MIND MOVES

Strategy brain games for the Commodore® 64™

Tom Rugg Phil Feldman

dilithium Press

COMMODORE USERS TAKE NOTE . . .

Software containing the **Mind Moves Strategic Brain Games** is available on a 5¼" diskette! The software comes with a warranty (if the product is defective we will replace it free of charge) as well as a "Forever Guarantee" (in case of damage, simply return the diskette with $5 and we'll send you a new one).

QUIKWIT TRIVIA FANS TAKE NOTE . . .

Did you enjoy the entertainment and challenge of QUIKWIT? Much of the game's appeal is the well written trivia questions from the mind of Penguin Dave. We've corralled him into creating a brand new disk of questions in his legendary style. Each side of the FAMILY FUN/EXPERT disk contains over 1000 questions to probe your brain and is available immediately. This double-sided disk contains one side with questions suitable for the whole family including a section just for youngsters. The other side contains questions with the serious trivia fan in mind. A section of super toughies is included.

Interested?

You bet I am.

☐ Send me the **Mind Moves Strategic Brain Games** diskette for my Commodore 64 computer! Please send me the 5¼" diskette (0038-1015A) for only $19.95.

☐ Send me the FAMILY FUN/EXPERT 5¼" diskette (0054-1015A) for the Commodore 64 for only $12.95.

☐ I've enclosed my check in the amount of _____. Please rush my software to the address below.

☐ Please charge my VISA ____ M/C ____ in the amount of _____ and send my software to the address below.

Acct# _____ Exp. Date _____

Signature _____

Name _____

Address _____

City, State, Zip _____

(To expedite your order, phone **1 (800) 547-1842** and charge to your VISA or M/C)

☐ Please send me your free catalog entitled **BRAIN FOOD™**

PLACE
STAMP
HERE

dilithium Press
P.O. Box 606
Beaverton, OR 97075-0606

MIND MOVES

Strategy brain games
for the
Commodore 64

MIND MOVES

Strategy brain games for the Commodore 64

Tom Rugg Phil Feldman

dilithium Press
Beaverton, Oregon

10 9 8 7 6 5 4 3 2 1

ISBN 0-88056-054-1

dilithium Press
8285 S.W. Nimbus
Suite 151
Beaverton, Oregon 97005-6401

CONTENTS

ACKNOWLEDGEMENTS

Our thanks to the many people who have helped us during the course of creating this book. Thanks to Lisa Trumbo and Gary Swanson for their editorial guidance and near-infinite patience, to Maria Katinos and Mykonos P. Johnson for their help in inventing program names, to Whitney Rugg for her help in program testing, and to Leslie, Whitney, Gilda, and Michael for their tolerance, patience, and all-around support.

Special thanks to Dave "Bro" Feldman. His exhaustive research for QUIKWIT was most certainly a nontrivial pursuit.

PREFACE

To tell you what this book is about, we need to start by looking back at our previous books. In the past few years, we have written a group of books that has come to be known as the *32 BASIC Programs* series. These books have been very well received, both by reviewers and by the public. Included in the series are such lung-draining titles as *32 BASIC Programs for the Apple Computer*, *32 BASIC Programs for the Atari Computer*, and (after adding a few more programs) *More Than 32 BASIC Programs for the Commodore 64 Computer*. There are 12 books in the series.

Our intent with that series of books was to provide an alternative for computer owners who were previously faced with prices of $20 to $50 for every program they bought. At the time of the first book in the series (for the PET computer), no one else had published a book with a wide variety of programs specifically adapted for a particular computer. We decided to do it, and we're glad. Many others have followed since.

Among the hundreds of letters we have received from readers of *32 BASIC Programs*, we were pleased to see such a large number of them asking if we have another book of programs for their computers. (We confess we were also pleased to see how many of them said they especially liked our books because the programs in them actually *worked*, unlike those in some other books they had bought. But that's another story.)

Anyway, we decided it was time to do something about all of these fine people with such excellent judgment. In

writing a new book, we decided to keep the same general format, with minor changes. As far as we know, ours are still the only books of microcomputer programs with *complete* explanations of the programs. Because so many people liked this approach, we have kept it and, in fact, have made the explanations even more complete.

This book differs from the previous series in that all the programs here are in one category — games. There are several reasons for this. The main one is that games are what most of our readers have expressed the most interest in. Oh yes, they liked the practical applications, the educational programs, the mathematical programs, and so on, but a higher percentage of our correspondence was about games than anything else. Another reason is that *we* like games. Both of us have been game nuts since childhood — board games, card games, word games, athletic games, whatever. We were creating computer games for our own pleasure on large mainframe computer systems long before home computers became available.

The approach we are taking in *Mind Moves* is to include in it a *limited* number of *intelligent* computer games. By having a limited number (six), the book can be small enough that it is affordable. At the same time, we have been able to focus enough attention on each of the six programs that they are more sophisticated and enjoyable games than the more simplistic ones in *32 BASIC Programs*. We've still kept them short enough that it's feasible to type them into your computer yourself. However, if you're not a strong typist and you have a disk drive, you might be better off buying the book-software package that includes a disk containing all the programs and data files.

The other important aspect of this approach is that *Mind Moves* contains *intelligent* computer games. You can find shoot-em-up arcade games and guess-a-number-from-1-to-5 games all over the place, and we think they lose their appeal pretty fast. We prefer games that postpone brain atrophy, and therefore we have included a variety of

different types of games that make you think while you are having fun with them. Almost any game forces you to plan ahead and improve your analytical skills, but we've tried to develop ones that do it better than most of the mediocrity floating around the marketplace.

