YOU LIKE AN{5 SPACES}INTROD
     UCTION (Y/N)"                              :rem 101
1260 GETQ$:IFQ$<>"Y"ANDQ$<>"N"THEN1260          :rem 191
1270 IFQ$="N"THENRETURN                         :rem 172
1280 PRINTCHR$(147);"WELCOME TO YOUR{7 SPACES}THER
     APY SESSION.{6 SPACES}DR. ROM";            :rem 31
1285 PRINT" WILL BE WITH{2 SPACES}YOU IN A ";
                                                :rem 172
1290 PRINT"MOMENT. WHILEYOU ARE WAITING, HERE ARE
     {SPACE}SOME HELPFUL"                       :rem 104
1300 PRINT"SUGGESTIONS ON HOW TO GET THE MOST OUT
     {SPACE}OF{3 SPACES}YOUR THERAPY SESSION."
                                                :rem 109
1305 PRINT:PRINT                                 :rem 29
```

```
1310 PRINT"AS WITH MOST THINGS INLIFE, WITH THERAP
     Y,{3 SPACES}THE MORE YOU ";              :rem 42
1320 PRINT"PUT IN,{2 SPACES}THE MORE{2 SPACES}YOU
     {SPACE}GET OUT.YOU MAY FIND IT FUN TOTRY AND
     {SPACE}TRIP ";                           :rem 228
1330 PRINT"UP THE{3 SPACES}DOCTOR; MAKE FUN OF
     {3 SPACES}HIS GRAMMAR, OR INSULTHIM MERCILESS
     LY."                                     :rem 175
1332 PRINT:PRINTCHR$(18)"HIT ANY KEY"         :rem 210
1335 POKE198,0:WAIT198,1                      :rem 103
1340 PRINT"{CLR}{DOWN}HOWEVER, EVEN THOUGH
     {2 SPACES}THIS IS A PARLOR{6 SPACES}GAME, YOU
      MAY STILL{3 SPACES}FIND ";              :rem 121
1350 PRINT"YOURSELF HAVING{2 SPACES}INTERESTING, A
     ND EVEN IMPORTANT,";                     :rem 110
1360 PRINT" INSIGHTS.{2 SPACES}THIS WILL ONLY HAPP
     EN IF YOU{2 SPACES}TRY YOUR BEST TO UTILIZE "
     ;                                        :rem 172
1370 PRINT"THIS{7 SPACES}SESSION AS AN{9 SPACES}EN
     JOYABLE WAY TO MULL OVER THE ";          :rem 218
1380 PRINT"PROBLEMS AND PEEVES OF LIFE."      :rem 127
1390 PRINT:PRINTCHR$(18)"HIT ANY KEY"         :rem 214
1400 POKE198,0:WAIT198,1                       :rem 96
1410 PRINTCHR$(147):PRINT"I SEE THE DOCTOR IS IN N
     OW."                                     :rem 115
1420 PRINT:PRINT"TO TALK TO DR. ROM,{3 SPACES}JUST
      TYPE IN YOUR"                           :rem 29
1430 PRINT"RESPONSE; AND HIT ";CHR$(18):PRINT"RETU
     RN";CHR$(146);" WHEN YOU ARE"            :rem 138
1440 PRINT"FINISHED.":PRINT:PRINT:PRINT"ENJOY YOUR
      THERAPY SESSION."                       :rem 238
1450 PRINT:PRINTCHR$(18);" HIT ANY KEY TO BEGIN"
                                               :rem 22
1460 POKE198,0:WAIT198,1:RETURN               :rem 128
```

5
Adventure Games

Adventure Games

Adventure games, like art, have been said to imitate life—but what a life it can be.

Take "Cave-In," by Paul L. Bupp and Stephen P. Drop. You've just been named superintendent at one of the world's deepest mines, and the first thing you discover is that the mine is about to cave in. It's up to you to rescue the miners. And you see it all in 3-D!

Dave and Casey Gardner's "Castle Dungeon" gives you a mission, too: to locate and defuse bombs hidden in a dark, dripping maze. Fortunately, you have a lantern, which reveals the passages as you go. Unfortunately, its light is dim—and you've only got minutes before the bombs explode.

Yet another form of adventure is the text adventure, where you and the computer communicate using words and phrases. In David Florance's "Time Capsule," an exciting text adventure set in the year 2184, you've accidentally stumbled across a time machine and been thrown far from your own time. What do you discover? A plot to invade your own time. What can you do? Only one thing: Escape and sound the alarm.

Finally, George Miller's "Sigma Mission" puts you at the mercy of a recalcitrant computer that has taken over a mission into deep space. The game uses innovative programming techniques to create an exciting text adventure on an *unexpanded* VIC.

Many people find adventure games unusually stimulating and challenging. As you play these games and look at the programs, you'll undoubtedly agree: Adventuring with your VIC is fun.

Cave-In

Paul L. Bupp and Stephen P. Drop

"Cave-In" is an excellent three-dimensional maze game which uses a screen-flipping technique to swap screen displays. The game requires a joystick and runs on the unexpanded VIC-20.

You are the newly appointed foreman of a mining operation, and after completing your initial inspection, you believe that a cave-in is imminent. You realize that you must explore every tunnel to find and rescue all of the miners. Considering your unfamiliarity with the mine, you decide to make a map of the workings as you travel.

Fortunately, you have your trusty VIC to help you. To refer to your map, push the fire button on the joystick. Push it again and you return to the mine. The dark circle on the map is where you started, and where you must return in order to escape the mine safely.

Just as you expected, no sooner do you find the last miner and warn him of the danger than the cave-in begins. Now you have to get out before the falling rock traps you.

Aren't you glad you made the map?

Other Game Controls

You may view instructions at any time by pressing the f1 special function key. However, once you see the instructions, you face a fresh maze upon returning to the game. To travel through the tunnels, change directions by moving the joystick right or left, and move forward by pushing the stick forward.

Observe some precautions when typing in this program. First, it requires using the Commodore key at the lower left of the keyboard. Some of the graphics symbols must be typed while this key is held down (like the SHIFT key) to correctly print the characters needed to build the maze. Second, each line must be entered *exactly* as printed, without extra spaces, if it is to fit into memory. Refer to Appendix A, "A Beginner's Guide to Typing In Programs" for assistance. This program uses all but about 15 of the 3583 available memory locations, and it will *not* run correctly with any memory expansion boards. Once you're comfortable with the beginner's level, try

the advanced game. It's basically similar to the beginner's version, but there is one catch: You lose the map after the cave-in starts, so you must rely on your memory to recall the mazelike passages.

Cave-In

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 POKE56,28:CLR:DEFFNW(X)=PEEK(37151)AND32:B=36865
  :GOSUB6                                    :rem 186
2 QQ=37154:PRINT"{CLR}{WHT}"SPC(73)"CAVE-IN"SPC(10
  1)"{BLK}F1 FOR HELP":GOTO56                :rem 130
3 A$="+01-22-01+22+01":F=A+VAL(MID$(A$,D*3+1,3))*V
                                             :rem 248
4 X=VAL(MID$(A$,D*3+4,3)):L=F+X:R=F-X:RETURN
                                             :rem 118
5 PRINTSPC(230)"V":RETURN                    :rem 194
6 Y=30:POKEB+14,42:POKEB+1,150:GOTO8         :rem 112
7 Y=28:POKEB+14,107:POKEB+1,22:POKEB,25:POKEA,VAL(
  MID$("235241243242",D*3+1,3))              :rem 146
8 POKE648,Y:IFFNW(W)THENRETURN               :rem 249
9 GOTO8                                      :rem 170
10 PRINTSPC(207)"M{DOWN}{LEFT}{M}{DOWN}{LEFT}N":RE
   TURN                                      :rem 186
11 PRINTSPC(161)"M{DOWN}M{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{2 DOWN}{2 LEFT}N{UP}N"
   :RETURN                                   :rem 79
12 PRINTSPC(92)W$MID$(X$,37)"{2 DOWN}{3 LEFT}N{UP}
   N{UP}N":RETURN                            :rem 81
13 PRINT"{DOWN}{RIGHT}"W$MID$(X$,19)"{2 DOWN}
   {3 LEFT}N{UP}N{UP}N":RETURN               :rem 34
14 PRINT"M"X$"{LEFT}N":RETURN                :rem 72
15 PRINTSPC(209)"N{DOWN}{LEFT}{G}{DOWN}{LEFT}M":RE
   TURN                                      :rem 191
16 PRINTSPC(188)"N{UP}N{2 DOWN}{2 LEFT}{G}{DOWN}
   {LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}M{DOWN}M":R
   ETURN                                     :rem 87
17 PRINTSPC(146)"N{UP}N{UP}N{3 DOWN}{3 LEFT}{G}"MI
   D$(Y$,40)W$:RETURN                        :rem 55
18 PRINTSPC(83)"N{UP}N{UP}N{3 DOWN}{3 LEFT}{G}"MID
   $(Y$,22)W$:RETURN                          :rem 8
19 PRINTSPC(20)"N"Y$"M{HOME}":RETURN         :rem 93
20 PRINTSPC(229)"P{DOWN}{LEFT}{T}":RETURN    :rem 66
21 PRINTSPC(205)"{T}P{DOWN}{LEFT}{M}{DOWN}{2 LEFT}
   {@}@":RETURN                              :rem 141
22 PRINTSPC(158)"{2 T}P"MID$(X$,40)"{3 LEFT}{3 T}"
   :RETURN                                   :rem 153
```

```
23 PRINTSPC(89)"[2 T]P"MID$(X$,22)"{3 LEFT}[3 T]":
   RETURN                                          :rem 109
24 PRINT"{DOWN}P"MID$(X$,4)"{LEFT}[T]":RETURN
                                                   :rem 225
25 PRINTSPC(231)"O{DOWN}{LEFT}[T]":RETURN          :rem 63
26 PRINTSPC(210)"O[T]{DOWN}{2 LEFT}[G]{DOWN}{LEFT}
   L[@]":RETURN                                    :rem 157
27 PRINTSPC(168)"O[2 T]{DOWN}{3 LEFT}[G]"MID$(Y$,4
   3)"[3 T]":RETURN                                :rem 88
28 PRINTSPC(105)"O[2 T]{DOWN}{3 LEFT}[G]"MID$(Y$,2
   5)"[3 T]":RETURN                                :rem 80
29 PRINTSPC(42)"O"MID$(Y$,4)"[T]":RETURN           :rem 213
30 PRINTSPC(230)"[T]{DOWN}{LEFT}[T]":RETURN:rem 14
31 PRINTSPC(207)"[3 T]{2 DOWN}{3 LEFT}[3 @]":RETUR
   N                                               :rem 237
32 PRINTSPC(161)MID$(Z$,13)SPC(147)MID$(Z$,13):RET
   URN                                             :rem 104
33 PRINTSPC(92)MID$(Z$,7)"{DOWN}"SPC(251)MID$(Z$,7
   ):RETURN                                        :rem 51
34 PRINTSPC(23)Z$SPC(154)SPC(245)Z$"{HOME}":RETURN
                                                   :rem 160
35 PRINT"{CLR}{WHT}":FORV=0TO5:GOSUB3:IFPEEK(F)=32
   THENPRINT"{HOME}":ONVGOSUB34,33,32,31,30:GOTO41
                                                   :rem 226
36 PRINT"{HOME}":IFPEEK(L)=32THENONV+1GOSUB14,13,1
   2,11,10,5:GOTO38                                :rem 86
37 ONV+1GOSUB24,23,22,21,20,5                      :rem 62
38 PRINT"{HOME}":IFPEEK(R)=32THENONV+1GOSUB19,18,1
   7,16,15,5:GOTO40                                :rem 112
39 ONV+1GOSUB29,28,27,26,25,5                      :rem 89
40 NEXT                                            :rem 163
41 GOSUB77:W=PEEK(37151):IFY=30GOTO44              :rem 251
42 IF(WAND32)=0THENPOKEB,Z:GOSUB6:POKEA,X          :rem 134
43 GOTO41                                          :rem 5
44 IFK=0ANDP<TITHENX=PEEK(B)+1:POKEB,X:P=TI+40:IFX
   =122THENRETURN                                  :rem 167
45 IF(WAND4)GOTO51                                 :rem 70
46 V=1:GOSUB3:IFPEEK(F)<>32THENA=F:POKE30720+A,1:G
   OTO35                                           :rem 123
47 IFA=7397ANDK=0THENRETURN                        :rem 236
48 IFPEEK(A)<>13GOTO51                             :rem 47
49 K=K-1:PRINTSPC(116)"MAN FOUND"SPC(34)"MEN LEFT=
   "K:POKEA,160                                    :rem 178
50 IFK=0THENPRINTSPC(72)"CAVE-IN":IFGTHENO=1
                                                   :rem 43
51 IF(WAND16)=0THEND=VAL(MID$(D$,D+3,1)):GOTO35
                                                   :rem 229
52 POKEQQ,127:X=PEEK(37152)AND128:POKEQQ,255:IFX=0
   THEND=VAL(MID$(D$,D+1,1)):GOTO35                :rem 179
```

```
53 IF(WAND8)=ØTHEND=VAL(MID$(D$,D+4,1)):GOTO35
                                                :rem 185
54 IF(WAND32)=ØANDO=ØTHENZ=PEEK(B):X=PEEK(A):GOSUB
   7                                            :rem 201
55 GOTO41                                        :rem 8
56 D=3:D$="+Ø2-44-Ø2+44":PRINTSPC(91)"{WHT}MINE BE
   ING DUG                                      :rem 131
57 POKE648,28:A=7397:X$="{21 SPACES}":PRINT"{CLR}
   {RVS}{CYN} "X$;:FORW=1TO21                   :rem 142
58 PRINT"{OFF}{BLU}"X$"{CYN}{RVS} ";:NEXT:PRINTX$"
   {HOME}":POKE7673,16Ø:POKE38393,3:PRINTSPC(141)"
   {WHT}E                                       :rem 131
59 GOSUB77:X=INT(RND(1)*4):Y=X                   :rem 83
6Ø W=A+VAL(MID$(D$,X*3+1,3))                     :rem 237
61 IFPEEK(W)=32THENZ=Ø:POKEW,X:POKEA+VAL(MID$(D$,X
   *3+1,3))/2,16Ø:A=W:GOTO59                     :rem 15
62 X=(X+1)*-(X<3):IFX<>YGOTO6Ø                   :rem 4
63 X=PEEK(A):POKEA,16Ø:IFZ=ØTHENPOKEA,13:Z=1:K=K+1
                                                :rem 76
64 IFX<>5THENA=A-VAL(MID$(D$,X*3+1,3)):GOTO59
                                                :rem 131
65 W$="M{DOWN}M{DOWN}M":X$="{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
   {LEFT}{M}{DOWN}{LEFT}{M}{DOWN}{LEFT}{M}{DOWN}
                                                :rem 101
66 Y$="{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}{G}{DOWN}{LEFT}
   {G}{DOWN}{LEFT}{G}{DOWN}{LEFT}                :rem 31
67 D$="3Ø123Ø1":GOSUB6:Z$="{19 T}":POKEA,2Ø9:GOSUB
   35                                           :rem 133
68 GOSUB7:POKE217,156:POKE218,156                :rem 188
69 PRINT"{HOME}{RVS}{CYN}PRESS THE FIRE BUTTON
   {OFF}{RIGHT}{WHT}TWICE=PLAY - ONCE=END        :rem 91
7Ø FORW=37933TO38329STEP22:FORX=ØTO18:POKEW+X,1:NE
   XT:NEXT                                      :rem 187
71 GOSUB77:IFFNW(X)GOTO71                        :rem 84
72 GOSUB6:PRINT"{CLR}                            :rem 155
73 IFFNW(X)=ØGOTO73                              :rem 157
74 FORW=ØTO3Ø:IFFNW(X)=ØGOTO88                   :rem 143
75 NEXT                                          :rem 171
76 POKE56,3Ø:CLR:END                             :rem 194
```

```
77 GETA$:IFA$<>CHR$(133)THENRETURN          :rem 83
78 POKEB,25:GOSUB6:PRINT"{CLR}PICK ONE{BLK}":PRINT
   "{DOWN} F1=NOVICE":PRINT"{DOWN} F3=ADVANCED
                                            :rem 20
79 PRINT"{DOWN} F5=OLD MAP":PRINT"{DOWN} F7=END":P
   RINT"{DOWN}{WHT}*CURRENT LEVEL           :rem 56
80 PRINT"{3 DOWN}GOAL-{BLK}FIND THE MINERS"SPC(7)"
   AND GET BACK                            :rem 112
81 PRINT"{WHT}{2 DOWN}JOYSTICK-{BLK}MOVE":PRINT"SE
   E MAP Q↑                                :rem 152
82 PRINT"SEE LEFT<W>SEE RIGHT"SPC(11)"V"SPC(18)"SE
   E BACK{HOME}{WHT}":IFGTHENPRINT"{3 DOWN}*":GOTO
   84                                      :rem 88
83 PRINT"{DOWN}*                           :rem 85
84 GETA$:IFA$<>""THENW=ASC(A$)-132:ONABS(W)GOTO86,
   87,68,72                                :rem 118
85 GOTO84                                   :rem 18
86 G=0:GOTO88                               :rem 5
87 G=1                                     :rem 36
88 O=0:K=0:PRINT"{CLR}":GOTO56            :rem 154
```

Castle Dungeon

Dave and Casey Gardner

Bombs with short fuses (not to mention locked doors, blind monsters, and bottomless pits) add urgency and danger to this all-graphics adventure game for the unexpanded VIC. A joystick is required.

Your job is to find three bombs hidden in the rooms and corridors of the castle dungeon. They were placed by an evil wizard who is trying to destroy the castle.

He also dug several seemingly bottomless pits to trap the unwary hero—and as if that were not enough, he added nine beasts to guard the bombs. Luckily for you, the beasts are blind and will attack only if you bump into them. However, if you are carrying the enchanted sword (which the wizard lost somewhere in the dungeon, many years ago), you will defeat the beasts. You will also need to find the magic key in order to open the locked doors.

A Light and Levitation

On your search through the dungeon you will be carrying a light which is only bright enough for you to see the area immediately around you. If you move too fast, you might fall into one of the bottomless pits and be lost forever. However, by standing next to a pit and pressing the L key, you can invoke a levitation spell which will allow you to cross over the pit without falling in.

The short fuses on the bombs give you only five minutes to locate and defuse all bombs. If you haven't found all three bombs by that time, they will explode and the castle will be destroyed. Each time you play, the wizard will place the various objects in different locations.

To save memory, the program is in two parts. The first part (Program 1) displays the title page and instructions and defines the programmable characters used in the second part.

A Special Filename

Type in Program 1 and SAVE it (if you are using a Datassette, change ,8 in line 28 of Program 1 to ,1). Then type in Program 2 and SAVE it as "D".

Here is what the program lines do:

Program 1 (VIC Loader)

Line(s)	Description
1	Clear the screen and lower the top of memory
2	Define variables for sounds and the screen
3–5	Display the title page
6–7	Play a tune
8–10	Complete the title page
11–22	Display instructions
23	Randomize (so each game will start differently)
24–25	Store character information in high memory
26–29	LOAD Program 2
30–41	Title page data
44–49	Character data

Program 2 (Main VIC Program)

Line(s)	Description
1–2	Initialize variables
3	Fill the screen with black spaces
4–5	Place the maze
6	Place doors
7	Place room floors
8	Place bombs
9	Place beasts
10	Place key
11	Place sword
12	Place pits
13	Choose starting point, set the clock to zero
14	Read joystick
15	Check if time is up
16–20	Set direction
21	If wall in way, stop
22	Sword?
23	Beast?
24	Door and no key?
25	Key?
26	Levitation spell?
27	Pit and no spell?
28	Pit and spell?
29	Bomb?
30	If not moving, jump ahead to line 44
31–33	Light up area around player
34	If key or sword found, make sound
35	If player fell in pit, jump to ending sequence

36–37	If player levitated over pit, redraw pit
38	Cancel levitation spell
39	Make player movement noise
40–43	Darken area just vacated
44	If third bomb found, jump to ending sequence
45	Do it again
46–47	Successful quest ending
48–55	Unsuccessful quest ending
56–62	Maze data
63	Door data
64–66	Room floor data
67–69	Subroutine for randomly placing objects
70–72	Sound subroutine for sword and key
73–74	Sound subroutine for locked door
75	Sound subroutine for bomb found
76–77	Sound and ending subroutine for falling in pit
78–81	Subroutine for fighting beast
82	Sound subroutine for levitation spell

We would like to thank Don Brunner and Todd Andrews of Rose City Computer Associates, Newark, New York, for their technical assistance in preparing this program.

The joystick reading routine is from "The Joystick Connection," by Paul Bupp and Stephen Drop (*COMPUTE!'s First Book of VIC*).