We like the way these games came out and we hope you do, too.

HOW TO USE THIS BOOK

Each chapter of this book presents a computer program that runs on a 64K Commodore 64 computer with either a disk drive or a Datasette (cassette) drive. Because of complications of using sequential data files with a cassette drive, QUIKWIT — our *only* program with that type of files, works reliably only with a disk drive. Although the programs use color and many use sound, all are playable in black and white and without sound, if necessary. The book is sold as a book alone, or as part of a book-software package that includes a disk with the programs and data files recorded on it. If you buy the book alone, you will have to type the programs yourself, and save them on disk or cassette. (If you find this tedious, you can buy the disk later by itself from the publisher, dilithium Press of Beaverton, Oregon.)

Each chapter of the book is made up of 10 sections, as shown below:

INTRODUCTION. Briefly gives you an overview of the game.

RULES. Explains the rules of the game.

HOW TO USE IT. Gives details of what happens when you run the program. Explains your options and the meanings of your responses.

SAMPLE RUN. Shows photographs of the screen during the course of the game.

PROGRAM LISTING. This is a *listing* or *printout* of the BASIC program that causes the computer to play the game.

EASY CHANGES. Shows you some very simple changes you can make to the program to make it work differently, if you wish.

PROGRAMMER'S NOTEBOOK. Explains the design of the program, in case you want to understand how it works.

MAIN ROUTINES. Shows the BASIC line numbers and gives a brief explanation of what each major portion of the program does.

MAIN VARIABLES. Explains what each of the key variables in the program is used for.

SUGGESTED PROJECTS. Provides a few ideas for major changes you might want to make to the program, if you are a tinkerer who understands BASIC pretty well.

RECOMMENDED PROCEDURE

Here is the way we recommend you try any of the programs if you do *not* have the disk with all the programs pre-recorded. (If you *do* have the disk, refer to Appendix A, the Software Backup and Loading Instructions.)

1. Read through the documentation that came with the computer to learn the fundamentals of how to use it. This will teach you to set up the computer, turn it on, enter a program, correct typing mistakes, and run a program. Be sure to read at least the first two chapters of the *Commodore 64 User's Guide.*
2. Select the game you would like to try first. As an example, let's say you select VERTIGO.

3. Turn the computer on, if you haven't already. If you have been using the computer since you turned it on, type the command NEW and press [RETURN] to eliminate any previous program that might be in memory. Be sure the computer is in uppercase/graphic mode unless the program listing is shown in lowercase. Pressing the [Commodore] key with the [SHIFT] key switches back and forth between uppercase/graphic mode and uppercase/lowercase mode, unless this feature has been disabled temporarily by a previous program.

4. Referring to the Program Listing section of the chapter, *carefully* type the program into the computer. Start with the top line (line 100) and type one line at a time, pressing [RETURN] after each line. If you notice *before* you press [RETURN] that you have made a typing mistake, backspace using the [DEL] key, complete the program line correctly, then press [RETURN]. If you notice after you have pressed [RETURN] that you have made a typing mistake, you have two choices for fixing the line. Either retype the entire line, or *edit* the line. To edit the line, use the [CRSR] movement keys to position the cursor over the area with the mistake, then retype the remainder of the line and press [RETURN]. Or, use the [INST/DEL] key to insert and delete characters as necessary. For details about line editing, refer to the first few pages of Chapter 3 of the *Commodore 64 User's Guide*.

5. Be sure to check your typing very carefully as you go. We can't stress this too strongly. Omitting a single keystroke or typing one character incorrectly is all it takes to prevent a program from working properly. Be especially careful with punctuation characters, such as periods, commas, semicolons, colons, and quotation marks. Also, be careful not to interchange the letter I and the number 1, or the letter O and the number 0. Don't confuse the *greater than* sign (>) with the *less*

than sign (<). The lowercase l (L) and the number 1 (one) look identical in the programs that are listed in lowercase. Be sure to type the right one! The rule to follow is: if there is any doubt from the context of the line you are typing whether the character is an l (L) or a 1 (one), it's a 1 (one). You'll only find l's (L's) in recognizable words and BASIC keywords (such as clr, left$, len, and val).

Another common typing mistake is entirely omitting a line from the program. Notice that all the program line numbers start with 100 and increase by 10 for each program line (110, 120, 130, etc.), except for line 9990 at the end. This is to make it easier for you to check back for missing lines.

We've tried to make the program listings as easy as possible for you to read and type. In addition to the predictable line numbers just mentioned, we have included spacing in program lines for easier readability. This takes up a few extra bytes of memory and very slightly slows down each program, but we believe it to be well worth it. Also, we have completely avoided the use of special control characters and graphics characters in the listings. Most published listings for the Commodore 64 require you to go through an agonizing translation process: you see one thing in the listing; you press a key on the keyboard that looks different; then you see a third thing on the screen. This is crazy. We have programmed around this by using BASIC's CHR$ function, as we did in the *32 BASIC Programs* books. We are amazed that more programmers don't have equal consideration for their readers. We subscribe to the philosophy of "what you see is what you type."

6. If you are not a risk-taker, you should periodically SAVE the partially typed program on disk or cassette. If you don't, Murphy's Law ("Anything that can go wrong, will.") dictates that you will have a power failure just before you finish entering the program (after an hour or two of typing). Then you'll have to start all over. Instead, SAVE the partial program every

20 or 30 minutes or so, using a different name each time (e.g., VERTIGO1, VERTIGO2, VERTIGO3). Then, after the power failure (which won't occur, now that you are prepared), you can simply LOAD the last partial copy you made, and continue your program entry.

7. When the entire program has been entered, SAVE another copy of it on disk or cassette. Then use the LIST command to review your typing, looking for the kinds of errors mentioned previously. If you find any errors and make corrections, SAVE another copy.

8. Finally, you are ready for the acid test. Type the RUN command and press [RETURN]. Is the same thing happening that is shown in the Sample Run photos and described in the How to Use It section of the chapter? If so, congratulations on your typing accuracy. Go on to step 10. If not, maintain your composure and go to step 9.

9. There are two fundamental types of errors. Let's call them fatal and nonfatal errors.

   Fatal error. A fatal error is one that causes the program to stop, generally with a screen display of an error message such as SYNTAX ERROR IN 270 or BAD SUBSCRIPT ERROR IN 1310. After the program stops, BASIC's normal READY message is displayed. This type of error indicates an error severe enough that BASIC could not continue with the program. The SYNTAX ERROR is the most common, and means that BASIC detected a typing error in the line indicated. For example, you would get a SYNTAX ERROR IN 140 of VERTIGO if you typed the word THAN or THE or THN instead of THEN. Lesson one: When you get a SYNTAX ERROR in a line, fix the typing error in that line, then go back to step 8.

   For any other error message, the course is not so clear. You should always check the line indicated to see if there is a typing error, but unfortunately, the error is often in another line. If you can't find an error in the

line indicated by the error message, check the last several lines before that line. If you find no errors there, go back and check the whole program — line by line and character by character. You can refer to the Error Message Appendix in the *Commodore 64 User's Guide*, but the explanations usually won't help you pin down the typing error unless you are pretty knowledgeable in both BASIC and the logic of the program you are running. Here are a few clues that may help:

OUT OF DATA ERROR	You probably have a typing error in one or more DATA statements in the program, perhaps leaving out a comma or typing a period instead of a comma. You also may have simply omitted a DATA statement.
BAD SUBSCRIPT ERROR	Look at the variable name(s) in parentheses in the line with the error. There may be a typing error in another line that deals with the same variable name.
RETURN WITHOUT GOSUB and NEXT WITHOUT FOR ERRORS	You may have typed an incorrect line number in a GOTO or GOSUB statement somewhere else in the program.

Nonfatal error. A nonfatal error is one that does not cause the program to end with an error message, but causes the program to work incorrectly. A simple example: line 1320 of VERTIGO has "GAME." in it, just like that — the word, followed by a period, all within quotation marks. Suppose you typed "GUM." instead. The program would still work without a fatal error, but the information displayed on the screen would be wrong — a nonfatal error. Other errors can be much trickier to find than this, but the best solution generally is to simply search through your typed version of the program, comparing it line by line and character by character with the printed listing in the book. Also see the following section entitled For Those with a Disk Drive.

Keep in mind that you may not be able to duplicate the Sample Run *exactly* because we use randomness in the games to prevent them from being exactly the same every time you play. You should be able to get the program to do the same *kinds* of things as the Sample Run, however. In either event (fatal or nonfatal error), find your typing error, fix it, and go back to step 8 (after using SAVE to make a new copy on disk or cassette, of course). If all else fails (you can't get the program to work right, and you can't find any typing errors after exhaustive checking and rechecking), refer to the Errata Offer in the back of the book.

10. Welcome to step 10! If you got here, it means things are working correctly so far. Keep running the program, playing the game as shown in the Sample Run photos and as explained in the How To Use It section of the chapter. If you find any errors as you continue testing the program, go back to step 9, fix the problem, and continue. If everything works properly, be sure to save this corrected version of the program on disk or tape, using the same name as in the book.
11. Now read the Easy Changes section of the chapter. Try any of these changes that look interesting. If you like the changed version better than the original, SAVE it on disk or cassette, too, but with a different name. Maybe VERTIGOC2 (change 2), for example.

FOR THOSE WITH A DISK DRIVE

If you have a disk drive, refer to Appendix D, which shows how you can use a program to get your computer to check your typing for you. This will be a lot easier than rechecking your typing manually, especially if you are not the world's greatest typist or proofreader.

1.

AEROJAM

"Bridge to sector traffic controller. Entering control territory. Do you read? Request permission to dock before early departure. Do you read?"

Yes, you're a futuristic air traffic controller in charge of spacecraft traveling through a busy flight corridor. You command these ships from your control console. Action is fast and furious. You must make accurate, quick decisions to get everyone through efficiently. The pace can be frenetic, and the slightest lapse can be costly. It's a pressure cooker environment demanding total concentration. Be sure you're well rested and your attention is riveted.

Give the manned spacecraft highest priority. Pay close attention to what's going on around the refueling dock; this area is most critical. And look out for space debris — that stuff can be most annoying.

Maybe this game will give you a deeper appreciation of the hectic job of the air traffic controller. At least it'll be something to think about the next time your plane is stuck on an airport runway for an hour.

RULES

The action takes place on a major east/west artery with traffic flowing in both directions. (It's not clear that compass directions like east and west have meaning here in

outer space but, what the heck, we claim poetic license.) Spacecraft enter from either end, must stop for refueling at the south-central dock, and exit at the opposite end from where they entered. There are two classes of ships — manned craft and drones. It's particularly important that you process the manned ships efficiently.

The playing field is subdivided into square sectors. Only one ship can be in any sector at a time. You command each ship's direction of movement. When you feel all are oriented properly, it's time to issue the movement command.

However, you have only a limited time to do this. The ever-present clock on your console ticks away. You must issue a movement command within a prescribed time, or one will be issued automatically. (After all, spacecraft can't just sit around for too long.)