Program 1. VIC Loader

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 PRINTCHR$(147):POKE52,29:POKE56,29:CLR  :rem 153
2 SV=36873:S1=SV+1:S2=SV+2:S3=SV+3:V=SV+5:SB=SV+6
                                          :rem 21
3 READL,N:IFN=-1THEN5                     :rem 43
4 FORJ=ØTON:READS:POKEL+J,S:NEXT:GOTO3    :rem 52
5 POKESB,11Ø:FORT=1TO5ØØ:NEXT             :rem 75
6 FORM=1TO3:READA,B,C,D,E                  :rem 3
7 POKES1,A:POKES2,B:POKES3,C:FORJ=15TOESTEP-1:POKE
  V,J:FORT=1TOD:NEXT:NEXT:NEXT            :rem 11
1Ø FORT=1TO2ØØØ:NEXT                     :rem 232
11 PRINTCHR$(147):FORT=1TO5ØØ:NEXT       :rem 113
12 POKESB,59:POKESV-4,242                :rem 248
13 FORT=1TO5ØØ:NEXT                      :rem 19Ø
14 PRINTCHR$(144)"{UP}{RIGHT}FIND AND DEFUSE THE"
                                          :rem 224
15 PRINT:PRINT" BOMBS HIDDEN IN THE"     :rem 149
16 PRINT:PRINT" DUNGEON. DON'T FALL"      :rem 56
```

```
17 PRINT:PRINT" INTO A PIT OR GET"          :rem 233
18 PRINT"{DOWN}{RIGHT}EATEN BY A BEAST."    :rem 78
19 PRINT"{DOWN}{RIGHT}PRESS THE 'L' KEY FOR"
                                            :rem 65
20 PRINT"{RIGHT}A LEVITATION SPELL."        :rem 62
21 PRINT:PRINT"{RIGHT}YOU HAVE 5 MINUTES"   :rem 19
22 PRINT"{DOWN}{RIGHT}TO COMPLETE YOUR":PRINT"
   {DOWN}{RIGHT}QUEST."                     :rem 167
23 POKE143,VAL(MID$(TI$,5,2))               :rem 91
24 READCL:IFCL=-1THEN26                     :rem 155
25 FORJ=CLTOCL+7:READCC:POKEJ,CC:NEXT:GOTO24
                                            :rem 139
26 PRINT:PRINT"(HIT ANY KEY TO BEGIN)"      :rem 143
27 GETA$:IFA$=""THEN27                      :rem 245
28 PRINT"{CLR}":FORJ=1TO3:POKESV+J,0:NEXT:S$="LO"+
   CHR$(34)+"D"+CHR$(34)+",8:"+CHR$(131)    :rem 192
29 FOR I=1TOLEN(S$):POKE630+I,ASC(MID$(S$,I)):NEXT
   :POKE198,I:END                           :rem 98
30 DATA7878,20,114,64,73,73,32,110,85,73,110,85,64
   ,73,112,64,75,85,64,73,85,73,110         :rem 14
31 DATA7900,20,93,32,93,93,32,93,93,93,93,93,64,73
   ,107,64,32,93,32,93,93,93,93             :rem 94
32 DATA7922,20,113,64,75,74,64,115,125,74,75,74,64
   ,75,109,64,73,74,64,75,125,74,75         :rem 34
33 DATA7792,17,85,64,73,112,64,110,85,64,75,64,114
   ,64,112,32,32,112,64,75                  :rem 75
34 DATA7814,16,93,32,32,107,64,115,74,64,73,32,93,
   32,93,32,32,107,64                       :rem 80
35 DATA7836,17,74,64,75,75,32,125,85,64,75,32,75,3
   2,74,64,75,109,64,73                     :rem 216
40 DATA -1,-1                               :rem 102
41 DATA0,0,219,36,5,0,236,231,36,5,237,231,226,100
   ,0                                       :rem 13
44 DATA7632,247,227,246,193,215,247,235,235,7640,2
   55,34,34,34,255,68,68,68                 :rem 162
45 DATA7648,255,231,195,129,129,131,199,255,7656,2
   55,191,95,64,90,186,255,255              :rem 76
46 DATA7664,191,121,112,1,0,135,55,115,7672,255,23
   9,247,231,195,195,231,255                :rem 184
47 DATA7424,255,255,255,255,255,255,255,255,7440,2
   55,255,255,129,129,255,255,255,7432      :rem 205
48 DATA255,255,255,255,255,255,255,255,7448,255,25
   3,251,247,143,207,175,255                :rem 210
49 DATA7456,191,121,112,1,0,135,55,115,7464,255,25
   5,231,0,0,231,255,255,-1                 :rem 98
```

Program 2. Main VIC Program

```
1 PRINTCHR$(147);CHR$(144):S1=36874:S2=S1+1:S3=S1+
  2:S4=S1+3:V=S1+4:SB=S1+5:CL=S1-5          :rem 255
2 C=30720:L=7680:MW=59:FC=0:PC=33:BT=8182:FV=15:FP
  =0:CS=0:POKESB,8:R=37154:AF=0:KF=0        :rem 215
3 POKEV,15:POKECL,255:FORJ=LTOL+505:POKEJ+C,0:POKE
  J,32:NEXT                                 :rem 236
4 READD:IFD=-1THEN6                         :rem 161
5 POKEL+D,MW:L=L+D:GOTO4                     :rem 20
6 L=7680:FORJ=1TO9:READD:POKEL+D,34:NEXT     :rem 19
7 FORJ=1TO46:READD:POKEL+D,33:NEXT          :rem 172
8 FORJ=1TO3:GOSUB67:POKEB+L,63:NEXT           :rem 4
9 FORJ=1TO9:GOSUB67:POKEB+L,36:NEXT          :rem 11
10 PC=32:GOSUB67:POKEB+L,61                 :rem 114
11 GOSUB67:POKEB+L,35                         :rem 5
12 FORJ=1TO3:GOSUB67:POKEB+L,60:NEXT         :rem 44
13 GOSUB67:M=B+L:TI$="000000"               :rem 106
14 POKER,127:JS=(PEEK(37137)AND28)OR(PEEK(37152)AN
   D128):JS=ABS(JS-100)/4-7:POKER,255       :rem 129
15 IFTI$>"000500"THEN48                     :rem 248
16 IFJS=6THENDR=-22                         :rem 153
17 IFJS=5THENDR=22                          :rem 108
18 IFJS=3THENDR=-1                          :rem 101
19 IFJS=11THENDR=1                          :rem 104
20 IFJS=7THENDR=0                            :rem 52
21 P=PEEK(DR+M):IFP=59THENDR=0              :rem 106
22 IFP=35THENCS=1:POKEBT,35:POKEBT+C,5:BT=BT+1
                                            :rem 123
23 IFP=36THEN78                             :rem 136
24 IFP=34ANDKF=0THENGOSUB73                 :rem 211
25 IFP=61THENKF=1:POKE8181,61:POKE8181+C,5:rem 240
26 GETL$:IFL$="L"THENLS=1:GOSUB82            :rem 30
27 IFP=60ANDLS<>1THENFP=1                    :rem 59
28 IFP=60ANDLS=1THENFP=2:PIT=M+DR           :rem 114
29 IFP=63THENAF=AF+1:POKEBT,63:POKEBT+C,5:BT=BT+1:
   GOSUB75                                   :rem 77
30 IFDR=0ANDFC=1THEN44                       :rem 86
31 POKEM,32:POKEM+C,7:POKEM+DR+C,7:POKEM+DR,58
                                            :rem 80
32 POKEM+DR+C-22,7:POKEM+DR+C+22,7:POKEM+DR+C+1,7:
   POKEM+DR+C-1,7                           :rem 37
33 POKEM+DR+C-23,7:POKEM+DR+C+23,7:POKEM+DR+C+21,7
   :POKEM+DR+C-21,7:FC=1                    :rem 189
34 IFP=35ORP=61THENGOSUB70                  :rem 150
35 IFFP=1THENPOKEM+DR,60:GOTO76             :rem 217
36 IFPS=1THENPOKEPIT,60:PS=0                 :rem 48
37 IFFP=2THENPS=1:FP=0                      :rem 123
38 LS=0:IFDR=0THEN44                        :rem 218
```

```
39 POKES3,240:FORT=1TO2:NEXT:POKES3,0          :rem 95
40 IFDR=-22THENPOKEM+C+23,0:POKEM+C+22,0:POKEM+C+2
   1,0:GOTO44                                  :rem 84
41 IFDR=1THENPOKEM+C-23,0:POKEM+C-1,0:POKEM+C+21,0
   :GOTO44                                     :rem 198
42 IFDR=-1THENPOKEM+C-21,0:POKEM+C+1,0:POKEM+C+23,
   0:GOTO44                                    :rem 242
43 POKEM+C-23,0:POKEM+C-22,0:POKEM+C-21,0      :rem 96
44 M=M+DR:FC=1:IFAF=3THEN46                    :rem 138
45 GOTO14                                       :rem 7
46 POKESB,27:FORT=1TO3000:NEXT:POKECL,240:PRINTCHR
   $(147):PRINT" YOU SAVED THE CASTLE"         :rem 12
47 PRINT:PRINT:PRINT" PLAY AGAIN?":GOTO52      :rem 121
48 POKES4,220:FORJ=15TO0STEP-.5:POKESB,127:POKEV,J
   :FORT=1TO10:NEXT                            :rem 232
49 POKESB,42:FORT=1TO10:NEXT:NEXT:POKES4,0:rem 232
50 FORT=1TO5000:NEXT                           :rem 239
51 POKECL,240:PRINTCHR$(147):PRINT"PLAY AGAIN? Y/N
   "                                           :rem 81
52 GETA$:IFA$=""THEN52                         :rem 241
53 IFA$="Y"THENRESTORE:GOTO1                    :rem 45
54 IFA$<>"N"THEN52                             :rem 255
55 END                                          :rem 65
56 DATA0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
   ,1,3,4,6,8,1,3,4,2,1,1,2,2,1,1,1,1,2        :rem 250
57 DATA1,2,1,1,1,2,2,4,2,4,2,1,2,5,2,2,1,1,2,1,3,2
   ,1,2,3,2,2,3,4,2,1,2,1,5,2,2,1,2,4,5        :rem 18
58 DATA1,1,1,1,1,1,2,3,2,1,1,1,1,2,1,1,1,1,7,3,6,2
   ,3,1,2,1,1,2,1,1,1,1,2,1,1,1,1,2,3,1        :rem 254
59 DATA2,6,2,2,4,2,1,2,1,2,3,3,4,2,7,1,2,1,1,1,1,1
   ,1,1,1,4,1,2,1,1,2,1,12,2,3,2,2,1,1,1       :rem 61
60 DATA2,1,2,2,1,2,2,3,2,2,1,2,2,3,3,2,1,1,1,1,1,2
   ,2,1,2,2,1,1,1,3,3,4,2,2,1,4,3,1,1,1        :rem 254
61 DATA1,2,2,1,1,4,1,2,2,9,6,2,1,2,1,1,1,1,1,1,1,1
   ,1,2,1,1,2,1,1,1,1,1,1,4,2,5,2,8,1,6        :rem 173
62 DATA3,4,3,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
   ,1,1,1,-1                                   :rem 252
63 DATA67,72,102,127,140,240,280,316,419       :rem 227
64 DATA23,24,26,27,28,45,46,48,49,50,82,83,84,105,
   106,123,124,125,145,146,147,162,163         :rem 154
65 DATA184,185,195,196,217,218,301,302,323,324,338
   ,339,360,361,441,442,443,445                :rem 85
66 DATA463,464,465,466,467                     :rem 81
67 B=INT(RND(1)*501)+0                         :rem 169
68 IFPEEK(B+L)<>PCTHEN67                       :rem 213
69 RETURN                                       :rem 79
70 POKES3,185:POKES2,202:POKES1,202:FORJ=15TO0STEP
   -.5:POKEV,J:NEXT:POKES3,210                 :rem 74
```

```
71 POKES1,232:POKES2,232:FORJ=15TO0STEP-.05:POKEV,
   J:NEXT                                        :rem 26
72 FORJ=0TO2:POKES1+J,0:NEXT:POKEV,15:RETURN
                                                 :rem 126
73 POKEM,32:POKEM+DR,37:POKES4,130:FORJ=15TO0STEP-
   1:POKEV,J:NEXT:POKES4,0:POKEM+DR,34           :rem 179
74 POKEM,58:POKEV,15:DR=0:FORT=1TO500:NEXT:RETURN
                                                 :rem 188
75 POKES3,220:FORT=1TO50:NEXT:POKES3,0:RETURN
                                                 :rem 170
76 FORJ=254TO180STEP-.5:POKES2,J:POKES3,J:POKEV,FV
   :FV=FV-.1:NEXT:POKES2,0:POKES3,0              :rem 23
77 POKECL,240:PRINTCHR$(147):PRINT" YOU FELL INTO
   {SPACE}A PIT":GOTO81                          :rem 18
78 FORG=190TO235:POKES4,G:FORT=1TO10:NEXT:NEXT:FOR
   G=235TO220STEP-1:POKES4,G:FORT=1TO20           :rem 72
79 NEXT:NEXT:FORJ=15TO5STEP-.1:POKEV,J:NEXT:POKES4
   ,0:POKEV,15:IFCS=1THEN24                      :rem 131
80 POKECL,240:PRINTCHR$(147):PRINT" YOU WERE KILLE
   D BY A{2 SPACES}BEAST!!"                      :rem 26
81 POKESB,27:FORT=1TO3000:NEXT:GOTO51            :rem 93
82 FORI=0TO3:FORJ=0TO15:POKEV,J:N=180+I*J:POKES3,N
   :NEXTJ,I:POKES3,0:RETURN                      :rem 63
```

Time Capsule

David Florance

There are almost as many ways to program text adventure games as there are games themselves. "Time Capsule," although a relatively simple adventure game, tests your logic and intuition as you try to escape a dangerous future. Best of all, the programmer has included detailed notes about how he created the game. For the VIC with 8K expansion.

Captured by an unknown enemy and transported into the future, you struggle to escape a labyrinth-like prison so you can return to your own time and sound the warning. That's the basic story line of "Time Capsule," a text adventure game for the VIC-20.

But there's more, for although dangers and rewards are built into the game, you create the plot through your character's actions. Do you try to bluff your way past the guards? How do you know what area of the prison is safe? Who will help you? Your decisions create the game. That's why playing text adventure games is so much fun. No two games seem to end up the same.

Typing In the World

Entering Time Capsule is easy when you use "The Automatic Proofreader," found in Appendix C. Before you begin typing in this adventure, make sure you read the appendix and have a copy of the Proofreader program on tape or disk. Using the Proofreader almost guarantees that you'll have a working copy of the game the first time you type it in. If you follow the instructions in Appendix C, it's almost impossible to make a typing error.

Once you've typed in and saved the game, LOAD and RUN it to play. You'll see a title screen, some simple instructions, and then your character appears in a room. The rest is up to you. You know how you would like the adventure to end—with you safely back in your own time so that you can warn the world about these dangerous enemies and their time capsule—but how you get to that ending is unknown. The game isn't difficult, but you do have to think things through. Use logic and common sense. Do you take the shoes or not? What would be the purpose? Try it and find out.

I've purposely left out a list of the adventure's vocabulary, since part of the fun is discovering how to communicate with the computer. The game doesn't require an esoteric vocabulary; just think a bit before you type something in, especially if your first request was ignored. Sometimes you'll miss the obvious.

Preplanning

When you're writing an adventure game, it's important that you have a clear idea of what your game will be about. Ask yourself some questions. Do you want characters in the game? Where does the adventure take place? What is the solution?

Once you've got a design that you like, you're ready to program. Don't be afraid to let the game develop as you write it, though. As you get deeper into the actual programming, you'll find certain things work better than you expected, while other things you thought essential are not practical. Be willing to make adjustments.

Setting up the program's variables is also important. If you want to take advantage of the VIC's graphics capabilities (if for nothing else than to liven up the display), this is the time to set variables for screen color, border color, and memory locations. I like to use variables that are easy to recognize and remember, such as CS for Color of Screen and CB for Color of Border. Logic plays the most important role in any programming, so use variables you're comfortable with and that make sense to you. You'll be glad you did later on.

The most economical place for your variables is at the beginning. There they will not interfere with the rest of the program. Another good reason for setting your variables at the top of your program is that all values will be set before execution of the main part of your program. Then, as you want to set parameters, you can call the variable instead of typing in the numbers each time. This is just one thing that makes BASIC so easy to use.

When writing text adventures, I've found it useful to have a structured line-numbering pattern. Leave yourself room to come back later and insert statements if you so desire. A good way to do this is to number in increments of 10 or 20, thus leaving space for anything you might later want to add.

The first statement should be a low number, say 10. After you've set your variables you may want to skip down to line

number 100 or so. This will remind you to keep your variable initialization at the beginning. If you make that statement number low, you will be less likely to put statements in front of it. Skipping down also provides a visual separation between your initialization procedures and the heart of the program.

IF-THEN?

There are many ways to program text adventures. Time Capsule is an example of a simple approach, one that doesn't use a lot of complicated formulas. In order to keep from being too technical, and to show how BASIC can be used to construct stimulating adventures, I didn't shy away from using the IF-THEN statement.

To me, it makes more sense to put most of the important IF-THEN statements together in a group. This is, of course, a personal preference. You may want to try other ways. The program incorporates two techniques that allow you to get around using a long list of IF-THENs. I've described them below, but you'll undoubtedly find others of your own.

One way is to use a FOR-NEXT loop to count the number of times you want the IF-THEN executed. This involves using variables to let the VIC know what it's counting and what to do on certain conditions. For instance, lines 92–120 in Time Capsule could have been translated into a list of IF-THEN statements that would look like this:

```
100 IFMID$(D$,3,1)=CHR$(32)THEND2$=MID$(D$,4,LEN(D
    $))
110 IFMID$(D$,4,1)=CHR$(32)THEND2$=MID$(D$,5,LEN(D
    $))
120 IFMID$(D$,5,1)=CHR$(32)THEND2$=MID$(D$,6,LEN(D
    $))
130 IFMID$(D$,6,1)=CHR$(32)THEND2$=MID$(D$,7,LEN(D
    $))
140 IFMID$(D$,7,1)=CHR$(32)THEND2$=MID$(D$,8,LEN(D
    $))
```

As you can see, the technique used in lines 92–120 is much shorter and does the same thing.

Second, use of the ON-GOTO can also replace a long list of IF-THEN statements. For instance, line 53 in Time Capsule replaced (from an earlier version of the game) this list of IF-THENs:

```
53 IF PZ=1 THEN 8010
54 IF PZ=2 THEN 8510
55 IF PZ=3 THEN 8710
56 IF PZ=4 THEN 8910
57 IF PZ=5 THEN 9020
58 IF PZ=6 THEN 9220
59 IF PZ=7 THEN 9420
60 IF PZ=7 THEN 9620
61 IF PZ=8 THEN 9840
```

Ultimately, this uses half the memory that a long list of IF-THEN statements uses.

In general, try to find ways to cut down on the number of IF-THEN statements, simply as a matter of economy. Don't be hesitant to use them if you cannot find a better way, because sometimes there *is* no better way, but do keep your options in mind.

Parsing

Perhaps the key to writing text adventures in BASIC is finding a way to have the computer read, or *parse*, the player's instructions. On the VIC-20, you can create an almost limitless vocabulary, and having the computer acknowledge a certain number of characters in the player's input is the method I've used here.

First, the program reads the entire player response. Then it separates the response into two parts: the first command or keyword and whatever else remains. Having the response divided into two parts allows the VIC to determine how it will act on that response.

There are 15 keywords in the game. If one of them is not used as the first word of the response, the computer tells the player that the response is unrecognized. In other words, unless the player begins the response with a keyword recognized by the program, the program will not "understand" it.

After the program *recognizes* a response as a keyword, it *identifies* the keyword. Then the program shifts to the routine where that particular keyword's actions are stored. Here the program considers the rest of the response, in order to properly respond to the player's instructions. If it doesn't recognize the rest of the instruction, the computer lets the player know. However, if the instruction is recognized, the program executes the entire command. The effect is much like "talking" to your VIC. It may seem almost as though the VIC is alive.

Determining how the program responds to certain words is entirely up to you, the programmer. However, I've found that players enjoy a text adventure more if the responses are simple. For example, long phrases such as "Walk two steps to the left and pick up the can" make for lots of typing. Try to design your program so it recognizes and acknowledges short phrases such as "Get can" or "East." This makes it easy for computer novices to play your game. It also makes it easier for you to test the program while it is being developed.

Most commercial text adventures allow the player to pick up and use items along the way. I've included this feature in Time Capsule. Although strings play an important role in this facet of programming text adventures, they are by no means the only way to handle it. READ and DATA statements could also be used. Since you'll want to alter the possible selection of items according to what the player has done, I think strings are as good a way as any; besides, they are not too complicated to use. For instance, Time Capsule defines A$ as the items used in the display you see in the first room.

All the items you'll see in the game are contained in strings. The rest of these items are defined in lines 5010–5045 of the program.

In Time Capsule, the player can pick up certain items. Each time that happens, the item is removed from the string. This is simply a matter of redefining the strings as selections are made. Be careful not to redefine a string before or after the player has made a choice, however.

```
5000 A$="WINDOW,SEAT,SHOES,CAN"
```

Playability

To keep your program readable, you may want to designate nomenclature for the different places the player explores. For instance, this game includes several puzzles that the players must solve, and I chose to separate the puzzles within the game. Such an approach seems to work well. Designations of PUZZLE 1, PUZZLE 2, and so on help remind you where the player is in the program. You'll find this helpful when editing the program as well. The VIC can then narrow down the actions it may take rather than review all its options before making a decision.

Say that a player types in a response that does not work

in the present situation but *does* work at some other point in the game. If the program knows which puzzle the player is on, it can determine not only the entire situation, but also the immediate situation. This will prevent the program from allowing a player to jump ahead, whether by intention or accident.

Changing screen colors or border colors for each puzzle also helps players complete the game. In addition, it enhances the program and keeps player interest alive.

Pacing is also important. It is vital that you design your game so it's not too slow. A player could easily get bored with puzzles that are too tedious or unusually difficult, so it's a good idea to throw in some easier ones periodically. On the other hand, a text adventure that can be solved in one sitting is not much fun either. Use your own judgment. Remember, *you* know the solution, the players don't.

Adventures of Your Own

If you're familiar with BASIC, you shouldn't have any trouble creating your own text adventure game. Even the simplest programming tools and techniques can form the heart of such a game; you don't have to know machine language or even how to program complicated graphics. These may be useful when you're writing an arcade-style game, but text adventures are different. Programming is almost secondary to the story you want to tell. Use your imagination. The rest is easy.

Time Capsule

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
10 C=36879:SC=4096:ES=4580:CC=37888:EC=38399:H=23:
   S=22:V=21:PZ=1                                  :rem 164
12 IF SK THEN POKEC,42:GOTO 16                     :rem 157
15 PRINT"{CLR}{YEL}":POKEC,42:GOSUB30000            :rem 83
16 GOTO 5000                                       :rem 101
17 IFSKTHEN30                                       :rem 39
19 POKEC,42                                        :rem 110
23 IF AC=20THEN9610                                 :rem 22
25 IF AC=23THEN60020                                :rem 67
26 FORT=15 TO 0 STEP-1:POKE36878,T:FORY=1 TO500:NE
   XT:NEXT:POKE 36877,0                            :rem 146
27 INPUT"{CLR}{4 DOWN}{RIGHT}INSTRUCTIONS(Y/N)";IS
   $                                                :rem 8
```

```
28 IFIS$="Y"THENGOSUB50000                          :rem 55
30 SK=0                                             :rem 110
32 ONPZGOTO 9000,9200,9400,9600,9800,10000,10200,1
   0400,10600,10800                                 :rem 59
50 PRINT:PRINT"{YEL}YOU SEE:"                       :rem 175
53 ONPZGOTO8010,8510,8710,8910,9020,9220,9420,9620
   ,9840                                            :rem 142
75 PRINT:PRINT:INPUT"{CYN}ACTION{WHT}";D$:PRINT"
   {WHT}"                                           :rem 227
78 IFAC=21THEN15024                                 :rem 77
79 IFAC=22THEN15028                                 :rem 83
80 D1$=LEFT$(D$,3)                                  :rem 165
90 G=3:G1=4                                         :rem 67
92 FORTY=1TO5                                       :rem 69
100 IFMID$(D$,G,1)=CHR$(32)THEND2$=MID$(D$,G1,LEN(
    D$))                                            :rem 192
110 G=G+1:G1=G1+1                                   :rem 124
120 NEXTTY                                          :rem 127
1000 REM D1$=KEYWORD:TEST FOR KEYWORD               :rem 38
1030 IFD1$=MID$(B$,47,1)THEN3050                    :rem 201
1040 IFD1$=MID$(B$,45,1)THEN3070                    :rem 202
1050 IFD1$=MID$(B$,43,1)THEN3090                    :rem 203
1060 IFD1$=MID$(B$,41,1)THEN3110                    :rem 195
1070 IFD1$=MID$(B$,37,3)THEN3130                    :rem 205
1080 IFD1$=MID$(B$,17,3)THEN3150                    :rem 206
1090 IFD1$=MID$(B$,13,3)THEN3170                    :rem 205
2000 IFD1$=MID$(B$,9,3)THEN3190                     :rem 156
2010 IFD1$=MID$(B$,5,3)THEN3210                     :rem 146
2020 IFD1$=MID$(B$,1,3)THEN3230                     :rem 145
2030 IFD1$=MID$(B$,21,3)THEN3250                    :rem 198
2040 IFD1$=MID$(B$,25,3)THEN3270                    :rem 205
2050 IFD1$=MID$(B$,29,3)THEN3290                    :rem 212
2060 IFD1$=MID$(B$,33,3)THEN3310                    :rem 201
2065 IFD1$=MID$(B$,49,3)THEN3320                    :rem 214
2070 IFD1$=MID$(B$,53,3)ORD1$=MID$(B$,57,3)THEN335
     0                                              :rem 243
2080 IFD1$=MID$(B1$,1,3)THEN3050                    :rem 200
2090 IFD1$=MID$(B1$,6,3)THEN3070                    :rem 208
2095 IFD1$=MID$(B1$,17,3)THEN3110                     :rem 2
3000 IFD1$=MID$(B1$,11,3)THEN3090                   :rem 246
3010 IFD1$=MID$(B1$,17,3)THEN3110                   :rem 246
3040 GOTO8500                                       :rem 205
3050 REM KEYWORD E(AST)                              :rem 79
3052 ONPZGOTO3053,3057,8700,8700,8700,8700,8700,87
     00,3060,8700                                   :rem 226
3053 PZ=2:GOTO32                                    :rem 188
3057 PZ=3:GOTO32                                    :rem 193
3060 PZ=8:GOTO32                                    :rem 192
3070 REM KEYWORD W(EST)                             :rem 103
```

```
3076 ONPZGOTO8700,3078,3078,8700,8700,8700,8700,30
     80,10800,8700                              :rem 30
3078 POKEC,42:PZ=PZ-1:GOTO32                   :rem 215
3080 PZ=9:GOTO32                               :rem 195
3090 REM KEYWORD S(OUTH)                       :rem 185
3092 ONPZGOTO8700,8700,3097,8700,8700,8700,3099,87
     00,8700,8700                              :rem 250
3097 PZ=4:GOTO32                               :rem 198
3099 PZ=8:GOTO32                               :rem 204
3110 REM KEYWORD N(ORTH)                       :rem 170
3115 ONPZGOTO8700,8700,8700,3122,8700,8700,8700,31
     19,8700,8700                              :rem 228
3119 POKEC,110:PZ=7:GOTO32                      :rem 46
3122 POKEC,42:PZ=PZ-1:GOTO32                   :rem 205
3130 REM KEYWORD SI(T)                          :rem 17
3132 GOSUB 38000                                :rem 68
3135 PRINT"WHY DO YOU WANT TO DO THAT? REMEMBER --
     -GET BACK TO YOUR OWN TIME!"                :rem 2
3140 FORTY=1TO3000:NEXT:GOTO32                 :rem 129
3150 REM KEYWORD RUN                           :rem 199
3152 GOSUB 38000                                :rem 70
3155 PRINT"DON'T BE SCARED--FEAR BREEDS FEAR.":FOR
     TY=1TO5000:NEXT:GOTO32                    :rem 162
3170 REM KEYWORD SHO(W)                        :rem 102
3172 IFD2$="ID"ORD2$="PAPERS"ANDHJTHEN10050:rem 21
3175 GOTO8700                                  :rem 216
3190 REM KEYWORD INV                           :rem 195
3192 GOSUB 35000                                :rem 71
3194 PRINT"YOU HAVE:"                          :rem 253
3196 ONIVGOTO3200,3205,3198,3206,3207,3208,3199,32
     01                                        :rem 224
3198 PRINTI$:PRINTE1$:FORCE=10TO2000:NEXT:GOTO32
                                               :rem 45
3199 PRINTC1$:PRINTE1$:FORCE=10TO2000:NEXT:GOTO32
                                               :rem 89
3200 PRINTI$:FORCE=1TO2000:NEXT:GOTO32         :rem 140
3201 PRINTI$:PRINTE1$:PRINTC1$:PRINTC2$:FORTY=1TO3
     000:NEXT:GOTO32                           :rem 211
3205 PRINTE1$:FORCE=1TO2000:NEXT:GOTO32        :rem 190
3206 PRINTC1$:FORCE=1TO2000:NEXT:GOTO32        :rem 189
3207 PRINTC1$:PRINTI$:FORCE=10TO2000:NEXT:GOTO32
                                               :rem 34
3208 PRINTI$:PRINTE1$:PRINTC1$:FORCE=10TO2000:NEXT
     :GOTO32                                   :rem 132
3209 FORCE=1TO2000:NEXT:GOTO32                  :rem 97
3210 REM KEYWORD DRO(P)                         :rem 85
3215 I$="":E1$="":GOSUB7000:GOTO32             :rem 103
3230 REM KEYWORD GET                           :rem 177
```

```
3231 IFLO=Ø ANDD2$=MID$(A$,13,5)THEN GOTO3239
                                          :rem 232
3232 IFLO=111ANDD2$=MID$(A$,13,5)THEN3235  :rem 15
3233 IFPZ=3ANDD2$=MID$(E$,4,6)THEN3240     :rem 144
3235 IFD2$="CAN"THEN3237                    :rem 71
3236 GOTO8ØØØ                             :rem 2Ø7
3237 GOSUB38ØØØ:PRINT"YOU DRANK IT----------WHATEV
     ER IT WAS TASTEDHORRIBLE!"             :rem 76
3238 FORTY=1TO3ØØØ:NEXT:GOTO32             :rem 137
3239 I$=MID$(A$,13,5):A$="WINDOW,SEAT,CAN":LO=111:
     GOSUB7ØØØ:IV=1:GOTO32                 :rem 18Ø
3240 E1$=MID$(E$,4,6):E$="":GOSUB7ØØØ:GOTO32
                                          :rem 151
3250 REM KEYWORD TAK(E)                     :rem 73
3252 IFD2$<>"POD"THEN88ØØ                  :rem 149
3253 IFPZ=4THENINPUT"{CLR}{5 DOWN}{4 RIGHT}LEVEL(1
     -4)";LL:ONLLGOTO15ØØØ,15Ø1Ø,15Ø2Ø,15Ø3Ø
                                           :rem 14
3255 GOTO89ØØ                              :rem 217
3267 GOSUB7ØØØ:GOTO15                      :rem 242
3270 REM KEYWORD TAL(K)                     :rem 82
3271 IFPZ<>9THEN 84ØØ                      :rem 18Ø
3272 IFD2$="TO MAN"ORD2$="MAN"THEN 3274    :rem 142
3273 GOTO84ØØ                              :rem 212
3274 PRINT"{BLK}THE MAN SAYS:'YOU HAVEBEEN VERY SM
     ART TO{4 SPACES}MAKE";                :rem 147
3275 PRINT" IT THIS FAR. TAKETHIS IGNITION PLUG. I
     TWILL START THE GREEN ";              :rem 146
3276 PRINT" VEHICLE{2 SPACES}WITH THE BLUETOP. GOO
     D LUCK.":IV=8:C2$="PLUG"              :rem 1Ø5
3280 GOTO75                                :rem 114
3285 GOSUB7ØØØ:GOTO15                      :rem 242
3290 REM KEYWORD USE                       :rem 196
3292 IFD2$="COMPUTERS"ORD2$="COMPUTER"THENPRINT"YE
     S, BUT HOW?":GOTO5Ø                   :rem 116
3294 IFD2$="CARD"ANDC1$<>""THEN985Ø        :rem 196
3295 IFD2$="POD"THEN3252                    :rem 91
3298 GOTO89ØØ                              :rem 224
3310 REM KEYWORD LOO(K)                     :rem 86
3312 L2$="OUT ":L1$=MID$(A$,1,6):IFL2$+L1$=D2$THEN
     3315                                  :rem 12Ø
3314 GOTO85ØØ                              :rem 2Ø9
3315 PRINT"YOU SEE A VEHICLE{5 SPACES}OUTSIDE":GOS
     UB7ØØØ:GOTO3Ø                          :rem 44
3320 REM KEYWORD ENT(ER)                   :rem 16Ø
3322 ONPZGOTO87ØØ,87ØØ,87ØØ,3325,87ØØ,3327,87ØØ,87
     ØØ,87ØØ,87ØØ                          :rem 234
3325 IFD2$="POD"THEN3253                    :rem 86
```

```
3327 IFD2$="DOOR"ANDLL=2THEN GOSUB38000:PRINT"YOU
     {SPACE}CANNOT WITHOUT{4 SPACES}VERIFICATION"
                                          :rem 179
3330 FORTY=1TO3000:NEXT:GOTO32            :rem 130
3350 REM KEYWORD HIT OR KIL(L)            :rem 215
3355 PRINT"{CLR}{4 DOWN}YOUR ATTEMPT AT{7 SPACES}V
     IOLENCE WAS UNWISE."                 :rem 196
3357 PRINT"{4 DOWN}YOU HAVE BEEN{9 SPACES}EXTERMIN
     ATED."                               :rem 218
3359 FORTY=1TO5000:NEXT:GOTO60000          :rem 32
5000 A$="WINDOW,SEAT,SHOES,CAN"           :rem 136
5010 B$="GET,DRO,INV,SHO,RUN,TAK,TAL,USE,LOO,SIT,N
     ,S,W,E,ENT,HIT,KIL"                  :rem 111
5015 B1$="EAST,WEST,SOUTH,NORTH"          :rem 245
5020 C$="ROOMS EAST & WEST":E$="ID PAPERS":F$="ELE
     VATION POD"                           :rem 0
5030 G$="COMPUTERS ALL AROUND":H$="GUARDS & DOOR":
     M1$="PEOPLE EVERYWHERE"              :rem 135
5040 M2$="{BLU}PEOPLE COMING FROM ANDGOING TO ROOM
     S AHEAD."                            :rem 189
5045 M3$="BLACK SUITED MAN"               :rem 250
5100 GOTO17                               :rem 103
7000 PRINT"OK":GOSUB 37000                :rem 230
7500 FORCE=1TO3000:NEXT:RETURN            :rem 210
8000 GOSUB 38000                           :rem 67
8005 PRINT"YOU CAN'T GET THAT":FORCE=1TO2000:NEXT
     :GOTO32                              :rem 246
8010 PRINTA$:GOTO75                       :rem 154
8400 PRINT"TALK IS CHEAP.":FORCE=1TO2000:NEXT:GOT
     O32                                  :rem 241
8500 GOSUB 38000                           :rem 72
8505 PRINT"DOES NOT COMPUTE":FORCE=1TO2000:NEXT:G
     OTO32                                :rem 217
8510 PRINTC$:GOTO75                       :rem 161
8700 GOSUB 38000                           :rem 74
8705 PRINT"YOU CAN'T":FORCE=1TO3000:NEXT:GOTO32
                                          :rem 189
8710 PRINTE$:GOTO75                       :rem 165
8800 GOSUB 38000                           :rem 75
8805 PRINT"TRY'GET'{14 SPACES}(NOT GUARANTEED TO
     {4 SPACES}WORK BUT IT SOUNDS{4 SPACES}BETTER)
     "                                     :rem 5
8810 FORTY=1TO3000:NEXT:GOTO32            :rem 138
8900 GOSUB 38000                           :rem 76
8905 PRINT"NO CAN DO":FORDI=1TO4000:NEXT:GOTO32
                                          :rem 125
8910 PRINTF$:GOTO75                       :rem 168
9000 GOSUB 35000:REM PUZZLE 1             :rem 122
9005 POKEC,42                             :rem 210
```

```
9010 PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN DIMLY LIT
     {2 SPACES}ROOM.":GOTO50                    :rem 247
9020 PRINTG$:GOTO75                             :rem 162
9200 GOSUB 35000:REM PUZZLE 2                   :rem 125
9210 POKEC,76:PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN A
      HALLWAY.":PZ=2:IFIV=0THEN9215              :rem 60
9213 GOTO50                                     :rem 109
9215 PRINT"YOU LOOKED SUSPICIOUS WITH NO SHOES ON.
     ":FORZ=1TO1000:NEXT:GOTO15007              :rem 208
9220 PRINTH$:GOTO75                             :rem 165
9400 GOSUB 35000:REM PUZZLE 3                   :rem 128
9405 POKEC,42                                   :rem 214
9410 PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN A LARGE
     {4 SPACES}ROOM WITH A LARGE OVALTABLE IN THE
     {SPACE}CENTER.                              :rem 0
9415 PZ=3:IV=2:IFI$THENIV=3:GOTO 50             :rem 129
9420 PRINTM1$:GOTO75                            :rem 221
9600 GOSUB 35000:REM PUZZLE 4                   :rem 131
9610 POKEC,110:PRINT"{CLR}{2 DOWN}{BLK}YOU ARE IN
     {SPACE}A CORRIDOR."                        :rem 143
9615 PZ=4:GOTO50                                :rem 200
9620 PRINTM2$:GOTO75                            :rem 224
9800 GOSUB 35000:REM PUZZLE 5                   :rem 134
9805 POKEC,110                                   :rem 6
9810 PRINT"{CLR}YOU ARE ON THE FOURTH FLOOR. THE W
     ALLS OF{3 SPACES}THE HUGE CHAMBER ARE ";
                                                :rem 25
9820 PRINT" LINED WITH HUNDREDS OFCOMPUTERS."
                                                :rem 46
9830 PZ=5:AC=0:GOTO 50                          :rem 243
9840 PRINTM3$:GOTO75                            :rem 229
9850 NR=INT(RND(1)*100)+1                       :rem 108
9860 PRINT"{CLR}{BLK}":POKEC,93                  :rem 15
9865 INPUT"{4 DOWN}{8 RIGHT}TERMINAL #";BN :rem 38
9870 IFBN<NRTHENPRINT"TRY TERMINAL RIGHT":GOTO9865
                                                :rem 251
9880 IFBN>NRTHENPRINT"TRY TERMINAL LEFT":GOTO9865
                                                :rem 171
9890 IFBN=NRTHENPRINT"{RVS}CORRECT TERMINAL{OFF}"
                                                :rem 232
9895 FORTY=1TO3000:NEXT:GOSUB40000              :rem 110
9900 PRINT"{CLR}AS3E43 382F6 9JDH8GEU LOW978H H14Q
     SN.":INPUT JK$                              :rem 28
9910 IF JK$="Q"THEN60000                         :rem 18
9920 IF JK$="TRANSLATE"THEN9940                  :rem 80
9925 IFJK$="LOG ON"THEN9940                      :rem 38
9930 GOTO 9900                                  :rem 224
9940 PRINT"{CLR}{RVS}SYSTEM ACCESS COMMAND{OFF}":I
     NPUTKJ$:IFKJ$="Q" THEN 60000               :rem 112
```

```
9950 IFKJ$="INFORMATION"THEN 9975          :rem 243
9960 GOTO 9940                             :rem 231
9975 PRINT"{CLR}{CYN}{4 DOWN}YOU SHOULD LOOK FOR A
      MAN WITH A BLACK SUIT ON."           :rem 20
9976 PRINT"YOU ARE TO GO TO THE{2 SPACES}FIRST LEV
     EL NOW. THE{2 SPACES}POD IS RETURNING....."
                                           :rem 150
9980 FORTY=1TO6000:NEXT:GOTO9600             :rem 0
10000 GOSUB 35000:REM PUZZLE 6             :rem 167
10010 PRINT"{CLR}{RED}YOU ARE ON THE SECOND LEVEL.
       THERE ARE ARMEDGUARDS STANDING IN ";
                                           :rem 208
10020 PRINT"{3 SPACES}FRONT OF THE ENTRANCE TO ANO
      THER ROOM."                          :rem 50
10030 PZ=6:AC=0:HJ=34:GOTO50               :rem 132
10050 GOSUB7000:PRINT"YOU ARE INSIDE THE{4 RIGHT}C
      OMPUTER ACCESS BANK. YOU ARE GIVEN ";:rem 34
10060 PRINT"A CARD. YOU LEAVE. THE POD IS RETURNIN
      G......."                            :rem 73
10070 FORTY=1TO6000:NEXT:C1=1:C1$="CARD":PZ=6
                                           :rem 188
10080 IV=4:IFI$THENIV=5                    :rem 127
10085 IFE1$THENIV=7                        :rem 105
10090 IFI$<>""ANDE1$<>""THENIV=6             :rem 32
10100 GOTO9600                             :rem 250
10200 REM PUZZLE 7                         :rem 248
10210 GOTO15000                              :rem 35
10400 GOSUB 35000:REM PUZZLE 8             :rem 173
10405 POKEC,142:FL=0                       :rem 104
10410 PRINT"{CLR}{BLU}THE MAN HAS{11 SPACES}DISAPP
      EARED. YOU HEAR MEN{SHIFT-SPACE}TALKING IN A
      {6 SPACES}";                         :rem 28
10420 PRINT"FOREIGN LANGUAGE. YOU ALSO HEAR ENGINE
      S{5 SPACES}RUNNING TO THE WEST."     :rem 175
10430 PZ=8:GOTO50                          :rem 239
10600 GOSUB35000:REM PUZZLE 9              :rem 176
10610 PRINT"{CLR}{CYN}YOU ARE IN A GARAGE
      {3 SPACES}THAT LEADS TO THE{5 SPACES}FRONT O
      F THE BUILDING."                     :rem 232
10620 PZ=9:GOTO50                          :rem 241
10800 REM PUZZLE 10                          :rem 40
10810 POKEC,76                                :rem 5
10820 GOSUB 35000                          :rem 115
10825 PRINT"{CLR}YOU ARE OUTSIDE THE{3 SPACES}BUIL
      DING. THERE ARE{3 SPACES}SEVERAL VEHICLES TH
      ERE";                                :rem 27
10830 INPUT"NOW. WHICH IS THE{5 SPACES}VEHICLE YOU
       WANT";VH$                           :rem 22
```

```
10840 IF VH$<>"GREEN WITH BLUE TOP" AND VH$<>"BLUE
      AND GREEN" THEN 10890                    :rem 196
10841 POKEC,237:Q=1                            :rem 49
10842 FORBV=1TO50                              :rem 244
10843 IFQTHENPOKEC,94:PRINT"{BLU}":Q=0:GOTO10850
                                               :rem 173
10845 POKEC,237:PRINT"{BLK}":Q=1               :rem 208
10850 POKE 36878,15:FORT=200TO252:POKE36876,T
                                               :rem 70
10855 PRINT"{CLR}YOU CHOSE THE CORRECT VEHICLE. YO
      U ARE{2 SPACES}NOW ON YOUR WAY BACK TO ";
                                               :rem 140
10860 PRINT"{2 SPACES}YOUR OWN TIME{2 SPACES}PERIO
      D.YOU CAN SAVE THE WORLDFROM DISASTER!!!"
                                               :rem 144
10865 NEXT:NEXT                                :rem 188
10870 GETAG$:IFAG$=""THEN 10870                :rem 177
10880 GOTO60000                                :rem 48
10890 PRINT"{CLR}":POKEC,127:PRINT"{4 DOWN}YOU HAV
      E BEEN ARRESTED AND FORCED ";            :rem 125
10900 PRINT"TO{4 SPACES}TAKE A KNOCKOUT PILL."
                                               :rem 161
10910 FORTY=1TO3000:NEXT:PRINT"{4 DOWN}YOU WILL WA
      KE SHORTLY.":POKEC,8                     :rem 251
10920 FORTY=1TO10000:NEXT:CLR:SK=30:GOTO10:rem 114
15000 GOSUB 35000:REMLL=1                      :rem 146
15002 PRINT"{CLR}YOU ARE ON THE FIRST{2 SPACES}LEV
      EL. THE MAN YOU ARELOOKING FOR IS "; :rem 64
15004 PRINT"WALKINGSOUTH.":FORTY=1TO5000:NEXT:PZ=7
      :IV=IV:IFFLTHEN50 IS ";                  :rem 171
15005 PRINT"{3 DOWN}THE AUTHORITIES COME{2 SPACES}
      UP AND QUESTION HIM.{2 SPACES}THEY NOW LOOK
      {SPACE}AT YOU."                          :rem 116
15006 PRINT"{RVS}OH NO!{OFF}"                  :rem 198
15007 FORTY=1TO5000:NEXT:PRINT"YOU HAVE BEEN ARRES
      TEDAND REIMPRISONED."                    :rem 18
15008 FORTY=1TO5000:NEXT:CLR:SK=30:GOTO10      :rem 72
15010 REMLL=2                                  :rem 226
15015 GOTO 10000                               :rem 38
15020 GOSUB 35000:REM LL=3                     :rem 150
15021 PRINT"{3 DOWN}THE THIRD LEVEL IS THEINTERROG
      ATION CENTER."                           :rem 34
15022 PRINT"YOU HAVE NOW BEEN{5 SPACES}BRAINWASHED
      .":AC=21:GOTO75                          :rem 141
15024 PRINT"YOU HAVE TRIED TO{5 SPACES}ESCAPE AND
      {SPACE}HAVE BEEN{2 SPACES}CAUGHT."  :rem 181
15026 PRINT"YOU CANNOT RESPOND TO QUESTIONS ABOUT
      {SPACE}YOUR{2 SPACES}LIFE AND ARE ABOUT ";
                                               :rem 99
```

```
15027 PRINT"TO BE EXTERMINATED.":AC=22:GOTO75
                                              :rem 240
15028 PRINT"YOUR EXPLANATION WAS{2 SPACES}INSUFFIC
      IENT. YOU HAVEBEEN EXTERMINATED."       :rem 246
15029 FORUU=1TO5000:NEXT:GOTO60000             :rem 74
15030 DV=4:REM LL=4                            :rem 43
15040 GOTO9800                                  :rem 4
30000 POKE 36878,15                           :rem 196
30010 FORD=0TO 255:POKE36877,D:NEXT            :rem 44
30030 FORM=0 TO 255                           :rem 214
30040 POKE 36875,M:NEXT                        :rem 37
30045 FORY=1TO205STEP5:PRINT"{3 DOWN}{19 RIGHT}
      {RVS}TIME CAPSULE{OFF}"1984+Y;           :rem 47
30047 NEXT:FORTY=1TO1000:NEXT                  :rem 86
30060 FORR=255 TO 0STEP-1:POKE 36875,R        :rem 108
30070 NEXT                                     :rem 57
30080 RETURN                                  :rem 219
35000 POKE36878,0:POKE36878,15:FORT=1TO20:POKE3687
      6,241:POKE36876,230:FORY=1TO50:NEXT      :rem 97
35005 NEXT:POKE36878,0                         :rem 17
35010 POKE 36876,0:RETURN                     :rem 172
37000 POKE 36878,15:FORH=145TO255:POKE 36876,H
                                              :rem 53
37010 NEXT:POKE36876,0:POKE 36878,0           :rem 226
37020 RETURN                                  :rem 220
38000 POKE 36878,15:POKE 36874,213:FORY=1TO900:NEX
      T:POKE 36874,195                        :rem 224
38010 FORT=1TO1200:NEXT:POKE36874,0:POKE36878,0:RE
      TURN                                     :rem 68
39000 FORJ=1 TO 15:POKE 36878,J:POKE36877,200:FORA
      =1TO100:NEXT                             :rem 70
39005 NEXT:FORI=1TO100                         :rem 76
39010 NEXT:FORJ=15 TO 1STEP-1:POKE 36878,J:POKE368
      77,200:FORE=1TO100:NEXT                  :rem 94
39020 NEXT:POKE 36877,0:POKE36878,0:RETURN      :rem 0
40000 REM COMPUTER                             :rem 71
40010 FL=30:PRINT"{CLR}{WHT}":FORT=CCTOEC:POKET,1:
      NEXT                                     :rem 192
40020 POKEC,25:FORTY=1TO1000:NEXT:POKEC,8:FORTY=1T
      O10000:NEXT                              :rem 109
40030 POKEC,110:FORK=SCTOSC+V:POKEK,102:NEXT:FORM=
      ESTOES+V:POKEM,102:NEXT                  :rem 249
40040 FORD=SC+STOESSTEPS:POKED,102:NEXT         :rem 35
40045 FORE=SC+VTOES+VSTEPS:POKEE,102:NEXT      :rem 174
40050 FORZA=SC+115TOSC+203STEPS:POKEZA,78:NEXT:FOR
      ZA=SC+114TOSC+203STEPS:POKEZA,223        :rem 79
40052 NEXT                                     :rem 58
40055 FORUM=SC+117TOSC+205STEPS:POKEUM,118:NEXT:FO
      RUM=SC+118TOSC+206STEPS:POKEUM,233       :rem 167
```

```
40057 NEXT                                          :rem 63
40060 FORGE=SC+125TOSC+213STEPS:POKEGE,78:NEXT:FOR
      GE=SC+124TOSC+212STEPS                        :rem 95
40065 POKEGE,223:NEXT                              :rem 246
40070 FORGE=SC+126TOSC+214STEPS:POKEGE,118:NEXT:FO
      RGE=SC+127TOSC+215STEPS:POKEGE,127            :rem 78
40080 NEXT                                          :rem 59
40090 FORZK=SC+377TOSC+392:RC=INT(RND(1)*26)+1:POK
      EZK,RC:NEXT                                   :rem 14
40150 GETQW$:IFQW$=""THEN40050                     :rem 228
40200 RETURN                                       :rem 214
50000 GOSUB39000:POKE C,11:PRINT"{CLR}{CYN}"
                                                   :rem 232
50010 PRINT"{DOWN}{4 RIGHT}{RVS}TIME CAPSULE"
                                                   :rem 154
50020 PRINT"{2 DOWN}{RIGHT}DURING A SEARCH IN ";
                                                   :rem 153
50022 PRINTSPC(22)"{3 SPACES}OUTER SPACE FOR";SPC(
      22)"{7 SPACES}MISSING ";                     :rem 182
50030 PRINT"ASTRONAUTS";SPC(26)"YOU HAVE BEEN";SPC
      (31)"CAUGHT BY THE ";                        :rem 195
50035 PRINT"ENEMY.";SPC(24)                         :rem 82
50040 PRINT"THEY PLACE YOU IN ";SPC(26)            :rem 215
50042 PRINT"THEIR SECRET WEAPON: ";                 :rem 77
50045 PRINTSPC(23)"THE BRAND NEW ";                :rem 216
50050 PRINTSPC(30)"{RVS}TIME CAPSULE{OFF}."
                                                   :rem 115
50052 PRINTSPC(27)"PRESS {RVS}RETURN{OFF}":rem 126
50055 GET R$:IF R$="" THEN 50055                    :rem 67
50058 IFR$<>CHR$(13)THEN50000                       :rem 91
50060 GOSUB39000:POKEC,56:PRINT"{CLR}{BLK}{DOWN}
      {4 RIGHT}{RVS}TIME CAPSULE"                  :rem 187
50062 PRINTTAB(45)"YOU HAVE REASONED ";SPC(26)
                                                   :rem 171
50064 PRINT"THAT THIS IS WHERE ";SPC(25)            :rem 41
50066 PRINT"ALL THE OTHER ";SPC(30)                :rem 227
50070 PRINT"ASTRONAUTS HAVE BEEN ";SPC(23):rem 246
50072 PRINT"DISAPPEARING TO. ";SPC(74)             :rem 244
50075 PRINT"PRESS {RVS}RETURN{OFF}"                :rem 227
50077 GET R$:IF R$=""THEN 50077                     :rem 75
50079 IFR$<>CHR$(13)THEN50000                       :rem 94
50080 GOSUB39000:POKE C,11:PRINT"{CLR}{CYN}"
                                                   :rem 240
50081 PRINT"{DOWN}{4 RIGHT}{RVS}TIME CAPSULE"
                                                   :rem 162
50082 PRINTTAB(45)"YOU ARRIVE IN THE ";SPC(26)
                                                   :rem 121
50084 PRINT"YEAR {RVS}2185{OFF}, WITH ";SPC(26)
                                                   :rem 185
```

```
50090 PRINT"{2 RIGHT}THE SAME AGE AND ";SPC(27)
                                          :rem 139
50092 PRINT"PERSONALITY, AND ";SPC(27)     :rem 5
50094 PRINT"YOU KNOW ESCAPE WILL ";SPC(23):rem 207
50100 PRINT"BE DIFFICULT.";SPC(79)        :rem 248
50105 PRINT"PRESS {RVS}RETURN{OFF}"       :rem 221
50107 GETR$:IF R$="" THEN 50107            :rem 63
50109 IFR$<>CHR$(13)THEN50000              :rem 88
50110 GOSUB39000:POKEC,56:PRINT"{CLR}{BLK}{DOWN}
      {4 RIGHT}{RVS}TIME CAPSULE"         :rem 183
50115 PRINTTAB(45)"YOUR MISSION: ";SPC(30):rem 222
50117 PRINT"RETURN TO YOUR OWN ";SPC(25)  :rem 110
50120 PRINT"TIME TO WARN YOUR ";SPC(26)   :rem 252
50122 PRINT"LEADERS OF THE ";SPC(29)       :rem 30
50124 PRINT"ENEMY'S ACTIONS. ";SPC(26)    :rem 222
50130 PRINTTAB(70)"PRESS {RVS}RETURN{OFF}":rem 106
50140 GETR$:IFR$=""THEN50140               :rem 57
50150 IFR$<>CHR$(13)THEN50000              :rem 84
50160 GOSUB 39000:POKEC,11:PRINT"{CLR}{CYN}"
                                          :rem 239
50170 PRINT"{DOWN}{4 RIGHT}{RVS}TIME CAPSULE"
                                          :rem 161
50180 PRINTTAB(45)"YOU MAY USE ";SPC(32)    :rem 8
50182 PRINT"SHORT ENGLISH PHRASES ";SPC(23):rem 88
50184 PRINT"SUCH AS 'GET CAN' ";SPC(26)   :rem 116
50190 PRINT"AND TO MOVE FROM ";SPC(27)    :rem 140
50192 PRINT"PLACE TO PLACE DON'T ";SPC(23):rem 114
50200 PRINT"USE 'GO':SIMPLY ";SPC(28)     :rem 141
50202 PRINT"TYPE THE DIRECTION ";SPC(25)  :rem 103
50204 PRINT"IN WHICH YOU WISH ";SPC(26)   :rem 232
50206 PRINT"TO MOVE.";SPC(39)             :rem 180
50207 PRINT"PRESS {RVS}RETURN{OFF}"       :rem 224
50208 GETR$:IF R$="" THEN 50208            :rem 67
50209 IFR$<>CHR$(13)THEN50000              :rem 89
50210 GOSUB 39000:POKEC,56:PRINT"{CLR}{BLK}{DOWN}
      {4 RIGHT}{RVS}TIME CAPSULE"         :rem 184
50212 PRINTTAB(45)"FOR EXAMPLE, ";SPC(31)  :rem 81
50214 PRINT"TYPE 'EAST' OR 'E' ";SPC(25)  :rem 151
50215 PRINT"TO GO EAST. THE ";SPC(28)      :rem 31
50220 PRINT"COMPUTER WILL TELL ";SPC(25)  :rem 123
50222 PRINT"YOU IF A WORD YOU ";SPC(26)   :rem 172
50224 PRINT"HAVE USED IS ";SPC(31)        :rem 149
50230 PRINT"ILLEGAL. PLAY SMART. ";SPC(23):rem 181
50232 PRINT"THE ANSWERS ARE ";SPC(28)     :rem 133
50234 PRINT"EASY!";SPC(43)                :rem 251
50240 PRINT"PRESS {RVS}RETURN{OFF}"       :rem 221
50250 GET R$:IF R$=""THEN 50250            :rem 61
50260 IF R$<>CHR$(13)THEN50000             :rem 86
50270 RETURN                              :rem 222
```

```
59999 END                                          :rem 240
60000 PRINT"{CLR}":POKEC,25:INPUT"{BLK}{6 DOWN}
      {4 RIGHT}PLAY AGAIN";FF$                     :rem 146
60005 IFFF$="Y"THENRUN                             :rem 56
60007 PRINT"{CLR}":POKEC,59:DE=1001:FORX=0TODE:PRI
      NTCHR$(191);:DE=DE+1:NEXT                    :rem 200
```