New ships enter the arena regularly. The expected arrival time of the next ship is shown on your console so you can plan for it if you have the time to check.

After docking for refueling, each craft must leave in its intended direction through a proper corridor. So you have to position ships correctly for exit too. If things get too cluttered, you can zap a ship from your territory as a last resort (no telling where it goes). This is a costly option, but it sometimes is necessary to avoid worsening problems. It also leaves space debris where the ship was, rendering that square dangerous for awhile. This is particularly troublesome near the dock, since all ships must pass through that region.

Speaking of space debris, that junk can also appear randomly. It must be the funny weather in space.

Needless to say, a ship-to-ship collision is big trouble. We can hardly bear to talk about it. If a manned ship is involved, there goes your promotion.

HOW TO USE IT

AEROJAM begins with a display of its title screen. Hit a key and the screen changes to a program option menu. You use

this to select the version of the game you wish to play. Choose an easy, medium, or hard game scenario by pressing the 1, 2, or 3 key as shown. If you wish to end the program to return to our mundane earthly environment, press the 4 key.

The game begins with you in front of your control console. Let's discuss what you see. The territory under your command is a grid of squares. Each square can be occupied by only one spaceship at a time. These spacecraft are color-coded. The blue and green ships are eastward bound. They enter from the west and must exit to the east. In contrast, the red and yellow ships are westward bound. They, of course, enter from the east and must exit to the west. The yellow and green ships are manned, while the blue and red ships are drones. At the beginning of the game, four ships enter the playing arena. Their types and locations are selected randomly.

The refueling dock is at the south-central part of the grid. Six docking squares are alongside it — two on top and two on each side. These squares are recognizable by the two little docking latches in each. To dock, a ship must have its bow between the two docking latches.

Each ship enters the playing arena unfueled. This is represented by the large open area at the base of such ships. After a ship docks and refuels, this area fills up, and the ship is ready to exit.

All action is controlled from the keyboard with keys near the right side. Note the flashing white command ball. It appears in the upper left grid square when the game begins. Move this ball around the playing field to stop on each ship that you wish to command. You control this movement with the [CRSR] keys at the lower right on your keyboard. All four directional movements are possible. You use the [SHIFT] key in combination with a [CRSR] key for up or left — as usual. The [CRSR] keys repeat, so you can get continual movement by simply keeping them depressed. (Don't *you* get depressed.) With a little practice, you'll become quite adept at moving the command ball quickly to each desired location. Note that the ball wraps around to the

opposite edge of the field when it gets to the end of a row or column.

Each ship is pointed toward the north, east, south, or west. When a movement command is issued (more about this soon) the ship attempts to move to the adjacent square in the direction it's pointing. To change the orientation of a ship, move the command ball to it and press the [f1] key. This is the top key of the four function keys at the right of the keyboard. The ship rotates 90 degrees clockwise. Press [f1] more than once, if necessary, to achieve your desired orientation.

Should you feel the need to remove a ship from the playing arena, get the ball over it and press the [f7] key. This is the lowermost of the four function keys. You might want to remove a ship because things are too hectic or cluttered. This costs points, especially for manned ships, but may lead to a better score in the long run if you can process other ships more effectively. These lost ships do not return, they are simply dismissed with a burst of valuable energy to the next jurisdiction. Unfortunately, this leaves space debris lingering for awhile in the vacated square. You'll recognize space debris when you see it. Should another ship try to move into debris, it will meet with a similar fate.

As you may know, debris in space is getting to be quite a problem. Seems the environment is polluted everywhere. In space, the stuff can drift aimlessly, and you may find some debris appearing sporadically on the playing field. Be careful.

At last it's time to discuss the crucial movement command. You can issue one anytime by pressing the [f3] key. This is the second from the top of the four function keys. All game processes are updated when this command is issued. Each ship moves one square if it can. They cannot move through the northern or southern boundaries. Should two or more ships try to move into the same square, one survives, and the other is lost in space. (This is a baddie, try to avoid it. It'll cost you lots of points.) However, two ships can pass by each other. If two ships are in adjacent squares facing

each other, they will move into each other's squares without harm during the movement phase.

Ships at the dock refuel during this movement phase. The top two dock stations are quick, any ship there will be refueled immediately. The side stations sometimes take longer, perhaps requiring a second or third movement period. This time is controlled randomly. After each ship is refueled, turn it so it can leave the dock for bigger and better things.

For a ship to successfully exit the playing field, three things are required. The ship must be refueled, be pointed in the exit direction, and be at a proper exit square. A ship's proper exit square is one marked with a banner of the same color as the exiting ship. (Manned and drone spacecraft have different exit corridors.) If these conditions are not met, the ship just sits there. For each ship that exits properly, your score is boosted. The faster you get a ship through the playing arena, the larger will be your score. And don't forget that manned craft score higher than drones.