Sigma Mission

George Miller

"Sigma Mission" was written for the unexpanded VIC-20 and demonstrates an unusual alternative to the arrays most text adventures use for data storage. Although the game uses less than 4K of memory, you'll be surprised at the challenge that it offers.

The year is 2084, and you are enroute to Alpha Centauri. Your home is an asteroid-turned-spaceship which contains living quarters for you and the other colonists. The voyage will take several lifetimes, and though you won't live to see the arrival, it's up to you to keep things running properly for your descendants.

Normally, the life support systems are controlled by the master computer, but now something has gone wrong with its program. You know you can fix it, but first you must get to the central terminal—and that's not easy. The master computer has locked the doors between you and the terminal room; it has also sent robots to dig deep pits as traps in some rooms.

As you explore the chambers, you'll find a variety of items. Some are worthless; others are vital to the successful completion of your mission.

Watch out for the roving alien Zapper, too; it enjoys using its transporter beam. The Zapper is looking for something (a box, perhaps?) and thinks you may have it. It gets upset if you don't. After any encounter with the Zapper, it would be wise to check your supplies.

Anticipating your attack, the computer has locked every door and activated the built-in defense systems. It will unlock doors if you give it the proper command, but it will also defend itself when necessary. For instance, if you try to access the master computer but are not successful, it's likely to take offense and transport you out of the master computer control room.

It's going to be dangerous, but that program has got to be fixed. And you are the only one who can do it.

A Lot of Game in a Little Memory

Adventure games generally require much data (and a great deal of memory) to create the game environment and define the game. As a result, most of the adventures that you see are for computers with relatively large memories. It is thus a challenge to create a quality adventure game for the unexpanded VIC, but with Sigma Mission, that goal has been realized.

Although Sigma Mission will run on most memory configurations, its design is an example of what can be done in a limited amount of memory by minimizing the use of variables. Variables are reused whenever possible in order to conserve memory, and no variables are used unless absolutely necessary. In addition, since PRINT statements use up a lot of memory, onscreen description was kept to a minimum.

It was necessary to devise a more efficient method of storing data than the customarily used arrays. The VIC-20 has an area of memory (828–1019) which is used only when LOADing and SAVEing data from or to the Datassette. That area is ordinarily untouched by BASIC and is a handy location for storing machine language routines. Sigma Mission uses a coding system to place a map of the adventure world into the cassette buffer, where it can be used by the computer as a reference. Other addresses in that buffer are used to hold variable information or for temporary storage.

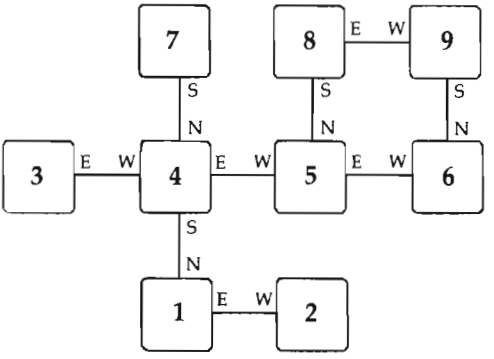
Designing a World

The first step in developing Sigma Mission was to design the world in which the game would be played. Obviously, areas of the world must interconnect, and there must be some orderly means of traveling from one to another. In addition, the available pathways must stay consistent throughout the game. Thus, the first step was to draw the map shown in Figure 1.

Unfortunately, computers can't store a map in the format of Figure 1, so a coding system was developed that would break the map down into terms the computer could understand.

To see how this was done, refer again to the map. For instance, look at Room 1. From Room 1 you can go north to Room 4 and east to Room 2. No doors lead south or west. In Room 2, the only path is west, back to Room 1. From Room 4, you can go north to Room 7, south to Room 1, east to Room 5, and west to Room 3, and so on.

Figure 1. The World of Sigma Mission



1. Control Room (start)
2. Reactor Room
3. Observatory
4. Living Quarters
5. Hydroponic Garden
6. Supply Room
7. Transportation Terminal
8. Life Support System
9. Main Computer Complex (final goal)

Following this procedure for each of the rooms, you can construct a move matrix chart (Table 1), using a zero to indicate directions in which movement is not possible. Rooms are identified along the left side of the chart, while directions are listed across the top. The numbers represent the rooms entered by travel from any particular point in any particular direction. For example, if you go north from Room 1, you enter Room 4. Again, the number 0 indicates that movement is not possible.

Table 1. Move Matrix

Room No.	Direction			
	N	S	E	W
1	4	0	2	0
2	0	0	0	1
3	0	0	4	0
4	7	1	5	3
5	8	0	6	4
6	9	0	0	5
7	0	4	0	0
8	0	5	9	0
9	0	6	0	8

The numbers from the move matrix chart are POKEd into the cassette buffer locations given in Table 2, occupying the memory area from 828 to 863.

Table 2. Cassette Buffer Locations

Room No.	Poke	Value
1	828	4
	829	0
	830	2
	831	0
2	832	0
	833	0
	834	0
	835	1
3	836	0
	837	0
	838	4
	839	0
4	840	7
	841	1
	842	5
	843	3
5	844	8
	845	0
	846	6
	847	4
6	848	9
	849	0
	850	0
	851	5
7	852	0
	853	4
	854	0
	855	0
8	856	0
	857	5
	858	9
	859	0
9	860	0
	861	6
	862	0
	863	8

Hither or Yon?

Now it's necessary to develop some type of reference so the computer will know where to look next. This is customarily handled by a two-dimensional array. But since the object here is to conserve memory, it's much better to POKE the necessary values into the cassette buffer and use them as pointers to the needed map areas.

Since a value higher than 255 cannot be POKEd into memory on an eight-bit microprocessor, and since the values

POKEd will be pointing to addresses in the range 828–863, some way to use numbers less than 255 must be found. By subtracting 768 (a purely arbitrary value) from 864–872 (the next block of unused cassette buffer locations), you get values 60–92. Those values, which correspond to locations 828–860, *can* be POKEd into memory, giving the program the information it needs to decide where to go next.

Data for each room is stored in four consecutive addresses. The information for Room 1 begins at location 828, the information for Room 2 begins at address 832, and so on. Thus, you POKE 864 with 60 to refer to Room 1, POKE 865 with 64 to refer to Room 2, and so on. The values are summarized in Table 3.

Table 3. Room Data

Pointer = 768 + Value

Poke	Value	Points to
864	60	828
865	64	832
866	68	836
867	72	840
868	76	844
869	80	848
870	84	852
871	88	856
872	92	860

Rules of the Rooms

In order to make the game more interesting, a unique set of conditions should apply to each room. Once again, to save memory, avoid the use of variables by POKEing the data into the cassette buffer and PEEKing the address as needed. This time, however, you'll signify the various conditions by turning individual bits in each byte on or off. This will let you use only one byte of memory to store information on as many as eight conditions. The storage area for the Sigma Mission condition table begins at 878; the area between 873 and 877 will be used for temporary storage of data while the program is running.

Each byte contains eight bits, as shown in Table 4, and each bit is turned on or off to indicate a desired condition. To set up the numbers to POKE, use the binary numbering system. The bits are numbered 0–7, with 0 at the right-hand side. For the bit to be off, set it to zero. Use the value 1 to signify that a bit is on.

Table 4. Binary Numbers

Binary								Decimal
128	64	32	16	8	4	2	1	
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	1	5
0	0	0	0	0	1	1	0	6
0	0	0	0	0	1	1	1	7
0	0	0	0	1	0	0	0	8
0	0	0	0	1	0	0	1	9
0	0	0	0	1	0	1	0	10

The maximum value which can be stored is 255, with all bits on (11111111 in binary). To check for a condition, PEEK the address, then AND the value with the bit value you are checking for. You'll be using this line: IF PEEK (Addr) AND...THEN....

For example, to check for bit 0, AND with 1 (2↑0). For bit 1, AND with 2 (2↑1), and so on, through checking for bit 7 by ANDing with 128 (2↑7).

The result of those operations specifies what is present at any particular time. In the case of Sigma Mission, that includes locked doors, unlocked doors, computer terminals, holes in the floor, lanterns, keys, oxygen masks, metal bars, and insufficient oxygen. Again, the conditions for each room are specified with only two bytes of memory for each room, as shown in Table 5.

Table 5. Room Conditions

Room No.	Address	Bit* 7	6	5	4	3	2	1	0	Decimal Value
1	878	0	0	0	1	0	0	0	0	16
2	879	0	0	1	0	0	0	0	0	32
3	880	1	1	0	1	0	1	0	0	212
4	881	0	1	0	1	1	0	0	1	89
5	882	0	1	1	0	0	1	0	0	100
6	883	1	1	0	1	0	1	0	1	213
7	884	0	1	1	0	0	0	1	0	98
8	885	0	1	1	0	0	1	0	0	100
9	886	1	1	0	1	1	0	1	0	218

*Bit No.	Condition	Value
0	North Door Locked	1
1	South Door Locked	2
2	East Door Locked	4
3	West Door Locked	8
4	Light	16
5	Dark	32
6	Computer Terminal	64
7	Hole in Floor	128

Room No.	Address	Bit* 7	6	5	4	3	2	1	0	Decimal Value
1	887	0	0	0	0	1	0	0	1	9
2	888	0	0	0	0	1	1	1	0	14
3	889	0	0	0	0	1	0	1	0	10
4	890	0	0	0	0	0	0	1	1	3
5	891	0	1	1	0	0	0	0	0	96
6	892	0	0	0	0	1	0	0	1	9
7	893	0	0	1	0	1	1	0	0	44
8	894	0	0	0	0	0	1	1	1	7
9	895	0	1	0	0	0	0	0	0	64

*Bit No.	Condition
0	Lantern
1	Key
2	Box
3	Oxygen Mask
4	Metal Box
5	No Air

Setting Up DATA Statements

Calculating these values is a tedious but necessary part of programming an adventure (like Sigma Mission) for the unexpanded VIC. After all the values have been calculated, they are put into DATA statements that the program can use. Be careful setting up the DATA statements, for a single error in transcription can cause problems that are maddeningly difficult to detect.

Note that the DATA statements in lines 1, 2, and 3 will put zeros in locations 874, 875, 876, and 877. Note too that address 873 holds the pointer to the present location. Initially, this is set to point to 828, so 60 is POKEd into this location, specifying Room 1.

Looking at the program listing, you'll notice that many lines contain IF-THEN statements. These statements are usually ANDed to a PEEK to check for a bit setting (that is, for a condition for that particular room). POKE is used to change a condition in a room or store temporary data in the area reserved for storage in the cassette buffer. Clearly, this approach to storage lets you manipulate large amounts of data without sacrificing the area of RAM used by BASIC.

Parsing

The parsing routine, used to interpret the player's inputs by reading C$, begins with line 55. Lines 56 and 57 read the leftmost two characters of C$ and use them to evaluate what type of action is being requested. Line 58 evaluates the first character to determine if an inventory of items carried has been requested. Since Sigma Mission uses limited memory, the valid commands are limited to GO, TA (take), and I (inventory). If these values are not found in C$, the computer will respond with I DON'T UNDERSTAND and return to line 54 to await another input. Note that you can enter TA or spell out TAKE, and the parsing routine will correctly interpret the input.

To take an object, simply use the first three letters of the object's name.

Elaborate parsing is not possible when working with limited memory, and the keywords are limited due to the size of the available memory. Obviously, with larger memory configurations, many commands could have been included in the parsing routine, making input more conversational.

Error trapping is also limited by available memory. Illegal moves are possible in Sigma Mission; for example, you can take objects that aren't in the rooms. Thus, you should follow the implied rules and make only valid moves (though with larger memory configurations, it would be wise to add routines to trap any illegal moves).

The program checks your inventory (by PEEKing 874 and testing the bits for the required conditions) and branches accordingly. For instance, it changes room conditions (from dark to light) if you're carrying a lantern. Similarly, if you encounter the Zapper, it checks to see if you're carrying the box. If so, the Zapper takes only the box but leaves you alone; if not, it takes everything and zaps you with the transporter beam too. It also checks to see if you have a key when you try to get to the master computer in the master computer room. If you do, you may be able to access the computer. If not, you will certainly be teleported out of the computer room and back to the beginning.

Entering the Program

Since Sigma Mission so nearly fills the unexpanded VIC's memory, use particular care when typing it in. Leave spaces *only* where called for in the PRINT statements, and omit closing quotation except where absolutely necessary.

Use "The Automatic Proofreader," found in Appendix C, when typing in this program. When you run Sigma Mission, the proofreader routine will be replaced in memory by the data for Sigma Mission.

If you should get an OUT OF MEMORY ERROR when you try to save Sigma Mission, enter CLR (in the immediate mode), then try to save it again. CLR will reset the pointers for all variables and free up a little more memory.

If you are curious about the amount of memory used by Sigma Mission, enter PRINT FRE(0) after running the program.

A Few Hints

As you read this article and type in the game, you'll not only learn a new approach to programming text adventures, but you'll end up with a good game at the same time. Since you typed it in, the odds are that you'll know the tricks it takes to win. However, when showing the game to your friends, let

them figure out the puzzles and create a map themselves. That makes it a lot more fun—for you as well as for them.

Here are some other hints that may help you. You will find a number of different objects as you make your way toward the master computer. Some are essential for survival. Others, however, have no value in the game—but you can carry them around if you want to.

Remember that you need to breathe. If you enter a room with no air but have an oxygen mask, your supply of oxygen will last for only one move. It's best to get out of any room with no air as quickly as possible.

Initially, some rooms are dark. However, if you enter a dark room while carrying a lantern, the room will become lit. You must leave your lantern behind when you leave that room, but the room will remain lit throughout the rest of the game.

Finally, here's a hint to help you deal with the master computer: You will need a key to do so. There are three keys scattered through the rooms, so you have three chances. But be forewarned. If you use all three keys without successfully reprogramming the master computer, there is no way to win that particular game. You'll just have to quit and try again by pressing RUN/STOP-RESTORE, and then typing RUN to play again.

Sigma Mission

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 DATA4,0,2,0,0,0,0,1,0,0,4,0,7,1,5,3,8,0,6,4,9,0,
  0,5,0,4,0,0,0,5,9,0,0,6,0,8                    :rem 106
2 DATA60,64,68,72,76,80,84,88,92,60,0,0,0,0,16,32,
  212,89,100,213,98,100,218                      :rem 147
3 DATA9,14,10,3,96,9,44,7,64,1                   :rem 237
4 FORA=828TO895:READB:POKEA,B:NEXT:READY:POKE877,0
                                                 :rem 199
5 B=INT(5*RND(.)+1):GOTO24                       :rem 231
6 IFC$="N"THENC=0:GOTO11                         :rem 175
7 IFC$="S"THENC=1:GOTO13                         :rem 184
8 IFC$="E"THENC=2:GOTO15                         :rem 174
9 IFC$="W"THENC=3:GOTO17                         :rem 196
10 GOTO54                                          :rem 3
11 IFPEEK(877+Y)AND1THEN19                        :rem 46
12 GOTO20                                        :rem 254
```

```
13 IFPEEK(877+Y)AND2THEN19                          :rem 49
14 GOTO20                                            :rem 0
15 IFPEEK(877+Y)AND4THEN19                          :rem 53
16 GOTO20                                            :rem 2
17 IFPEEK(877+Y)AND8THEN19                          :rem 59
18 GOTO20                                            :rem 4
19 PRINT"DOOR LOCKED":GOTO54                       :rem 253
20 Z=Y:Y=PEEK(X+C):IFY=0THEN22                     :rem 128
21 X=PEEK(863+Y)+768:GOTO23                         :rem 59
22 PRINT"CAN'T DO THAT":GOTO54                      :rem 34
23 POKE873,X-768                                   :rem 140
24 X=PEEK(873)+768                                 :rem 227
25 D=PEEK(X):E=PEEK(X+1):F=PEEK(X+2):G=PEEK(X+3):P
   RINT"{CLR}ROOM: ";Y                              :rem 23
26 IFY=9THEN115                                    :rem 140
27 PRINT"{DOWN}MOVES:"PEEK(877):PRINT"DOORS LEAD
                                                   :rem 177
28 IFDTHENPRINT"NORTH                              :rem 166
29 IFETHENPRINT"SOUTH                              :rem 176
30 IFFTHENPRINT"EAST                                :rem 67
31 IFGTHENPRINT"WEST                                :rem 91
32 IFPEEK(877+Y)AND1THENPRINT"NORTH DOOR LOCKED
                                                   :rem 231
33 IFPEEK(877+Y)AND2 THENPRINT"SOUTH DOOR LOCKED
                                                   :rem 241
34 IFPEEK(877+Y)AND4THENPRINT"EAST DOOR LOCKED
                                                   :rem 142
35 IFPEEK(877+Y)AND8THENPRINT"WEST DOOR LOCKED
                                                   :rem 169
36 IFPEEK(877+Y)AND16THENPRINT"LIGHT IS ON :rem 97
37 IFPEEK(877+Y)AND32THENPRINT"IT IS DARK":GOSUB61
                                                    :rem 77
38 IFPEEK(877+Y)AND64THENPRINT"COMPUTER TERMINAL H
   ERE                                             :rem 164
39 PRINT"YOU SEE:":IFPEEK(877+Y)AND128THENGOSUB63
                                                    :rem 64
40 IFPEEK(874)AND1THEN42                           :rem 165
41 IFPEEK(886+Y)AND1THENPRINT"LANTERN      :rem 138
42 IFPEEK(874)AND2THEN44                           :rem 170
43 IFPEEK(886+Y)AND2THENPRINT"KEY           :rem 98
44 IFPEEK(874)AND4THEN46                           :rem 176
45 IFPEEK(886+Y)AND4THENPRINT"BOX          :rem 102
46 IFPEEK(874)AND8THEN48                           :rem 184
47 IFPEEK(886+Y)AND8THENPRINT"OXYGEN MASK :rem 137
48 IFPEEK(874)AND16THEN51                          :rem 227
49 IFPEEK(886+Y)AND16THENPRINT"METAL BAR"  :rem 30
50 IFPEEK(886+Y)AND64THEN68                        :rem 110
51 IFY=1THEN54                                      :rem 84
52 IFINT(10*RND(.)+1)=1THEN89                      :rem 138
```

```
53 IFINT(10*RND(.)+1)=2THEN71                    :rem 131
54 PRINT:PRINT"COMMAND";                          :rem 59
55 INPUTC$:POKE877,PEEK(877)+1                    :rem 20
56 IFLEFT$(C$,2)="GO"THENGOTO76                   :rem 75
57 IFLEFT$(C$,2)="TA"THEN101                      :rem 55
58 IFLEFT$(C$,1)="I"THEN92                       :rem 196
59 PRINT"I DON'T UNDERSTAND":GOTO55              :rem 185
60 GOTO6:GOTO24                                  :rem 174
61 IFPEEK(886+Y)AND1THENPOKE(877+Y),(PEEK(877+Y)-1
   6):RETURN                                      :rem 62
62 IFPEEK(874)AND1THENPRINT"YOUR LANTERN LIGHTS
   {3 SPACES}THE WAY.":RETURN                     :rem 92
63 IFI=1THENRETURN                               :rem 190
64 IFPEEK(877+Y)AND16THENPRINT"A HOLE IN THE FLOOR
   ":POKE877+Y,PEEK(877+Y)-128:RETURN            :rem 119
65 IFPEEK(877+Y)AND32THENPRINT"YOU FELL IN A HOLE.
   {3 SPACES}YOUR ARE TRAPPED.                    :rem 99
66 PRINT"MISSION TERMINATED":GOTO126              :rem 88
68 IFPEEK(874)AND8THENPRINT"NO AIR!!{14 SPACES}YOU
   R OXYGEN MASK SAVEDYOU.                       :rem 173
69 IFPEEK(874)AND8THENGOSUB128:POKE886+Y,PEEK(886+
   Y)-64:GOTO21                                   :rem 56
70 PRINT"NO AIR!!{2 SPACES}GASP!!{6 SPACES}YOU HAV
   E EXPIRED.":GOTO126                           :rem 204
71 IFY=1THENRETURN                               :rem 205
72 PRINT"AN ALIEN IS HERE                        :rem 208
73 IFPEEK(874)AND4THENPRINT"THE ALIEN TAKES YOUR
   {2 SPACES}BOX, BUT LETS YOU LIVE              :rem 109
74 IFPEEK(874)AND4THENPOKE874,PEEK(874)-4:GOSUB128
   :GOTO21                                       :rem 236
75 PRINT"THE ALIEN POINTS AT{3 SPACES}YOU,":GOTO12
   4                                              :rem 44
76 FORA=1TOLEN(C$)                                :rem 61
77 IFMID$(C$,A,1)="N"THENC$="N":GOTO83           :rem 143
78 IFMID$(C$,A,1)="S"THENC$="S":GOTO83           :rem 154
79 IFMID$(C$,A,1)="E"THENC$="E":GOTO83           :rem 127
80 IFMID$(C$,A,1)="W"THENC$="W":GOTO83           :rem 155
81 IFMID$(C$,A,1)="C"THENC$="C":GOTO84           :rem 117
82 NEXTA                                         :rem 234
83 GOTO6:GOTO24                                  :rem 179
84 PRINT"{CLR}COMPUTER TERMINAL                  :rem 121
85 PRINT:PRINT"ENTER CODE                        :rem 124
86 PRINT"NUMBER 1-2                              :rem 118
87 INPUTN                                         :rem 77
88 IFN<>INT(2*RND(.)+1)THEN91                    :rem 183
89 PRINT"YOU'VE BEEN ZAPPED!                     :rem 222
90 GOSUB128:Y=1:GOTO21                            :rem 91
91 POKE877+Y,0:GOTO21                            :rem 245
92 I=1:PRINT"{CLR}YOU HAVE:                      :rem 249
```

```
93 IFPEEK(874)AND1THENPRINTTAB(5)"A LANTERN
                                          :rem 168
94 IFPEEK(874)AND2THENPRINTTAB(5)"A KEY   :rem 127
95 IFPEEK(874)AND4THENPRINTTAB(5)"A BOX   :rem 130
96 IFPEEK(874)AND8THENPRINTTAB(5)"AN OXYGEN MASK
                                          :rem 242
97 IFPEEK(874)AND16THENPRINTTAB(5)"A METAL BAR
                                           :rem 22
98 IFPEEK(874)AND31THEN100                 :rem 16
99 PRINTTAB(5)"NOTHING                    :rem 149
100 GOSUB128:GOTO24                       :rem 133
101 FORA=1TOLEN(C$)                        :rem 98
102 IFMID$(C$,A,3)="LAN"THEN108           :rem 200
103 IFMID$(C$,A,3)="KEY"THEN109           :rem 216
104 IFMID$(C$,A,3)="BOX"THEN110           :rem 209
105 IFMID$(C$,A,3)="OXY"THEN111           :rem 234
106 IFMID$(C$,A,3)="MET"THEN112           :rem 210
107 NEXT:GOTO113                          :rem 223
108 POKE874,PEEK(874)+1:POKE886+Y,PEEK((886)+Y)-1:
    GOTO25                                :rem 244
109 POKE874,PEEK(874)+2:POKE886+Y,PEEK((886)+Y)-2:
    GOTO25                                :rem 247
110 POKE874,PEEK(874)+4:POKE886+Y,PEEK((886)+Y)-4:
    GOTO25                                :rem 243
111 POKE874,PEEK(874)+8:POKE886+Y,PEEK((886)+Y)-8:
    GOTO25                                :rem 252
112 POKE874,PEEK(874)+16:POKE886+Y,PEEK((886)+Y)-1
    6:GOTO25                               :rem 91
113 PRINT"TAKE WHAT?":GOTO55              :rem 219
114 GOSUB128:GOTO25                       :rem 139
115 IFPEEK(874)AND2THEN117                 :rem 12
116 GOTO89                                 :rem 66
117 PRINT"{CLR}{RVS}{4 SPACES}MASTER COMPUTER
    {3 SPACES}"                            :rem 74
118 PRINT"ENTER PROPER CODE TO{2 SPACES}CORRECT MA
    STER PROGRAM                           :rem 83
119 PRINT:PRINT"ENTER NUMBER 1-5          :rem 235
120 INPUTN                                :rem 113
121 IFN=BTHEN125                          :rem 183
122 PRINT"ACCESS CODE ERROR               :rem 155
123 GOSUB128                              :rem 177
124 Y=1:POKE874,0:GOSUB128:GOTO21         :rem 241
125 PRINT"{CLR}{5 DOWN}{RIGHT}MASTER COMPUTER RESE
    T{6 DOWN}                             :rem 112
126 PRINT"{2 SPACES}TYPE {RVS} RUN {OFF} TO PLAY
    {3 SPACES}AGAIN                        :rem 92
127 END                                   :rem 113
128 FORA=1TO6000:NEXT:RETURN               :rem 45
```

6
Machine Language Games

Machine Language Games

BASIC is adequate for many game programs, but in some cases machine language is undoubtedly better. It is faster and more responsive than BASIC, so it's ideal when you want smoothly flowing graphics or realistic play.

The four games in this chapter give you an idea of what machine language can do. "Shooting Gallery," by Siva Krishna and Prabhudeva Kavi, turns your unexpanded VIC into a remarkably realistic rifle range.

If you prefer your target practice in outer space, Steve Haynal's "Demons of Osiris" may be just the game for you. It pits you against wave after wave of Osirian attackers, which bound toward you at machine language speed.

"CUT-OFF!," by Tom R. Halfhill, is another example of the fast action possible with ML. It lets two players go head-to-head in an exciting race to cut each other off at the pass.

"Trenchfire," by Don Gibson, offers a race of another kind. You're winging your way through a trenchlike earthquake fault, blasting wave after wave of fighters from the lair of evil King Krypos. Just don't run into the walls, which are scrolling past your view port at incredible speeds—and in 3-D! When typing in these games, follow the instructions in the articles. You should be sure to use "The Automatic Proofreader" (Appendix C) or "Tiny MLX" (Appendix D) as specified.

Shooting Gallery

Siva R. Krishna and Prabhudeva N. Kavi

"Shooting Gallery" combines BASIC and ML to give you target practice in the comfort of your home. A two-part program for the unexpanded VIC.

The object of "Shooting Gallery" is to knock down as many targets as possible. You must hit as many targets as possible on the lower level in order to advance to higher level play.

It's not as easy as it sounds. You get points for hitting the targets, but every missed shot costs you five points. Go easy on the rapid fire. You'll also need to watch out for two careless robot attendants who keep getting in your way; don't hit one of them or you'll lose 200 points. You have two minutes to get your best possible score.

About the Programs

It is not necessary to understand machine language techniques to enjoy this game, but some programmers may like to know how the routines work. The first program loads the main machine language routine to the top of memory. That routine detects collisions (hits), increments scores, reads the joystick, and controls all motion.

To type in the game, first type in and SAVE Program 1. Then type in and SAVE Program 2, using the filename SG. You must name the second program SG, since that is the name that the first program looks for. As given, the programs are written for disk. To use them with the Datassette, change the 8 to a 1 in line 45 of Program 1.

Program 1 uses dynamic keyboard techniques to automatically load the second program. One of the first things that the second program does is to relocate the routine POKEd by Program 1 into a lower part of memory. This is necessary because the DATA statements in Program 1 occupy so much memory that the routine could not otherwise be POKEd into its proper place. Lines 15–203 then redefine the character set. Lines 300–328 provide DATA for an interrupt-driven music program; at the end of the music program there is a JMP instruction to the game routine located at address $1910 (hex).

Following that routine, there is another JMP to the normal

interrupt service routine at $EABF. The targets are PRINTed to the screen in lines 120–135. Lines 500–540 print the score and elapsed time.

To replay the game, press RUN/STOP–RESTORE and then RUN it again.

Program 1. Shooting Gallery Loader

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
1 POKE51,0:POKE55,0:POKE52,27:POKE56,27:CLR:rem 58
10 FORI=7179TO7329:READA:POKEI,A:SU=SU+A:NEXT
                                          :rem 128
15 IFSU<>18614THENPRINT"ERROR100-110":STOP:rem 117
20 SU=0:FORI=7330TO7509:READA:POKEI,A:SU=SU+A:NEXT
                                          :rem 197
25 IFSU<>20527THENPRINT"ERROR110-122":STOP:rem 118
30 SU=0:FORI=7510TO7679:READA:POKEI,A:SU=SU+A:NEXT
                                          :rem 206
35 IFSU<>23469THENPRINT"ERROR122-133":STOP:rem 132
45 PRINT"{CLR}{3 DOWN}LOAD";CHR$(34);"SG";CHR$(34)
   ;",8";"{HOME}":POKE 631,13:POKE 198,1  :rem 145
100 DATA 238,251,3,173,251,3,201,2,16,3,76,198,25,
    169,0                                 :rem 134
101 DATA 141,251,3,238,250,3,173,250,3,201,4,16,3,
    76,75                                 :rem 123
102 DATA 25,169,0,141,250,3,162,20,173,131,30,72,1
    89,110,30                              :rem 58
103 DATA 232,157,110,30,202,202,224,255,208,243,10
    4,141,110,30,238                      :rem 139
104 DATA 249,3,173,249,3,201,6,16,3,76,116,25,169,
    0,141                                 :rem 139
105 DATA 249,3,173,176,30,72,162,1,189,176,30,202,
    157,176,30                            :rem 142
106 DATA 232,232,224,22,208,243,104,141,197,30,238
    ,248,3,173,248                         :rem 76
107 DATA 3,201,3,16,3,76,157,25,169,0,141,248,3,17
    3,241                                 :rem 135
108 DATA 30,72,162,240,189,0,30,232,157,0,30,202,2
    02,224,219                            :rem 110
109 DATA 208,243,104,141,220,30,238,247,3,173,247,
    3,201,3,16                            :rem 118
110 DATA 3,76,198,25,169,0,141,247,3,173,30,31,72,
    162,31                                :rem 187
111 DATA 189,0,31,202,157,0,31,232,232,224,52,208,
    243,104,141                           :rem 155
```

```
112 DATA 51,31,169,32,174,243,3,157,229,31,173,19,
    145,72,169                                    :rem 147
113 DATA 0,141,19,145,173,17,145,41,16,240,3,238,2
    43,3,173                                      :rem 23
114 DATA 17,145,41,32,208,8,173,240,3,208,3,238,24
    0,3,104                                       :rem 226
115 DATA 141,19,145,173,34,145,72,169,0,141,34,145
    ,173,32,145                                   :rem 185
116 DATA 41,128,240,3,206,243,3,173,243,3,201,19,4
    8,5,169                                       :rem 236
117 DATA 19,141,243,3,170,169,41,157,229,31,104,14
    1,34,145,173                                  :rem 233
118 DATA 243,3,201,0,16,5,169,1,141,243,3,173,237,
    3,162                                         :rem 125
119 DATA 0,129,251,173,240,3,201,1,48,14,173,242,3
    ,208,9                                        :rem 179
120 DATA 76,211,26,173,243,3,141,241,3,173,242,3,2
    08,3,76                                       :rem 229
121 DATA 191,234,238,242,3,201,13,48,29,169,0,141,
    242,3,169                                     :rem 81
122 DATA 32,141,237,3,169,31,133,252,169,238,133,2
    51,169,0,141                                  :rem 231
123 DATA 13,144,141,240,3,76,191,234,173,242,3,73,
    238,141,13                                    :rem 124
124 DATA 144,169,31,133,252,169,255,133,251,56,165
    ,251,237,241,3                                :rem 86
125 DATA 105,0,133,251,174,242,3,56,165,251,233,22
    ,133,251,165                                  :rem 220
126 DATA 252,233,0,133,252,202,224,0,208,238,161,2
    51,201,40,240                                 :rem 252
127 DATA 17,201,39,240,21,201,32,240,20,76,226,26,
    141,244,3                                     :rem 62
128 DATA 238,246,3,169,40,141,237,3,76,191,234,238
    ,245,3,169                                    :rem 158
129 DATA 32,141,237,3,162,0,169,42,129,251,76,191,
    234,0,0                                       :rem 236
130 DATA 0,238,242,3,24,169,28,237,243,3,141,241,3
    ,76,69                                        :rem 194
131 DATA 26,173,242,3,141,244,3,169,18,141,242,3,2
    38,246,3                                      :rem 30
132 DATA 169,0,141,245,3,76,183,26,255,239,251,255
    ,255,223,62                                   :rem 198
133 DATA 215,64,130,72,68,0,40,233,64,218,68,72,12
    8,0,64                                        :rem 195
```