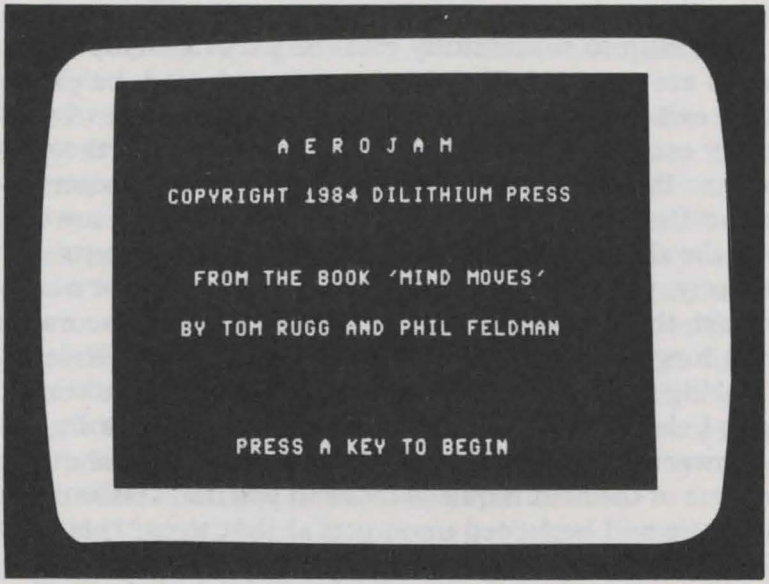
During the game, everything happens while the console clock ticks away. The current time (in seconds) is shown in the lower left corner of your video console. Also shown is the time of the next required move. If you don't issue one by then, one will be forced upon you at that time. This can be most unsettling if you're not prepared for it. (After all, air traffic controllers can't just sit still all day, especially in outer space.) Also shown is the time a space ship next is expected to arrive in your territory.

The area at the lower right of the screen tells you what's going on. You'll be told when moves are issued, when new ships arrive, when new debris is spotted, when collisions occur, etc. Your current score is also kept there so you can see how you're doing.

The arrival time of the last ship is indicated at the lower left. The game ends when you process all ships out of your territory after the last one arrives. Your final score and rank are then displayed. Remember them, and see if you can improve later.

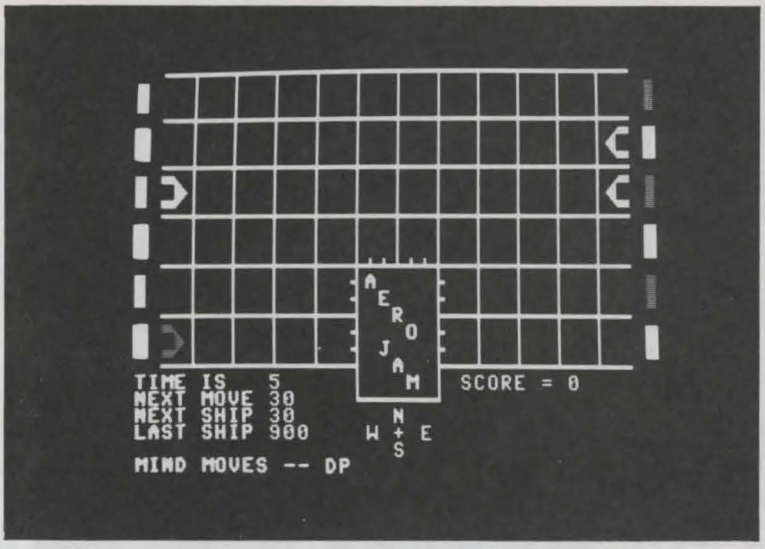
When you press a key as requested at the end of the game, the screen goes again to the original title display. You can start again or quit to resume earthly delights.

SAMPLE RUN

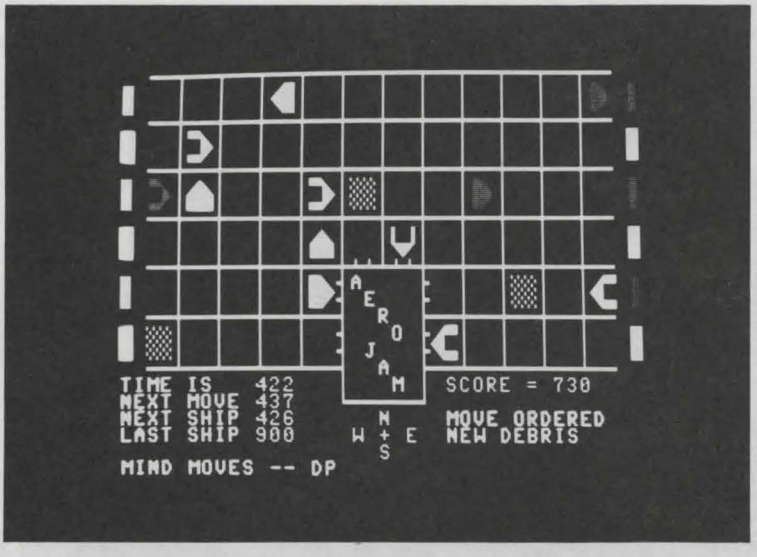


The title screen is displayed. A key must be pressed to begin.

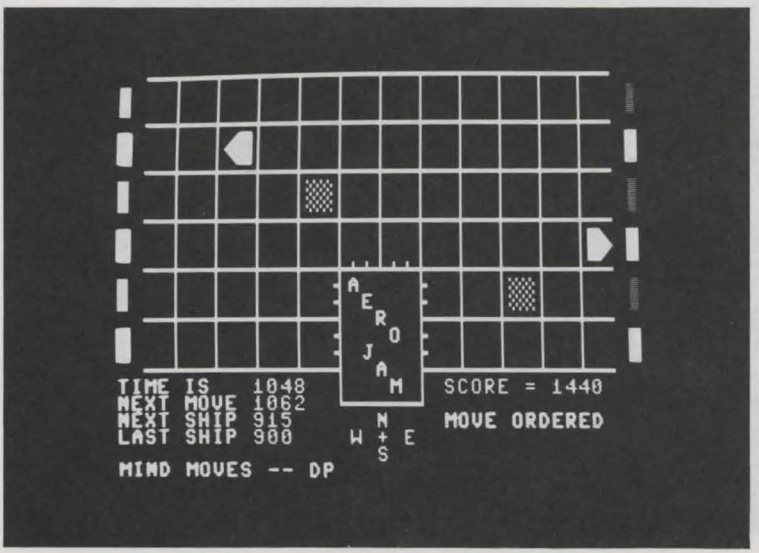
The option menu is shown. The player is about to press 2 to start a game of medium difficulty.