Program 2. Shooting Gallery, Main Program

```
1 POKE52,25:POKE51,0::POKE56,25:POKE55,0:CLR:POKE3
  6869,255:POKE251,31:POKE252,255          :rem 12
5 IFPEEK(6416)=238ANDPEEK(6417)=251THEN12 :rem 103
10 FORI=0TO490:POKE6416+I,PEEK(7179+I):NEXT:rem 83
12 PRINT"{CLR}{BLU}{3 RIGHT}SHOOTING GALLERY{OFF}"
   :POKE36879,90                           :rem 89
15 FORI=7168TO7423:POKEI,255-(PEEK(I+25600)):NEXT
                                           :rem 116
16 FORI=7424TO7640:POKEI,PEEK(I+25600):NEXT:rem 87
30 FORI=1TO64:READA:POKE7447+I,A:NEXT      :rem 62
90 TI$="000000":SC=0:FORI=828TO1019:POKEI,0:NEXT
                                           :rem 70
120 PRINT"{4 DOWN}{WHT}$((($(($(($(($(($(($(((("
                                            :rem 1
125 PRINT"{DOWN}{BLU}(((%((%((%((%((%((%((%((%"
                                           :rem 244
130 PRINT"{RED}&((&((&((&((&((&((&(((("    :rem 227
135 PRINT"{DOWN}{YEL}(((#((#((#((#((#((#((#((#"
                                           :rem 102
200 DATA 96,224,105,127,126,62,28,255,24,24,14,7,1
    90,252,246,59                           :rem 35
201 DATA 8,48,112,48,56,60,24,255,60,66,153,165,16
    5,153,66,255                           :rem 252
202 DATA 56,68,170,130,198,186,68,56,00,00,00,00,0
    0,00,00,255                            :rem 157
203 DATA 56,56,56,56,56,124,124,124,0,16,16,16,16,
    16,16,0                                :rem 233
299 POKE6737,13:IFNU>0THEN500              :rem 216
300 FORI=830TO970:READC:POKEI,C:SU=SU+C:NEXT:POKE3
    6878,9                                  :rem 36
305 IFSU<>14379THENPRINT"ERROR320-328":STOP
                                           :rem 186
310 SYS830:FORI=980TO1019:POKEI,0:NEXT:POKE1005,32
                                            :rem 43
320 DATA 120,169,5,141,60,3,169,6,141,61,3,169
                                           :rem 208
321 DATA 133,133,0,169,3,133,1,169,93,141,20,3,169
    ,3,141,21,3,88,96                      :rem 214
322 DATA 206,61,3,208,28,72,152,72,172,60,3,200,17
    7,0,141,61,3,200,177,0,201             :rem 121
323 DATA 1,240,12,141,11,144,140,60,3,104,168,104,
    76,16 ,25 ,160,255,208,243              :rem 29
325 DATA 200,100,60,215,15,212,15,215,15,223,30,22
    5,1,100,15,225,1,100,15,225            :rem 141
326 DATA 15,228,15,225,15,223,15,228,15,225,15,223
    ,30,209                                :rem 235
```

```
327 DATA 4,100,30,209,15,207,15,209,15,219,30,221,
    2,100,8,221,2,100,8,221                    :rem 215
328 DATA 15,228,15,225,15,223,15,228,15,225,15,223
    ,30,207,1,1                                :rem 165
500 PRINT"{HOME}{DOWN}{BLK}TIME:";RIGHT$(TI$,3);TA
    B(11)"SCORE:{7 SPACES}{7 LEFT}";SC         :rem 172
503 I=INT(RND(1)*9):IFI>0THEN520               :rem 105
505 POKE8076+Z,32:POKE8081+Z,32:Z=INT(RND(1)*15)+2
    :POKE8076+Z,39:POKE8081+Z,39               :rem 22
520 SC=INT(SC+(((PEEK(1014)*6)*PEEK(1012)/2)-200*P
    EEK(1013))):IFSC<0THENSC=0                 :rem 220
525 POKE37677,250:NU=NU+PEEK(1014):IFPEEK(1008)>0T
    HENSC=SC-5                                 :rem 48
526 IFNU>6THENPOKE6737,18                      :rem 13
530 POKE1014,0:POKE1013,0:POKE1012,0:IFVAL(TI$)>20
    0THENPOKE36878,0:GOTO530                   :rem 95
535 POKE36879,PEEK(1013)+90:IFNU>17THENPRINT"
    {HOME}":NU=1:GOTO120                       :rem 78
540 GOTO500                                    :rem 103
```

Demons of Osiris

Steve Haynal

You must defend your fleet of base ships against wave after wave of lightning-fast Osirian attackers as they weave and dodge through your covering fire. But the Osirians do not descend blindly; they counter your evasive moves and seek you out. Theirs is a maniacal mission. Written for the unexpanded VIC.

"Demons of Osiris" is a fast-paced, arcade-style machine language game. The object is to shoot the falling Osirians, but at the same time they'll use their intelligence to try to destroy you. You can choose from 240 speed levels, with level 1 being the fastest. You can also give yourself as many as 240 base ships. Be prepared to battle as many as eight Osirians at a time.

Your base ship is located at the bottom of the screen. You control its functions as follows: Press T to move left, U to move right, and SHIFT to fire. Pressing the SHIFT-LOCK key will give you rapid fire. When the screen flashes red, it means you have lost a base ship.

Simple But Effective

The Osirians have a simple but effective strategy, and they have both a defensive and an offensive move. On a defensive move, the Osirians will dodge your oncoming bullet, moving either right or left. On the offensive, they will move to one side of your line of fire. They do not come down directly above you, because it would increase their chances of being hit. The Osirians can destroy your base ship by being in the space directly above your base ship, directly above you to the right, or directly above you to the left. On some occasions they will activate a special defensive mechanism which triggers evasive action around your missiles.

The best strategy is to keep moving and fire rapidly. At slow speeds (25–240), try to aim as much as possible. At fast speeds (1–24), things move so quickly it's best just to try to dodge the Osirians.

You'll Need to Abbreviate

The machine language portion of Demons of Osiris takes 696 bytes, while the BASIC part (which runs with the machine language portion) is only three lines long. The machine language portion is in the form of DATA statements which are POKEd into memory. The whole program, including the DATA statements, takes all of an unexpanded VIC-20's memory.

Because of the VIC's limited memory, most of the program lines are quite long. You may need to abbreviate some BASIC keywords (see Appendix D of *Personal Computing on the VIC-20,* which came with your computer). In particular, you should use the abbreviation for DATA (D and SHIFT-A) in lines 35–190.

After typing the program, be sure to SAVE it before you RUN it. One mistake in the DATA statements might cause a crash, and you would have to type in the whole program again.

When you run it, there will be a short wait while the computer reads the DATA. It will then ask you for the speed and the number of base ships you want; in each case, you should input a value in the range 1–240. An average game would use 60 for speed and 5 for ships. There will again be a short wait, to allow you time to position your fingers on the T and U keys. Press SHIFT-LOCK if you want rapid fire. Otherwise, press SHIFT for normal fire. When you finish the game, the screen will display your score and give you a chance to select the speed and the number of ships for the next round.

Demons of Osiris

For error-free program entry, refer to the "Automatic Proofreader" article in Appendix C. Remember, do not type the checksum number at the end of each line. For example, do not type rem:123.

```
10 POKE52,27:POKE56,27:POKE51,71:POKE55,71:PRINT"
   {CLR}":FORA=6984TO7679:READB:POKEA,B:NEXT
                                          :rem 128
15 POKE649,10:INPUT"SPEED";A:INPUT"SHIPS";B:IFA>24
   0ORB>240ORA<1ORB<1THEN15               :rem 155
20 POKE7074,A:POKE7039,B:POKE649,0:FORB=0TO2000:NE
   XT                                     :rem 144
25 SYS6984:POKE36869,240:PRINT"{CLR}SCORE:"PEEK(24
   8)+PEEK(249)*256:GOTO15                :rem 196
35 DATA162,10,169,0,149,247,202,208,251,168,169,59
   ,157,0,30,157,0,31,232,208,247,141,15  :rem 237
```

```
40 DATA144,169,255,141,5,144,169,15,141,14,144,138
   ,157,0,150,157,228,150,232,208,247       :rem 93
45 DATA169,6,162,22,157,227,151,202,208,250,169,5,
   133,253,200,208,253,232,208,253,169      :rem 145
50 DATA238,133,251,169,31,133,252,169,63,145,251,1
   65,197,201,50,240,31,201,51,240,13       :rem 76
55 DATA140,13,144,162,63,200,208,253,202,208,250,2
   40,41,165,251,201,249,240,237,32,202     :rem 159
60 DATA27,230,251,76,196,27,165,251,201,228,240,22
   3,32,202,27,198,251,169,63,145,251       :rem 99
65 DATA208,215,169,129,141,13,144,169,59,145,251,9
   6,169,1,44,141,2,240,44,162,66,189       :rem 119
70 DATA161,31,201,61,240,35,202,208,246,165,251,56
   ,233,22,133,251,169,61,145,251,165       :rem 78
75 DATA251,24,105,22,133,251,140,13,144,169,160,14
   1,11,144,141,10,144,232,208,253,169      :rem 114
80 DATA30,133,255,169,21,133,254,162,21,160,22,177
   ,254,201,61,208,29,32,246,28,177,254     :rem 190
85 DATA201,59,240,9,32,14,29,32,232,28,76,50,28,16
   9,61,145,254,32,232,28,169,59,145        :rem 64
90 DATA254,136,208,218,32,232,28,202,208,210,162,2
   2,189,255,29,201,61,208,5,169,59,157     :rem 202
95 DATA255,29,202,208,241,140,10,144,140,11,144,16
   2,66,189,255,29,201,62,240,21,202,208    :rem 223
100 DATA246,32,86,29,165,141,162,0,232,56,233,12,1
    76,250,169,62,157,255,29                :rem 118
105 DATA169,31,133,255,169,227,133,254,160,22,177,
    254,201,63,240,3,136,208,247            :rem 55
110 DATA132,250,32,246,28,177,254,201,62,208,6,169
    ,59,145,254,16,27,136,177,254,201,62    :rem 199
115 DATA208,7,169,59,145,254,200,16,13,200,200,177
    ,254,201,62,208,24,169,59,145,254,136   :rem 244
120 DATA32,232,28,32,4,29,169,59,160,22,153,227,31
    ,136,208,250,76,130,27,160,22,169,59    :rem 190
125 DATA145,254,136,208,251,162,21,32,246,28,160,2
    2,177,254,201,62,208,3,32,113,29,136    :rem 178
130 DATA208,244,202,208,236,76,148,27,165,254,24,1
    05,22,133,254,165,255,105,0,133         :rem 192
135 DATA255,96,165,254,56,233,22,133,254,165,255,2
    33,0,133,255,96,177,254,201,63,240      :rem 108
140 DATA12,201,61,208,67,230,248,208,13,230,249,20
    8,9,169,42,141,15,144,198,253,240,69    :rem 195
145 DATA169,60,145,254,165,255,24,105,120,133,255,
    177,254,72,169,2,145,254,169,222,141    :rem 207
150 DATA13,144,230,146,208,252,206,13,144,48,247,1
    04,145,254,165,255,56,233,120,133,255   :rem 233
155 DATA169,59,145,254,141,15,144,96,169,62,145,25
    4,96,72,138,72,152,72,32,148,224,104    :rem 226
```

```
160 DATA168,104,170,104,96,160,0,140,14,144,169,27
    ,141,15,144,104,104,96,169,240,141      :rem 78
165 DATA12,144,169,59,145,254,32,86,29,32,232,28,1
    38,24,105,32,10,10,197,141,16,42,177   :rem 195
170 DATA254,201,59,208,6,169,62,145,254,16,58,169,
    48,197,141,16,12,192,1,240,8,136,169   :rem 217
175 DATA62,145,254,200,16,40,192,22,240,240,200,16
    9,62,145,254,136,16,28,196,250,240      :rem 84
180 DATA228,48,12,136,196,250,208,1,200,169,62,145
    ,254,16,10,200,196,250,208,1,136,169   :rem 187
185 DATA62,145,254,169,0,141,12,144,76,246,28,0,0,
    0,0,0,0,0,0,4,168,214,72,37,170,80      :rem 51
190 DATA20,0,16,56,40,40,40,56,16,60,90,255,126,36
    ,66,66,36,16,16,16,16,56,124,124,254   :rem 180
```

CUT-OFF!

Tom R. Halfhill

"CUT-OFF!" is a fast-paced, two-player game for the unexpanded VIC-20. Programmed entirely in machine language, it has ten levels of difficulty—ranging in speed from moderately slow to impossibly fast. A joystick is required. Be sure to unplug or switch off any memory expanders before typing in or running the game.

Over the years, some computer games have become classics. Usually they are simple in concept, yet universal in appeal, and general enough to be translated for almost any computer. Some examples are *Pong* (the granddaddy of all videogames), *Breakout, Lunar Lander,* and the venerable *Space Invaders.* For legal reasons they may be disguised by different names, but there probably isn't a computer or videogame machine anywhere for which some version of these all-time favorites isn't available.

Another classic game is *Blockade.* Again, it goes by different names, but the basic concept remains the same. Two players square off against each other by steering a moving line around the screen, trying to head off the other player or force him to crash into a wall or his own trail. This concept dates back to the early days of videogames—and it's developed one more time in "CUT-OFF!". This VIC version preserves all the traditional concepts and includes color, sound, and the broad range of speed levels possible only in a program written entirely in machine language.

Typing In CUT-OFF!

Pure machine language programs are usually more difficult to enter than BASIC programs, because they consist of seemingly endless streams of numbers. To make typing CUT-OFF! easier, we've listed the programs in MLX format.

You may already be familiar with MLX if you've typed in some of the machine language programs published in *COMPUTE!* or in *COMPUTE!'s Gazette.* MLX is a utility designed to make typing errors almost impossible. Note, however, that you must use a new version of MLX adapted especially for the VIC version of CUT-OFF!. It is a stripped-down version dubbed "Tiny MLX," and it allows you to enter the game on

an unexpanded VIC—something not possible with the full-length MLX. For the details, see "Tiny MLX for the VIC" in Appendix D.

Here are the starting and ending addresses you'll need to enter CUT-OFF! in the VIC-20. Put these values in line 210 of Tiny MLX:

Starting address—6063
Ending address—7658

To run the program, enter SYS 6063. To stop it, press RUN/STOP–RESTORE.

Remember, to load a machine language program from disk or tape, you must use this special form of the LOAD command:

LOAD"filename",8,1 (for disk)
LOAD"filename",1,1 (for tape)

If you forget to append the ,1 to the command, the program loads into the wrong area of memory and will not work.

Starting the Game

After you enter the proper SYS command, the game screen appears instantly. (One of the best things about machine language is that you don't have to wait around for programs to initialize.)

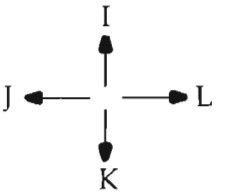
The opening screen allows you to select a skill level ranging from 0 (the slowest speed, suitable for youngsters) to 9 (recommended for superhumans only). The skill levels are spaced equally apart, so you might want to start at 3 or 4. The level you select remains the same for the entire game. To change levels in the middle of a game, press RUN/STOP–RESTORE and restart the program with the SYS command. Of course, this cancels the game in progress.

To choose a skill level, move the joystick up or down. You'll see the number on the screen change and wrap around if you try to go below 0 or above 9. To lock in your choice and begin the game, press the fire button.

The game starts with the players aimed at each other head-on. Player one uses the joystick to steer up, down, right, or left. Diagonal motion is not allowed.

Since the VIC has only one joystick port, player two must use the keyboard for control. Don't assume that this puts the keyboard player at a disadvantage, for many players seem to

adapt to the keyboard and end up with more control than the person with the joystick. This is due partly to the arrangement of the control keys, an arrangement sometimes seen in Apple games:



Notice how this differs from the usual I-J-K-M diamond pattern. Although the diamond seems the most logical way to go for four-way movement, in practice it's clumsy compared to this I-J-K-L arrangement. Try it. Rest your right index finger on the J key, your fourth finger on the L key, and then move your middle finger up and down on the I and K keys to control vertical movement. You may want to adopt this pattern for your next keyboard-controlled game.

The joystick button toggles a pause feature. To freeze the action, quickly press and release the button. This leaves you free to answer the phone or do other things. To restart the action, press and release the button again. The keyboard player cannot activate this feature.

Scoring and Winning

There are four ways you can crash: hitting a wall, running into the other player's trail, crossing your own trail, or backing into yourself by trying to reverse your direction.

After a crash, the surviving player is awarded points equal to the number of segments in the crashed player's trail. This means that the longer the players last before crashing, the more points are at stake. Thus, it's possible to catch up even if you're way behind.

Each time you crash, you lose one life. Each player starts with ten lives, and the game ends when one player runs out. After each crash, the screen updates the score and reminds you how many lives each player has left. To restart each round, press the joystick fire button.

When the game is over, you get a chance to change the skill level for the next game. Just to get a peek at how fast machine language can be, try a game at level 9. You'll be lucky if you can make one turn before crashing into a wall. Yet even this level had to be slowed down with delay loops!

CUT-OFF!