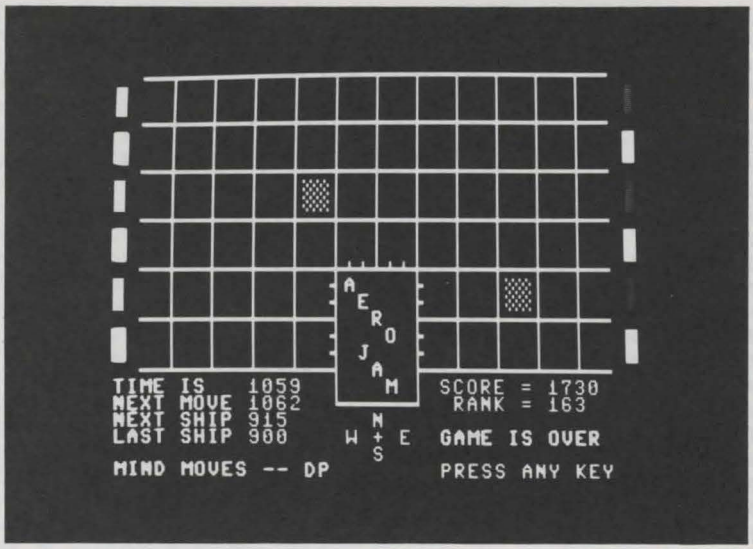
The AEROJAM display console is shown five seconds into the game. Four ships have entered the control territory for processing.



A typical situation later in the game is shown. Ships are at the dock refueling and others are coming and going. Near the center, a ship is in danger of colliding with space debris.



The game is almost over. Two ships have yet to exit to conclude the session.



The game ends with a display of the final score and rank. When a key is pressed, the title screen will reappear.

PROGRAM LISTING

```
100 REM: AEROJAM
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY PHIL FELDMAN AND TOM RUGG
130 POKE 53280,0:POKE 53281,0:PRINT CHR$(147)
140 GOTO 230
150 PRINT HM$;:IF CD=0 THEN 170
160 PRINT LEFT$(CD$,CD);
170 IF CR=0 THEN 190
180 PRINT LEFT$(CR$,CR);
190 RETURN
200 CR=XP(Q):CD=YP(Q):GOSUB 150:RETURN
210 CR=24:FOR CD=21 TO 24:GOSUB 150
220 PRINT LEFT$(SP$,15);:NEXT:RETURN
230 J=0:K=0:Q=0:M=0:W=0:N=0:ND=0:TF=TI:CD=0:CR=0
240 NM=0:NV=0:EV=0:NB=0:P=1:R$="A"
250 MX=30
260 DIM G(72),CG(MX),GD(MX),TV(MX),DK(MX),TE(MX)
270 DIM DM(MX),XP(72),YP(72)
280 AU$=CHR$(145):AR$=CHR$(29):AD$=CHR$(17):AL$=CHR$(157)
290 HM$=CHR$(19):Z$=CHR$(32):CC$=CHR$(152)
300 DIM CV$(4),T$(4,4)
310 YA$=CHR$(133):YB$=CHR$(136):YC$=CHR$(134)
320 RV$=CHR$(18):RF$=CHR$(146)
330 FF$=CHR$(157)+CHR$(157)+CHR$(17)
```

```
340 T$(1,1)=RV$+CHR$(169)+Z$+FF$+RF$+CHR$(127)+RV$+Z$+RF$
350 T$(1,2)=RV$+CHR$(169)+CHR$(127)+FF$+Z$+Z$+RF$
360 T$(1,3)=RV$+Z$+CHR$(127)+FF$+Z$+RF$+CHR$(169)
370 T$(1,4)=RV$+Z$+Z$+FF$+RF$+CHR$(127)+CHR$(169)
380 T$(2,1)=RV$+CHR$(169)+CHR$(162)+FF$+RF$+CHR$(127)
390 T$(2,1)=T$(2,1)+CHR$(162)
400 T$(2,2)=RV$+CHR$(169)+CHR$(127)+FF$+RF$+CHR$(161)+RV$
410 T$(2,2)=T$(2,2)+CHR$(161)+RF$
420 T$(2,3)=RV$+CHR$(162)+CHR$(127)+FF$+RF$+CHR$(162)
430 T$(2,3)=T$(2,3)+CHR$(169)
440 T$(2,4)=CHR$(161)+RV$+CHR$(161)+FF$+RF$+CHR$(127)
450 T$(2,4)=T$(2,4)+CHR$(169)
460 CV$(1)=CHR$(28):CV$(2)=CHR$(31)
470 CV$(3)=CHR$(158):CV$(4)=CHR$(30)
480 DB$=CHR$(166)+CHR$(166)+FF$+CHR$(166)+CHR$(166)
490 CL$=Z$+Z$+FF$+Z$+Z$
500 A$="":CR$=A$:CD$=A$:SP$=A$
510 FOR K=1 TO 40:CR$=CR$+CHR$(29):CD$=CD$+CHR$(17)
520 SP$=SP$+CHR$(32):NEXT
530 FOR J=1 TO 12:FOR K=0 TO 5
540 XP(J+K*12)=J*3-1
550 YP(J+K*12)=1+K*3:NEXT:NEXT
560 DIM PA(6),PB(6)
570 FOR K=1 TO 6:READ PA(K),PB(K):NEXT
580 DATA 1,12,13,24,25,36,37,48,49,60,61,72
590 DA=1:DB=0.5:DC=0.5
600 TF=60
610 MD=30:VD=30:BD=35
620 EV=900
630 GOSUB 3460
640 POKE 53280,15:POKE 53281,3:PRINT CHR$(147)
650 PRINT"A E R O J A M";SPC(3);"OPTION MENU":PRINT
660 PRINT"1 - PLAY EASY GAME":PRINT
670 PRINT"2 - PLAY MEDIUM GAME":PRINT
680 PRINT"3 - PLAY HARD GAME":PRINT
690 PRINT"4 - END PROGRAM"
700 PRINT:PRINT"PRESS 1, 2, 3, OR 4"
710 GET R$:IF LEN(R$)=0 THEN 710
720 IF R$<"1" OR R$>"4" THEN 710
730 Q=VAL(R$):ON Q GOTO 740,780,760,3580
740 VD=50:MD=40:BD=50
750 DA=1:DB=1:DC=0.7:GOTO 780
760 VD=20:MD=25:BD=25
770 DA=1:DB=0.5:DC=0.2:GOTO 780
780 MD=MD*TF:VD=VD*TF:BD=BD*TF:EV=EV*TF
790 NM=MD:NV=VD:NB=BD
800 POKE 53280,0:POKE 53281,0:PRINT CHR$(147);CC$;
810 FF$=CHR$(96)+CHR$(96)
820 C$=Z$+Z$:D$=C$:A$=C$:B$=C$:FOR K=1 TO 11
830 C$=C$+FF$+CHR$(178):D$=D$+FF$+CHR$(177)
840 A$=A$+FF$+CHR$(123):B$=B$+Z$+Z$+CHR$(125):NEXT
850 D$=D$+FF$:C$=C$+FF$:A$=A$+FF$:PRINT C$
860 FOR J=1 TO 5:PRINT B$:PRINT B$:PRINT A$:NEXT
870 PRINT B$:PRINT B$:PRINT D$:CD=12:CR=17:GOSUB 150
```

```
880 FOR K=1 TO 5:PRINT CHR$(177);:NEXT
890 CR=16:FOR CD=13 TO 18:GOSUB 150
900 PRINT CHR$(179)+LEFT$(SP$,5)+CHR$(171):NEXT
910 CD=18:CR=16:GOSUB 150
920 PRINT CHR$(179);LEFT$(SP$,5);CHR$(171)
930 CD=19:CR=16:GOSUB 150
940 PRINT CHR$(125);LEFT$(SP$,5);CHR$(125)
950 CD=20:CR=16:GOSUB 150
960 PRINT CHR$(173);:FOR K=1 TO 5:PRINT CHR$(96);:NEXT
970 PRINT CHR$(189):CR=0
980 FOR Q=1 TO 3:CD=6*Q-5:GOSUB 150:PRINT RV$;CV$(1);Z$
990 CR=38:GOSUB 150:PRINT RV$;CV$(2);Z$
1000 CD=CD+1:GOSUB 150:PRINT RV$;CV$(2);Z$
1010 CR=0:GOSUB 150:PRINT RV$;CV$(1);Z$:NEXT
1020 FOR Q=1 TO 3:CD=6*Q-2:GOSUB 150:PRINT RV$;CV$(3);Z$
1030 CR=38:GOSUB 150:PRINT RV$;CV$(4);Z$
1040 CD=CD+1:GOSUB 150:PRINT RV$;CV$(4);Z$
1050 CR=0:GOSUB 150:PRINT RV$;CV$(3);Z$:NEXT:PRINT CC$
1060 CD=19:CR=0:GOSUB 150:PRINT CHR$(159);"TIME IS"
1070 PRINT"NEXT MOVE";CC$;INT(NM/TF)
1080 PRINT CHR$(159);"NEXT SHIP";CC$;INT(NV/TF)