Load this program using "Tiny MLX" (Appendix D).

```
6063 :032,081,025,076,181,023,081
6069 :173,066,003,024,105,001,041
6075 :141,066,003,173,067,003,128
6081 :105,000,141,067,003,174,171
6087 :060,003,032,020,027,169,254
6093 :001,141,065,003,032,078,013
6099 :027,162,002,161,247,201,243
6105 :032,240,032,032,029,027,097
6111 :032,143,029,032,090,026,063
6117 :173,068,003,201,000,240,146
6123 :080,173,069,003,201,000,249
6129 :240,073,032,097,024,032,227
6135 :092,028,076,181,023,032,167
6141 :029,027,169,000,141,065,172
6147 :003,032,227,027,162,000,198
6153 :161,247,201,032,240,032,154
6159 :032,029,027,032,143,029,051
6165 :032,090,026,173,068,003,157
6171 :201,000,240,029,173,069,227
6177 :003,201,000,240,022,032,019
6183 :097,024,032,092,028,076,132
6189 :181,023,032,029,027,173,254
6195 :031,145,041,032,240,006,034
6201 :076,181,023,076,007,029,193
6207 :173,031,145,041,032,240,213
6213 :249,162,250,032,020,027,041
6219 :173,031,145,041,032,208,193
6225 :249,173,031,145,041,032,240
6231 :240,249,162,250,032,020,016
6237 :027,076,181,023,169,147,204
6243 :032,210,255,169,025,141,163
6249 :015,144,162,000,169,160,243
6255 :157,000,030,169,000,157,112
6261 :000,150,232,224,022,208,185
6267 :241,162,000,169,160,157,244
6273 :228,031,169,000,157,228,174
6279 :151,232,224,022,208,241,189
6285 :169,000,133,253,169,030,127
6291 :133,254,169,000,133,251,063
6297 :169,150,133,252,162,000,251
6303 :169,160,160,000,145,253,022
6309 :169,000,145,251,160,021,143
6315 :169,160,145,253,169,000,043
6321 :145,251,024,165,253,105,096
6327 :022,133,253,165,254,105,091
6333 :000,133,254,024,165,251,248
```

```
6339 :105,022,133,251,165,252,099
6345 :105,000,133,252,232,224,123
6351 :023,208,205,169,004,162,210
6357 :007,157,000,150,232,224,215
6363 :015,208,248,162,000,189,017
6369 :072,025,240,006,157,007,220
6375 :030,232,208,245,169,006,097
6381 :141,248,150,169,002,141,064
6387 :001,151,169,081,141,248,010
6393 :030,169,087,141,001,031,196
6399 :169,000,141,066,003,141,007
6405 :067,003,169,007,141,075,211
6411 :003,169,020,141,074,003,165
6417 :169,001,141,070,003,169,058
6423 :031,141,071,003,169,248,174
6429 :141,072,003,169,030,141,073
6435 :073,003,024,162,000,160,201
6441 :002,032,240,255,174,061,037
6447 :003,173,062,003,032,205,013
6453 :221,024,162,000,160,016,124
6459 :032,240,255,174,063,003,058
6465 :173,064,003,032,205,221,251
6471 :096,131,149,148,173,143,143
6477 :134,134,161,000,169,000,163
6483 :141,061,003,141,062,003,238
6489 :141,063,003,141,064,003,248
6495 :169,081,141,077,003,169,223
6501 :087,141,076,003,032,097,025
6507 :024,169,004,162,002,157,113
6513 :154,150,232,224,020,208,077
6519 :248,162,002,157,198,150,012
6525 :232,224,020,208,248,141,174
6531 :018,151,162,000,189,052,191
6537 :026,240,006,157,156,030,240
6543 :232,208,245,162,000,189,155
6549 :071,026,240,006,157,200,081
6555 :030,232,208,245,169,048,063
6561 :141,060,003,141,018,031,043
6567 :162,100,032,020,027,173,169
6573 :031,145,041,012,201,008,099
6579 :240,033,201,004,240,010,139
6585 :173,031,145,041,032,240,079
6591 :063,076,167,025,173,060,243
6597 :003,056,233,001,201,047,226
6603 :240,028,141,060,003,141,048
6609 :018,031,076,167,025,173,187
6615 :060,003,024,105,001,201,097
6621 :058,240,020,141,060,003,231
6627 :141,018,031,076,167,025,173
```

```
6633 :169,057,141,060,003,141,036
6639 :018,031,076,167,025,169,213
6645 :048,141,060,003,141,018,144
6651 :031,076,167,025,173,031,242
6657 :145,041,032,240,249,162,102
6663 :250,032,020,027,173,060,057
6669 :003,056,233,048,170,169,180
6675 :050,141,060,003,224,000,241
6681 :240,013,173,060,003,056,058
6687 :233,005,141,060,003,202,163
6693 :076,023,026,032,097,024,059
6699 :169,010,141,068,003,141,063
6705 :069,003,096,013,015,022,011
6711 :005,032,019,020,009,003,143
6717 :011,032,021,016,047,004,192
6723 :015,023,014,000,020,015,154
6729 :032,016,009,003,011,032,176
6735 :004,009,006,006,009,003,116
6741 :021,012,020,025,000,173,080
6747 :070,003,205,072,003,240,172
6753 :003,076,230,026,173,071,164
6759 :003,205,073,003,208,121,204
6765 :173,075,003,174,074,003,099
6771 :201,014,240,012,201,007,022
6777 :240,015,201,013,240,018,080
6783 :201,011,240,021,224,044,100
6789 :240,024,076,230,026,224,185
6795 :020,240,017,076,230,026,236
6801 :224,012,240,010,076,230,169
6807 :026,224,021,240,003,076,229
6813 :230,026,173,061,003,024,162
6819 :109,066,003,141,061,003,034
6825 :173,062,003,105,000,141,141
6831 :062,003,173,067,003,024,251
6837 :109,062,003,141,062,003,049
6843 :174,068,003,202,142,068,076
6849 :003,173,063,003,024,109,056
6855 :066,003,141,063,003,173,136
6861 :064,003,105,000,141,064,070
6867 :003,173,067,003,024,109,078
6873 :064,003,141,064,003,174,154
6879 :069,003,202,142,069,003,199
6885 :096,173,065,003,010,170,234
6891 :189,061,003,024,109,066,175
6897 :003,157,061,003,189,062,204
6903 :003,105,000,157,062,003,065
6909 :173,067,003,024,125,062,195
6915 :003,157,062,003,174,065,211
6921 :003,189,068,003,056,233,049
```

```
6927 :001,157,068,003,096,160,244
6933 :000,200,208,253,202,208,068
6939 :248,096,174,065,003,188,033
6945 :076,003,138,010,170,181,099
6951 :247,157,070,003,181,248,177
6957 :157,071,003,152,129,247,036
6963 :181,248,024,105,120,149,110
6969 :248,224,002,208,008,169,148
6975 :006,129,247,032,180,029,174
6981 :096,169,002,129,247,032,232
6987 :180,029,096,162,127,142,043
6993 :034,145,173,032,145,041,139
6999 :128,074,074,141,078,003,073
7005 :162,255,142,034,145,173,236
7011 :031,145,041,028,013,078,179
7017 :003,074,074,174,065,003,242
7023 :201,014,240,018,201,007,024
7029 :240,038,201,013,240,058,139
7035 :201,011,240,078,189,074,148
7041 :003,076,111,027,157,074,065
7047 :003,138,010,170,056,189,189
7053 :070,003,233,022,149,247,097
7059 :189,071,003,233,000,149,024
7065 :248,076,226,027,157,074,193
7071 :003,138,010,170,024,189,181
7077 :070,003,105,001,149,247,228
7083 :189,071,003,105,000,149,176
7089 :248,076,226,027,157,074,217
7095 :003,138,010,170,024,189,205
7101 :070,003,105,022,149,247,017
7107 :189,071,003,105,000,149,200
7113 :248,076,226,027,157,074,241
7119 :003,138,010,170,056,189,005
7125 :070,003,233,001,149,247,148
7131 :189,071,003,233,000,149,096
7137 :248,096,174,065,003,165,208
7143 :197,201,012,240,018,201,076
7149 :021,240,038,201,044,240,253
7155 :058,201,020,240,078,189,005
7161 :074,003,076,232,027,157,050
7167 :074,003,138,010,170,056,194
7173 :189,070,003,233,022,149,159
7179 :247,189,071,003,233,000,242
7185 :149,248,076,091,028,157,254
7191 :074,003,138,010,170,024,186
7197 :189,070,003,105,001,149,034
7203 :247,189,071,003,105,000,138
7209 :149,248,076,091,028,157,022
7215 :074,003,138,010,170,024,210
```

```
7221 :189,070,003,105,022,149,079
7227 :247,189,071,003,105,000,162
7233 :149,248,076,091,028,157,046
7239 :074,003,138,010,170,056,010
7245 :189,070,003,233,001,149,210
7251 :247,189,071,003,233,000,058
7257 :149,248,096,162,000,189,165
7263 :231,028,240,006,157,159,148
7269 :030,232,208,245,162,000,210
7275 :189,244,028,240,006,157,203
7281 :203,030,232,208,245,162,169
7287 :000,189,000,029,240,006,071
7293 :157,054,031,232,208,245,028
7299 :162,000,189,000,029,240,239
7305 :006,157,064,031,232,208,067
7311 :245,169,004,162,000,157,112
7317 :155,150,232,224,020,208,114
7323 :248,162,000,157,199,150,047
7329 :232,224,020,208,248,162,231
7335 :000,157,053,151,232,224,216
7341 :020,208,248,024,162,014,081
7347 :160,008,032,240,255,174,024
7353 :069,003,169,000,032,205,151
7359 :221,024,162,014,160,018,022
7365 :032,240,255,174,068,003,201
7371 :169,000,032,205,221,173,235
7377 :031,145,041,032,208,249,147
7383 :173,031,145,041,032,240,109
7389 :249,162,000,032,020,027,199
7395 :032,097,024,096,016,018,254
7401 :005,019,019,032,002,021,075
7407 :020,020,015,014,000,020,072
7413 :015,032,003,015,014,020,088
7419 :009,014,021,005,000,012,056
7425 :009,022,005,019,061,000,117
7431 :032,097,024,162,000,189,255
7437 :106,029,240,006,157,160,199
7443 :030,232,208,245,162,000,128
7449 :189,116,029,240,006,157,250
7455 :203,030,232,208,245,162,087
7461 :000,189,129,029,240,006,118
7467 :157,056,031,232,208,245,204
7473 :169,004,162,000,157,155,184
7479 :150,232,224,020,208,248,113
7485 :162,000,157,199,150,232,193
7491 :224,020,208,248,162,000,161
7497 :157,053,151,232,224,020,142
7503 :208,248,173,031,145,041,157
7509 :032,208,249,173,031,145,155
```

```
7515 :041,032,240,249,162,250,041
7521 :032,020,027,032,081,025,058
7527 :076,181,023,007,001,013,148
7533 :005,032,015,022,005,018,206
7539 :000,016,018,005,019,019,192
7545 :032,002,021,020,020,015,231
7551 :014,000,020,015,032,016,224
7557 :012,001,025,032,001,007,211
7563 :001,009,014,000,169,220,040
7569 :141,013,144,169,015,141,000
7575 :014,144,162,000,032,020,011
7581 :027,173,014,144,056,233,036
7587 :001,141,014,144,162,100,213
7593 :032,020,027,201,000,208,145
7599 :238,141,013,144,096,173,212
7605 :065,003,201,000,240,022,200
7611 :169,200,141,012,144,169,254
7617 :015,141,014,144,174,060,229
7623 :003,032,020,027,169,000,194
7629 :141,012,144,096,169,180,179
7635 :141,012,144,169,015,141,065
7641 :014,144,174,060,003,032,132
7647 :020,027,169,000,141,012,080
7653 :144,096,013,013,013,013,009
```

Trenchfire

Don Gibson

This is a fast-action space game which uses the speed of machine language (ML), the power of sprite graphics, and a special trick to simulate motion. It runs on the unexpanded VIC, but must be initially saved using "Tiny MLX" and an 8K or more expander.

As "Trenchfire" begins, you find yourself on a distant planet, speeding through a trench formed by an earthquake fault. You are attempting to infiltrate evil King Krypos's lair, where he holds your king captive. But first you must face Krypos's deadly kamikaze drone ships. The battle never seems to end--you blast and dodge debris only to encounter another wave of enemy ships. Only total concentration and quick reflexes can possibly bring success.

Loading the Game

You'll need an 8K expander to enter and save Trenchfire. You must also use "Tiny MLX," the abbreviated version of MLX, found in Appendix D.

Follow these procedures carefully:

1. Insert the 8K expander, turn on your computer, and enter this line:

```
POKE648,24:SYS58648:POKE642,26:SYS58232
```

2. Enter the short version of MLX.
3. Delete line 100 from the MLX program, and change line 210 as follows to reflect a starting address of 4352 and an ending address of 6079:

```
210 S=4352:E=6079
```

4. Type RUN.
5. Type in the program.
6. SAVE what you just typed into MLX to tape or disk.
7. Turn your computer off and remove the 8K expander. Turn it back on.
8. Now LOAD "TRENCHFIRE",1,1 for tape (or LOAD "TRENCHFIRE",8,1 for disk).
9. Enter SYS 4352 to run the program.

You start with three ships, earn a bonus ship for every 1000

points, and can achieve a maximum of seven ships. Extra features include a pause function (press SHIFT/LOCK) for freezing the game at any time and four levels of play. Use the function keys to choose a level: f1 gives you the beginner level, while f7 is for experts; f3 and f5 are intermediate and advanced. If you don't make a selection, the program defaults to the intermediate level. The expert level is only for the strong of heart. You also go up one level for every 250 points scored.

Trenchfire

Load this program using "Tiny MLX" (Appendix D).

```
4097 :000,000,032,210,255,169,155
4103 :008,141,015,144,169,032,004
4109 :162,000,027,089,201,232,212
4115 :224,242,208,248,169,032,118
4121 :162,000,037,016,031,232,247
4127 :224,006,080,248,162,000,239
4133 :050,016,017,157,242,150,157
4139 :232,224,242,208,245,162,076
4145 :000,063,016,018,157,228,019
4151 :151,232,224,022,208,245,113
4157 :169,000,091,016,003,169,253
4163 :064,141,046,145,169,141,005
4169 :141,020,003,169,017,141,052
4175 :021,123,175,192,141,046,009
4181 :145,076,170,018,162,000,144
4187 :098,016,150,041,015,065,220
4193 :000,119,016,168,200,152,240
4199 :201,008,208,002,169,005,184
4205 :157,242,150,105,000,242,237
4211 :208,230,162,000,157,016,120
4217 :151,041,015,168,200,152,080
4223 :201,008,208,002,169,005,208
4229 :157,228,151,232,224,022,123
4235 :208,234,096,238,060,003,210
4241 :172,060,003,192,010,208,022
4247 :008,032,088,017,160,000,200
4253 :174,016,003,076,191,234,083
4259 :007,007,007,007,007,007,205
4265 :007,007,007,007,000,196,137
4271 :016,007,007,007,007,007,226
4277 :007,007,007,007,006,006,221
4283 :006,006,006,006,006,006,223
4289 :006,007,000,197,016,006,169
4295 :006,006,006,006,006,006,235
4301 :006,006,005,005,005,005,237
```

```
4307 :005,005,005,005,006,007,244
4313 :007,007,007,006,005,005,254
4319 :005,019,019,005,005,005,025
4325 :007,007,007,112,033,007,146
4331 :007,005,006,006,006,006,015
4337 :006,006,005,007,007,007,023
4343 :007,183,185,250,040,006,150
4349 :006,119,097,169,147,032,055
4355 :210,255,169,008,141,015,033
4361 :144,169,032,162,000,157,161
4367 :242,030,232,224,242,208,169
4373 :248,169,032,162,000,157,021
4379 :228,031,232,224,022,208,204
4385 :248,162,000,189,162,017,043
4391 :157,242,150,232,224,242,006
4397 :208,245,162,000,189,148,229
4403 :018,157,228,151,232,224,037
4409 :022,208,245,169,000,141,074
4415 :060,003,169,064,141,046,034
4421 :145,169,141,141,020,003,176
4427 :169,017,141,021,003,169,083
4433 :192,141,046,145,076,170,083
4439 :018,162,000,189,242,150,080
4445 :041,015,201,000,240,009,087
4451 :168,200,152,201,008,208,012
4457 :002,169,005,157,242,150,062
4463 :232,224,242,208,230,162,129
4469 :000,189,228,151,041,015,229
4475 :168,200,152,201,008,208,036
4481 :002,169,005,157,228,151,073
4487 :232,224,022,208,234,096,127
4493 :238,060,003,172,060,003,165
4499 :192,010,208,008,032,088,173
4505 :017,160,000,141,060,003,022
4511 :076,191,234,007,007,007,169
4517 :007,007,007,007,007,007,207
4523 :007,000,000,007,007,007,199
4529 :007,007,007,007,007,007,219
4535 :007,006,006,006,006,006,220
4541 :006,006,006,006,007,000,220
4547 :000,007,006,006,006,006,226
4553 :006,006,006,006,006,005,236
4559 :005,005,005,005,005,005,237
4565 :005,006,007,007,007,007,252
4571 :006,005,005,005,005,005,250
4577 :005,005,005,007,007,007,005
4583 :007,007,007,007,005,006,014
4589 :006,006,006,006,006,005,016
4595 :007,007,007,007,007,007,029
```

```
4601 :007,006,006,006,006,006,030
4607 :006,007,005,006,006,006,035
4613 :006,006,006,005,007,006,041
4619 :006,006,006,006,006,006,047
4625 :006,006,006,006,006,007,054
4631 :005,005,005,005,005,005,053
4637 :005,005,007,006,006,006,064
4643 :006,006,006,005,005,005,068
4649 :005,005,006,007,005,005,074
4655 :005,005,005,005,005,005,077
4661 :007,006,005,005,005,005,086
4667 :005,005,005,005,005,005,089
4673 :006,007,007,007,007,007,106
4679 :007,007,007,007,007,006,112
4685 :005,005,005,005,005,007,109
4691 :007,007,007,005,006,007,122
4697 :007,007,007,007,007,007,131
4703 :007,007,007,006,005,007,134
4709 :007,007,007,007,007,007,143
4715 :007,005,006,006,006,006,143
4721 :006,006,006,006,006,006,149
4727 :006,006,005,007,007,007,157
4733 :007,006,006,007,007,005,163
4739 :006,006,006,006,006,006,167
4745 :006,006,006,006,006,006,173
4751 :005,007,007,006,006,006,180
4757 :006,007,007,005,005,005,184
4763 :005,005,005,005,005,005,185
4769 :005,005,005,005,005,007,193
4775 :007,006,006,169,010,141,250
4781 :065,003,169,008,141,066,113
4787 :003,169,014,141,067,003,064
4793 :169,004,141,069,003,169,228
4799 :002,141,070,003,169,008,072
4805 :141,071,003,169,015,141,225
4811 :014,144,169,170,141,013,086
4817 :144,169,050,141,128,022,095
4823 :162,000,169,037,157,000,228
4829 :030,232,224,242,208,248,125
4835 :162,000,189,000,128,157,095
4841 :000,028,232,224,000,208,157
4847 :245,189,000,129,157,000,191
4853 :029,232,224,000,208,245,159
4859 :189,131,023,157,000,029,012
4865 :232,224,064,208,245,169,119
4871 :255,141,005,144,169,000,209
4877 :141,072,003,141,073,003,190
4883 :032,118,019,162,000,169,007
4889 :037,157,000,030,232,224,193
```

```
4895 :242,208,248,162,000,189,056
4901 :086,023,142,062,003,170,011
4907 :169,046,157,000,030,169,102
4913 :001,157,000,150,174,062,081
4919 :003,232,224,015,208,231,200
4925 :162,000,189,079,023,157,159
4931 :000,030,169,004,157,000,171
4937 :150,232,224,007,208,240,110
4943 :169,003,141,074,003,032,245
4949 :078,021,162,005,169,005,013
4955 :141,061,003,170,169,033,156
4961 :157,212,031,169,232,141,015
4967 :075,003,169,003,141,076,058
4973 :003,169,000,141,078,003,247
4979 :076,195,019,162,006,160,221
4985 :006,024,032,240,255,162,072
4991 :000,189,099,023,032,210,168
4997 :255,232,224,011,208,245,028
5003 :162,008,160,001,024,032,014
5009 :240,255,162,000,189,110,077
5015 :023,032,210,255,232,224,103
5021 :021,208,245,169,000,141,173
5027 :072,003,141,073,003,173,116
5033 :017,145,041,032,240,249,125
5039 :032,010,022,032,010,022,047
5045 :032,010,022,032,010,022,053
5051 :173,017,145,041,032,208,035
5057 :249,096,162,000,142,068,142
5063 :003,032,065,020,174,068,049
5069 :003,032,025,022,032,117,180
5075 :022,162,170,142,013,144,096
5081 :032,243,019,173,135,003,054
5087 :201,000,240,003,032,196,127
5093 :020,238,068,003,174,068,032
5099 :003,224,003,208,213,076,194
5105 :195,019,173,141,002,041,044
5111 :001,201,001,240,054,165,141
5117 :198,201,000,240,047,169,084
5123 :000,133,198,173,119,002,116
5129 :201,133,208,006,169,100,058
5135 :141,128,022,096,201,134,225
5141 :208,006,169,050,141,128,211
5147 :022,096,201,135,208,006,183
5153 :169,025,141,128,022,096,102
5159 :201,136,208,006,169,010,001
5165 :141,128,022,096,096,120,136
5171 :032,159,255,173,141,002,045
5177 :041,001,201,001,240,244,017
5183 :088,096,169,127,141,034,206
```

```
5189 :145,173,032,145,162,255,213
5195 :142,034,145,041,128,208,005
5201 :021,174,061,003,169,032,029
5207 :157,212,031,232,224,010,185
5213 :144,002,162,009,142,061,101
5219 :003,076,221,020,173,017,097
5225 :145,041,016,208,021,174,198
5231 :061,003,169,032,157,212,233
5237 :031,202,224,255,208,002,015
5243 :162,000,142,061,003,076,055
5249 :221,020,173,135,003,201,114
5255 :001,240,082,173,017,145,025
5261 :041,032,208,075,173,061,219
5267 :003,010,010,010,141,133,198
5273 :003,162,001,142,135,003,087
5279 :174,133,003,024,105,008,094
5285 :141,134,003,189,255,022,141
5291 :170,189,242,030,201,032,011
5297 :240,008,169,000,141,135,102
5303 :003,076,109,021,169,035,084
5309 :157,242,030,142,132,003,127
5315 :096,174,132,003,169,032,033
5321 :157,242,030,238,133,003,236
5327 :174,133,003,236,134,003,122
5333 :144,209,162,000,142,135,237
5339 :003,096,189,212,031,201,183
5345 :032,208,008,169,033,157,064
5351 :212,031,076,131,020,206,139
5357 :074,003,032,078,021,032,221
5363 :210,022,174,061,003,169,114
5369 :032,157,212,031,169,033,115
5375 :162,005,157,212,031,142,196
5381 :061,003,173,065,003,170,224
5387 :169,032,157,008,031,173,069
5393 :066,003,170,169,032,157,102
5399 :008,031,173,067,003,170,219
5405 :169,032,157,008,031,169,083
5411 :010,141,065,003,169,008,175
5417 :141,066,003,169,014,141,063
5423 :067,003,169,004,141,069,244
5429 :003,169,002,141,070,003,185
5435 :169,008,141,071,003,173,112
5441 :074,003,201,000,240,001,072
5447 :096,032,118,019,076,022,178
5453 :019,162,007,169,037,236,195
5459 :074,003,240,007,157,014,066
5465 :030,202,076,082,021,224,212
5471 :000,240,010,169,036,157,195
5477 :014,030,202,224,000,208,011
```

```
5483 :248,096,032,243,021,142,121
5489 :062,003,189,065,003,170,093
5495 :169,038,157,008,031,160,170
5501 :255,140,011,144,032,241,180
5507 :022,136,192,080,208,245,246
5513 :169,000,141,011,144,032,122
5519 :132,022,174,062,003,189,213
5525 :065,003,170,169,032,157,233
5531 :008,031,174,062,003,165,086
5537 :162,041,007,024,105,001,245
5543 :157,069,003,024,105,005,018
5549 :157,065,003,238,078,003,205
5555 :173,078,003,201,025,240,131
5561 :001,096,206,128,022,169,039
5567 :000,141,078,003,173,128,202
5573 :022,201,007,208,003,238,108
5579 :128,022,120,169,234,141,249
5585 :149,017,141,150,017,088,003
5591 :032,229,022,032,010,022,050
5597 :032,010,022,032,010,022,093
5603 :032,235,022,120,169,208,245
5609 :141,149,017,169,008,141,090
5615 :150,017,088,096,138,056,016
5621 :233,022,205,065,003,208,213
5627 :003,162,000,096,205,066,015
5633 :003,208,003,162,001,096,218
5639 :162,002,096,162,000,160,077
5645 :000,200,192,000,208,251,096
5651 :232,224,000,208,246,096,001
5657 :142,063,003,189,065,003,234
5663 :170,169,032,157,008,031,086
5669 :174,063,003,173,061,003,002
5675 :221,069,003,176,009,222,231
5681 :065,003,222,069,003,076,231
5687 :063,022,254,065,003,254,204
5693 :069,003,189,065,003,024,158
5699 :105,022,170,224,220,144,184
5705 :020,165,162,041,007,174,130
5711 :063,003,024,105,001,157,176
5717 :069,003,024,105,005,170,205
5723 :076,104,022,189,008,031,009
5729 :201,033,208,003,076,236,086
5735 :020,169,034,157,008,031,010
5741 :138,174,063,003,157,065,197
5747 :003,096,162,000,160,000,024
5753 :200,192,000,208,251,232,180
5759 :224,100,208,246,096,173,150
5765 :072,003,024,105,010,141,232
5771 :072,003,144,003,238,073,160
```

```
5777 :003,162,000,160,006,024,244
5783 :032,240,255,173,073,003,159
5789 :174,072,003,032,205,221,096
5795 :173,072,003,205,075,003,182
5801 :208,038,173,073,003,205,101
5807 :076,003,208,030,173,075,228
5813 :003,024,105,232,141,075,249
5819 :003,173,076,003,105,003,038
5825 :141,076,003,173,074,003,151
5831 :201,007,240,006,238,074,197
5837 :003,032,078,021,096,162,085
5843 :255,142,013,144,032,241,014
5849 :022,202,224,080,208,245,174
5855 :169,170,141,013,144,096,188
5861 :162,220,142,013,144,096,238
5867 :162,170,142,013,144,096,194
5873 :140,077,003,160,000,200,053
5879 :192,000,208,251,172,077,123
5885 :003,096,204,182,161,139,014
5891 :118,096,075,054,205,183,222
5897 :162,140,119,097,076,054,145
5903 :206,184,163,141,119,098,158
5909 :076,054,207,185,163,142,080
5915 :120,098,076,054,208,186,001
5921 :164,142,120,098,076,054,175
5927 :209,187,165,143,121,099,195
5933 :077,055,210,188,166,143,116
5939 :121,099,077,055,211,189,035
5945 :166,144,122,099,077,055,208
5951 :212,190,167,145,122,100,231
5957 :077,055,213,191,168,146,151
5963 :123,101,078,055,019,003,198
5969 :015,018,005,037,048,027,231
5975 :048,077,099,118,142,167,226
5981 :181,197,210,221,234,240,096
5987 :159,084,082,069,078,067,126
5993 :072,070,073,082,069,030,245
5999 :080,082,069,083,083,037,033
6005 :066,085,084,084,079,078,081
6011 :037,084,079,037,080,076,004
6017 :065,089,255,255,255,255,023
6023 :255,255,255,255,231,231,081
6029 :231,195,066,066,000,126,057
6035 :126,126,102,000,000,102,091
6041 :126,126,255,255,126,126,143
6047 :126,126,255,255,024,024,201
6053 :024,060,189,189,255,129,243
6059 :000,000,000,000,000,000,171
6065 :000,000,126,165,219,165,084
6071 :165,219,165,126,013,013,116
```

Appendices

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 published in this book are written in a computer language called BASIC. BASIC is easy to learn and is built into all VIC-20s.

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 Appendix B, "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 reverse video, lowercase, and control characters? It's all explained in your VIC's manual, *Personal Computing on the VIC-20.*

A Quick Review

1. Type in the program a line at a time, in order. Press RETURN at the end of each line. Use the INST/DEL key to erase 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 Appendix B, "How to Type In Programs").

How to Type In Programs

Many of the programs in this book contain special control characters (cursor control, color keys, reverse characters, and so on). 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, VIC-20 program listings will contain words within 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, 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 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.

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:
{CLR}	SHIFT + CLR/HOME		{GRN}	CTRL + 6	
{HOME}	CLR/HOME		{BLU}	CTRL + 7	
{UP}	SHIFT + CRSR ↑		{YEL}	CTRL + 8	
{DOWN}	CRSR ↑		{F1}	f1	
{LEFT}	SHIFT + CRSR →		{F2}	SHIFT + f1	
{RIGHT}	CRSR →		{F3}	f3	
{RVS}	CTRL + 9		{F4}	SHIFT + f3	
{OFF}	CTRL + 0		{F5}	f5	
{BLK}	CTRL + 1		{F6}	SHIFT + f5	
{WHT}	CTRL + 2		{F7}	f7	
{RED}	CTRL + 3		{F8}	SHIFT + f7	
{CYN}	CTRL + 4		←	←	
{PUR}	CTRL + 5		↑	SHIFT + ↑	

The Automatic Proofreader

"The Automatic Proofreader" will help you type in program listings without typing mistakes. It is a short error-checking program that hides itself in memory. When activated, it lets you know immediately after typing a line from a program listing if you have made a mistake. Please read these instructions carefully before typing any programs in this book.

Preparing the Proofreader

1. Using the listing below, type in the Proofreader. Be very careful when entering the DATA statements—don't type an l instead of a 1, an O instead of a 0, extra commas, etc.

2. SAVE the Proofreader on tape or disk at least twice *before running it for the first time.* This is very important because the Proofreader erases part of itself when you first type RUN.

3. After the Proofreader is saved, type RUN. It will check itself for typing errors in the DATA statements and warn you if there's a mistake. Correct any errors and save the corrected version. Keep a copy in a safe place—you'll need it again and again, every time you enter a program from this book, *COMPUTE!'s Gazette* or *COMPUTE!* magazine.

4. When a correct version of the Proofreader is run, it activates itself. You are now ready to enter a program listing. If you press RUN/STOP–RESTORE, the Proofreader is disabled. To reactivate it, just type the command SYS 886 and press RETURN.

Using the Proofreader

All listings in this book have a *checksum number* appended to the end of each line, for example, ":rem 123". *Don't enter this statement when typing in a program.* It is just for your information. The rem makes the number harmless if someone does type it in. It will, however, use up memory if you enter it, and it will confuse the Proofreader, even if you entered the rest of the line correctly.

When you type in a line from a program listing and press

RETURN, the Proofreader displays a number at the top of your screen. *This checksum number must match the checksum number in the printed listing.* If it doesn't, it means you typed the line differently than the way it is listed. Immediately recheck your typing. Remember, don't type the rem statement with the checksum number; it is published only so you can check it against the number which appears on your screen.

The Proofreader is not picky with spaces. It will not notice extra spaces or missing ones. This is for your convenience, since spacing is generally not important. But occasionally proper spacing *is* important, so be extra careful with spaces, since the Proofreader will catch practically everything else that can go wrong.

There's another thing to watch out for: If you enter the line by using abbreviations for commands, the checksum will not match up. But there is a way to make the Proofreader check it. After entering the line, LIST it. This eliminates the abbreviations. Then move the cursor up to the line and press RETURN. It should now match the checksum. You can check whole groups of lines this way.

Special Tape SAVE Instructions

When you're done typing a listing, you must disable the Proofreader before saving the program on tape. Disable the Proofreader by pressing RUN/STOP–RESTORE (hold down the RUN/STOP key and sharply hit the RESTORE key). This procedure is not necessary for disk SAVEs, *but you must disable the Proofreader this way before a tape SAVE.*

SAVE to tape erases the Proofreader from memory, so you'll have to load and run it again if you want to type another listing. SAVE to disk does not erase the Proofreader.

Hidden Perils

The proofreader's home in the VIC is not a very safe haven. Since the cassette buffer is wiped out during tape operations, you need to disable the Proofreader with RUN/STOP–RESTORE before you save your program. This applies only to tape use. Disk users have nothing to worry about.

Not so for VIC owners with tape drives. What if you type in a program in several sittings? The next day, you come to your computer, load and run the Proofreader, then try to load

the partially completed program so you can add to it. But since the Proofreader is trying to hide in the cassette buffer, it's wiped out!

What you need is a way to load the Proofreader after you've loaded the partial program. The problem is, a tape LOAD to the buffer destroys what it's supposed to load.

After you've typed in and run the Proofreader, enter the following lines in direct mode (without line numbers) exactly as shown:

```
A$="PROOFREADER.T": B$="{10 SPACES}": FOR X = 1 TO 4:
  A$=A$+B$: NEXTX
FOR X = 886 TO 1018: A$=A$+CHR$ (PEEK(X)): NEXTX
OPEN 1, 1,1,A$:CLOSE1
```

After you enter the last line, you will be asked to press RECORD and PLAY on your cassette recorder. Put this program at the beginning of a new tape. This gives you a new way to load the Proofreader. Anytime you want to bring the Proofreader into memory without disturbing anything else, put the cassette in the tape drive, rewind, and enter:

```
OPEN1:CLOSE1
```

You can now start the Proofreader by typing SYS 886. To test this, PRINT PEEK (886) should return the number 173. If it does not, repeat the steps above, making sure that A$ ("PROOFREADER.T") contains 13 characters and that B$ contains ten spaces.

You can now reload the Proofreader into memory whenever LOAD or SAVE destroys it, restoring your personal typing helper.

Replace Original Proofreader

If you typed in the original version of the Proofreader from the October 1983 issue of *COMPUTE!'s Gazette*, you should replace it with the improved version below.

The Automatic Proofreader

```
100 PRINT"{CLR}PLEASE WAIT...":FORI=886TO1018:READ
    A:CK=CK+A:POKEI,A:NEXT
110 IF CK<>17539 THEN PRINT"{DOWN}YOU MADE AN ERRO
    R":PRINT"IN DATA STATEMENTS.":END
120 SYS886:PRINT"{CLR}{2 DOWN}PROOFREADER ACTIVATE
    D.":NEW
```

```
886 DATA 173,036,003,201,150,208
892 DATA 001,096,141,151,003,173
898 DATA 037,003,141,152,003,169
904 DATA 150,141,036,003,169,003
910 DATA 141,037,003,169,000,133
916 DATA 254,096,032,087,241,133
922 DATA 251,134,252,132,253,008
928 DATA 201,013,240,017,201,032
934 DATA 240,005,024,101,254,133
940 DATA 254,165,251,166,252,164
946 DATA 253,040,096,169,013,032
952 DATA 210,255,165,214,141,251
958 DATA 003,206,251,003,169,000
964 DATA 133,216,169,019,032,210
970 DATA 255,169,018,032,210,255
976 DATA 169,058,032,210,255,166
982 DATA 254,169,000,133,254,172
988 DATA 151,003,192,087,208,006
994 DATA 032,205,189,076,235,003
1000 DATA 032,205,221,169,032,032
1006 DATA 210,255,032,210,255,173
1012 DATA 251,003,133,214,076,173
1018 DATA 003
```

Tiny MLX for the VIC

Remember the last time you typed in the BASIC loader for a long machine language program? You typed in hundreds of numbers and commas. Even then, you couldn't be sure if you 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 again, rechecked your typing . . . . Frustrating, wasn't it?

Until now, though, that has been the best way to get machine language into your computer. Unless you happen to have 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 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 with the total. Some programmers make your task easier by including checksums every few lines, so you can locate your errors more easily.

Now, "Tiny MLX" comes to the rescue. MLX is a great way to enter all those long machine language programs with a minimum of fuss. Tiny MLX lets you enter the numbers from a special list that looks similar to 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 numbers on the wrong line. In short, Tiny MLX will make proofreading obsolete.

Using Tiny MLX

Type in and save Tiny MLX (you'll want to use it in the future). When you are ready to type in a machine language program, load Tiny MLX (be sure to follow any specific instructions given in the article for each program you plan to enter).

Enter the correct values on line 210; for CUT-OFF!, line 210 should be:

210 S=6063:E=7658

for Trenchfire, line 210 should be:

210 S=4352:E=6079

These numbers tell MLX where to put your machine language program.

Run Tiny MLX. You'll see a prompt corresponding to the starting address. The prompt is the correct line you are entering from the listing. It increases by six each time you enter a line. That's because each line has seven numbers—six actual data numbers plus a *checksum number*. The checksum verifies that you typed the previous six numbers correctly. If you enter any of the numbers incorrectly, the computer will buzz and prompt you to reenter the line. If you enter it correctly, a bell tone sounds and you continue to the next line.

MLX accepts only numbers as inputs. If you make a typing error, press the INST/DEL key; the entire number is deleted. You can press it as many times as necessary back to the start of the line. If you enter three-digit numbers as listed, the computer automatically prints the comma and goes 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 automatically appears in reverse video for emphasis.

Saving Your Program

Once you have completed entering all the data, Tiny MLX will prompt you for a filename for your program and ask you whether the SAVE should be to tape or disk.

That's all there is to it. You are now ready to load and use your new ML program. Be sure to follow the specific directions for loading and running each ML program you save using Tiny MLX.

Tiny MLX

```
100 POKE55,174:POKE56,23:CLR:POKE788,194    :rem 76
210 S=6063:E=7658                           :rem 136
300 PRINT"{CLR}";CHR$(14):AD=S              :rem 56
310 PRINTRIGHT$("0000"+MID$(STR$(AD),2),5);":";:FO
    RJ=1TO6                                 :rem 234
```

```
320 GOSUB570:IFN=-1THENJ=J+N:GOTO320        :rem 228
480 IFN<0THENPRINT:GOTO310                  :rem 168
490 A(J)=N:NEXTJ                            :rem 199
500 CKSUM=AD-INT(AD/256)*256:FORI=1TO6:CKSUM=(CKSU
    M+A(I))AND255:NEXT                      :rem 200
510 PRINTCHR$(18);:GOSUB570:PRINTCHR$(20)   :rem 234
515 IFN=CKSUMTHEN530                        :rem 255
520 PRINT:PRINT"LINE ENTERED WRONG":PRINT"RE-ENTER
    ":PRINT:GOSUB1000:GOTO310               :rem 129
530 GOSUB2000                               :rem 218
540 FORI=1TO6:POKEAD+I-1,A(I):NEXT          :rem 80
550 AD=AD+6:IFAD<ETHEN310                   :rem 212
560 GOTO710                                 :rem 108
570 N=0:Z=0                                 :rem 88
580 PRINT"{+}";                             :rem 79
581 GETA$:IFA$=""THEN581                    :rem 95
585 PRINTCHR$(20);:A=ASC(A$):IFA=13ORA=44ORA=32THE
    N670                                    :rem 229
590 IFA>128THENN=-A:RETURN                  :rem 137
600 IFA<>20 THEN 630                        :rem 10
610 GOSUB690:IFI=1ANDT=44THENN=-1:PRINT"{LEFT}
    {LEFT}";:GOTO690                        :rem 172
620 GOTO570                                 :rem 109
630 IFA<48ORA>57THEN580                     :rem 105
640 PRINTA$;:N=N*10+A-48                    :rem 106
650 IFN>255 THEN A=20:GOSUB1000:GOTO600     :rem 229
660 Z=Z+1:IFZ<3THEN580                      :rem 71
670 IFZ=0THENGOSUB1000:GOTO570              :rem 114
680 PRINT",";:RETURN                        :rem 240
690 S%=PEEK(209)+256*PEEK(210)+PEEK(211)    :rem 149
692 FORI=1TO3:T=PEEK(S%-I)                  :rem 68
695 IFT<>44ANDT<>58THENPOKES%-I,32:NEXT     :rem 205
700 PRINTLEFT$("{3 LEFT}",I-1);:RETURN      :rem 7
710 PRINT"{CLR}{RVS}*** SAVE ***{3 DOWN}"   :rem 236
720 INPUT"{DOWN} FILENAME";F$               :rem 228
730 PRINT:PRINT"{2 DOWN}{RVS}T{OFF}APE OR {RVS}D
    {OFF}ISK: (T/D)"                        :rem 228
740 GETA$:IFA$<>"T"ANDA$<>"D"THEN740        :rem 36
750 DV=1-7*(A$="D"):IFDV=8THENF$="0:"+F$    :rem 158
760 T$=F$:ZK=PEEK(53)+256*PEEK(54)-LEN(T$):POKE782
    ,ZK/256                                 :rem 3
762 POKE781,ZK-PEEK(782)*256:POKE780,LEN(T$):SYS65
    469                                     :rem 109
763 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 69
765 POKE254,S/256:POKE253,S-PEEK(254)*256:POKE780,
    253                                     :rem 12
766 POKE782,E/256:POKE781,E-PEEK(782)*256:SYS65496
                                            :rem 124
770 IF(PEEK(783)AND1)OR(ST AND191)THEN780   :rem 111
```

```
775 PRINT"{DOWN}DONE.":END                          :rem 106
780 PRINT"{DOWN}ERROR ON SAVE.{2 SPACES}TRY AGAIN.
    ":IFDV=1THEN720                                 :rem 171
781 OPEN15,8,15:INPUT#15,E1$,E2$:PRINTE1$;E2$:CLOS
    E15:GOTO720                                     :rem 103
782 GOTO720                                         :rem 115
845 POKE780,1:POKE781,DV:POKE782,1:SYS65466:rem 70
1000 REM BELL TONE                                  :rem 250
1001 POKE36878,15:POKE36874,190                     :rem 206
1002 FORW=1TO300:NEXTW                              :rem 117
1003 POKE36878,0:POKE36874,0:RETURN                  :rem 74
2000 REM BELL SOUND                                  :rem 78
2001 FORW=15TO0STEP-1:POKE36878,W:POKE36876,240:NE
     XTW                                             :rem 22
2002 POKE36876,0:RETURN                             :rem 119
```

Index

Notes

Notes

Notes

Notes

Notes

Notes

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!** Magazine. Use this form to order your subscription to **COMPUTE!**.

For Fastest Service,
Call Our **Toll-Free** US Order Line

800-334-0868

In NC call 919-275-9809

COMPUTE!
P.O. Box 5406
Greensboro, NC 27403

My Computer Is:
☐ Commodore 64 ☐ TI-99/4A ☐ Timex/Sinclair ☐ VIC-20 ☐ PET
☐ Radio Shack Color Computer ☐ Apple ☐ Atari ☐ Other _____
☐ Don't yet have one...

☐ $24 One Year US Subscription
☐ $45 Two Year US Subscription
☐ $65 Three Year US Subscription

Subscription rates outside the US:

☐ $30 Canada
☐ $42 Europe, Australia, New Zealand/Air Delivery
☐ $52 Middle East, North Africa, Central America/Air Mail
☐ $72 Elsewhere/Air Mail
☐ $30 International Surface Mail (lengthy, unreliable delivery)

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank; International Money Order, or charge card.

☐ Payment Enclosed ☐ VISA
☐ MasterCard ☐ American Express
Acct. No. _____ Expires ____/____

COMPUTE! Books

P.O. Box 5406 Greensboro, NC 27403

Ask your retailer for these **COMPUTE! Books**. If he or she has sold out, order directly from **COMPUTE!**

For Fastest Service
Call Our **TOLL FREE US Order Line**

800-334-0868

In NC call 919-275-9809

Quantity	Title	Price	Total
_____	Machine Language for Beginners	**$14.95***	_____
_____	Home Energy Applications	**$14.95***	_____
_____	COMPUTE!'s First Book of VIC	**$12.95***	_____
_____	COMPUTE!'s Second Book of VIC	**$12.95***	_____
_____	COMPUTE!'s First Book of VIC Games	**$12.95***	_____
_____	COMPUTE!'s First Book of 64	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari	**$12.95***	_____
_____	COMPUTE!'s Second Book of Atari	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari Graphics	**$12.95***	_____
_____	COMPUTE!'s First Book of Atari Games	**$12.95***	_____
_____	Mapping The Atari	**$14.95***	_____
_____	Inside Atari DOS	**$19.95***	_____
_____	The Atari BASIC Sourcebook	**$12.95***	_____
_____	Programmer's Reference Guide for TI-99/4A	**$14.95***	_____
_____	COMPUTE!'s First Book of TI Games	**$12.95***	_____
_____	Every Kid's First Book of Robots and Computers	**$ 4.95†**	_____
_____	The Beginner's Guide to Buying A Personal Computer	**$ 3.95†**	_____

Add $2 shipping and handling. Outside US add $5 air mail; $2 surface mail.

†Add $1 shipping and handling. Outside US add $5 air mail; $2 surface mail

Please add shipping and handling for each book ordered. _____

Total enclosed or to be charged. _____

All orders must be prepaid (money order, check, or charge). All payments must be in US funds. NC residents add 4% sales tax.

☐ Payment enclosed Please charge my: ☐ VISA ☐ MasterCard
☐ American Express Acc't. No. _____ Expires ___/___

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Allow 4-5 weeks for delivery.

If you've enjoyed the articles in this book, you'll find the same style and quality in every monthly issue of **COMPUTE!'s Gazette** for Commodore.

For Fastest Service
Call Our **Toll-Free** US Order Line
800-334-0868
In NC call 919-275-9809

COMPUTE!'s GAZETTE
P.O. Box 5406
Greensboro, NC 27403

My computer is:
☐ Commodore 64 (01) ☐ VIC-20 (02) ☐ Other (03) _____

☐ $20 One Year US Subscription
☐ $36 Two Year US Subscription
☐ $54 Three Year US Subscription

Subscription rates outside the US:

☐ $25 Canada
☐ $45 Air Mail Delivery
☐ $25 International Surface Mail

Name _____

Address _____

City _____ State _____ Zip _____

Country _____

Payment must be in US Funds drawn on a US Bank, International Money Order, or charge card. Your subscription will begin with the next available issue. Please allow 4–6 weeks for delivery of first issue. Subscription prices subject to change at any time.

☐ Payment Enclosed ☐ VISA
☐ MasterCard ☐ American Express

Acct. No. _____ Expires ____/____

The *COMPUTE!'s Gazette* subscriber list is made available to carefully screened organizations with a product or service which may be of interest to our readers. If you prefer not to receive such mailings, please check this box ☐.

How About a Game?

Your VIC-20 is a powerful machine. Its sophisticated operating system and easy-to-use BASIC give it tremendous computing capabilities. As you've undoubtedly discovered, it can be a real workhorse.

But computer users also like to have fun. The VIC's creators realized that and designed the VIC for games too—and now you've got *COMPUTE!'s Second Book of VIC Games* to make owning a VIC-20 even more fun.

Like its predecessor, the popular *First Book of VIC Games*, this book has something for almost everybody. It's divided into six chapters, offering everything from fundamental game-writing techniques to complete games in BASIC and machine language. All games are ready to type in and run. Here's just a small sampling of what you'll find inside:

- "Frantic Fisherman," where a peaceful afternoon at the lake turns into a battle with frenzied sharks and huge raindrops.
- "Poker," a sophisticated computer version of the classic card game.
- "Checkers" pits you against the computer on a field of black and red squares.
- "Tree Tutor," a captivating educational game that helps young children learn basic math operations.
- "Trenchfire," a machine language game that scrolls in 3-D.
- "Time Capsule," a text adventure that pits you against an advanced society centuries in the future.
- Articles on developing game concepts, creating game programs, and writing text adventures.
- And much more.

COMPUTE!'s Second Book of VIC Games includes some of the best games from recent issues of *COMPUTE!* and *COMPUTE!'s Gazette*, as well as several games and tutorials that have never before been published. All are of the high quality you expect from COMPUTE! Publications.

You may not be able to decide which of the games is your favorite. But whether you're a beginning VIC user or an advanced programmer, one thing is certain: *COMPUTE!'s Second Book of VIC Games* will make an exciting computer more exciting than ever before.

ISBN 0-942386-57-4

COMPUTE!'s Second Book of **VIC Games**

COMPUTE! Books