1090 PRINT CHR$(159);"LAST SHIP";CC$;INT(EV/TF)
1100 CD=24:CR=0:GOSUB 150
1110 PRINT CHR$(156);"MIND MOVES -- DP";
1120 FOR J=1 TO 4:CD=12+J:CR=16+J:GOSUB 150
1130 PRINT MID$("AERO",J,1);:NEXT
1140 FOR J=1 TO 3:CD=16+J:CR=17+J:GOSUB 150
1150 PRINT MID$("JAM",J,1);:NEXT:PRINT CC$;
1160 CD=21:CR=19:GOSUB 150:PRINT"N"
1170 CD=22:CR=17:GOSUB 150:PRINT"W";Z$;CHR$(43);Z$;"E"
1180 CD=23:CR=19:GOSUB 150:PRINT"S"
1190 CD=19:CR=24:GOSUB 150:PRINT"SCORE =":GOSUB 3370
1200 A$=LEFT$(CR$,2):B$=Z$+Z$+CHR$(29):C$=A$
1210 FOR K=1 TO 12:A$=A$+B$:NEXT
1220 FOR K=1 TO 5:C$=C$+B$:NEXT:C$=C$+LEFT$(CR$,6)
1230 FOR K=1 TO 5:C$=C$+B$:NEXT
1240 TI$="000000":REM 6 ZEROES
1250 FOR C=1 TO 4
1260 J=INT(6*RND(1))+1:IF RND(1)>0.5 THEN 1290
1270 IF G(PA(J))<>0 THEN 1260
1280 Q=PA(J):N=N+1:TV(N)=2+2*INT(2*RND(1)):DM(N)=3:GOTO 1310
1290 IF G(PB(J))<>0 THEN 1260
1300 Q=PB(J):N=N+1:TV(N)=1+2*INT(2*RND(1)):DM(N)=1
1310 DK(N)=2:CG(N)=Q:G(Q)=N:TE(N)=TI:GOSUB 200
1320 PRINT CV$(TV(N));T$(DK(N),DM(N));CC$;:NEXT
1330 GET R$:CD=19:CR=9:GOSUB 150:PRINT INT(TI/TF)
1340 J=1024+XP(P)+40*YP(P):K=PEEK(J):POKE J,81
1350 FOR D=1 TO 50:NEXT:POKE J,K
1360 IF TI>=NM THEN 1810
1370 IF TI>=NV AND TI<EV THEN 3140
1380 IF TI>=NB THEN 3020
1390 IF N=0 AND TI>EV THEN 3380
1400 IF R$=AR$ THEN 1480
1410 IF R$=AL$ THEN 1520
```

```
1420 IF R$=AU$ THEN 1560
1430 IF R$=AD$ THEN 1610
1440 IF R$=YA$ THEN 1660
1450 IF R$=YB$ THEN 1720
1460 IF R$=YC$ THEN 1830
1470 GOTO 1330
1480 P=P+1:IF P=54 THEN P=56:GOTO 1330
1490 IF P=66 THEN P=68:GOTO 1330
1500 IF P=73 THEN P=1:GOTO 1330
1510 GOTO 1330
1520 P=P-1:IF P=55 THEN P=53:GOTO 1330
1530 IF P=67 THEN P=65:GOTO 1330
1540 IF P=0 THEN P=72:GOTO 1330
1550 GOTO 1330
1560 P=P-12:IF P=-11 THEN P=72:GOTO 1330
1570 IF P<1 THEN P=P+71
1580 IF P=66 THEN P=42:GOTO 1330
1590 IF P=67 THEN P=43:GOTO 1330
1600 GOTO 1330
1610 P=P+12:IF P=84 THEN P=1:GOTO 1330
1620 IF P>72 THEN P=P-71:GOTO 1330
1630 IF P=54 THEN P=7:GOTO 1330
1640 IF P=55 THEN P=8:GOTO 1330
1650 GOTO 1330
1660 M=G(P):IF M<=0 THEN 1710
1670 Q=P:GOSUB 200
1680 DM(M)=DM(M)+1:IF DM(M)=5 THEN DM(M)=1
1690 Q=DK(M):IF Q=3 THEN Q=2
1700 PRINT CV$(TV(M));T$(Q,DM(M));CC$
1710 GOTO 1330
1720 J=G(P):IF J<=0 THEN 1800
1730 CR=24:CD=23:GOSUB 150:PRINT LEFT$(SP$,15);:GOSUB 150
1740 PRINT CHR$(31);"BYE BYE SHIP";CC$;
1750 GOSUB 2930
1760 Q=P:GOSUB 200:PRINT DB$
1770 W=W-30:IF TV(J)>2 THEN W=W-30
1780 G(Q)=-2:GOSUB 3330:GOSUB 3370
1790 ND=ND+1:GD(ND)=Q
1800 GOTO 1330
1810 GOSUB 210:CD=21:GOSUB 150
1820 PRINT CHR$(30);"MOVE FORCED";CC$;:GOTO 1850
1830 GOSUB 210:CD=21:GOSUB 150
1840 PRINT CHR$(158);"MOVE ORDERED";CC$
1850 FOR K=1 TO 49 STEP 24
1860 J=G(K):IF J<=0 THEN 1880
1870 IF TV(J)=1 AND DM(J)=1 AND DK(J)=1 THEN GOSUB 2890
1880 NEXT
1890 FOR K=13 TO 61 STEP 24:J=G(K):IF J<=0 THEN 1910
1900 IF TV(J)=3 AND DM(J)=1 AND DK(J)=1 THEN GOSUB 2890
1910 NEXT
1920 FOR K=12 TO 60 STEP 24:J=G(K):IF J<=0 THEN 1940
1930 IF TV(J)=2 AND DM(J)=3 AND DK(J)=1 THEN GOSUB 2890
1940 NEXT
1950 FOR K=24 TO 72 STEP 24:J=G(K):IF J<=0 THEN 1970
```

```
1960 IF TV(J)=4 AND DM(J)=3 AND DK(J)=1 THEN GOSUB 2890
1970 NEXT
1980 IF ND=0 THEN 2000
1990 FOR K=1 TO ND:G(GD(K))=-1:NEXT
2000 FOR K=2 TO 11:M=G(K):IF M<=0 THEN 2050
2010 ON DM(M) GOTO 2020,2050,2030,2040
2020 CG(M)=CG(M)-1:GOTO 2050
2030 CG(M)=CG(M)+1:GOTO 2050
2040 CG(M)=CG(M)+12
2050 NEXT
2060 FOR K=1 TO 61 STEP 12:M=G(K):IF M<=0 THEN 2120
2070 ON DM(M) GOTO 2120,2080,2100,2110
2080 IF K<>1 THEN CG(M)=CG(M)-12
2090 GOTO 2120
2100 CG(M)=CG(M)+1:GOTO 2120
2110 IF K<>61 THEN CG(M)=CG(M)+12
2120 NEXT
2130 FOR K=12 TO 72 STEP 12:M=G(K):IF M<=0 THEN 2190
2140 ON DM(M) GOTO 2150,2160,2190,2180
2150 CG(M)=CG(M)-1:GOTO 2190
2160 IF K<>12 THEN CG(M)=CG(M)-12
2170 GOTO 2190
2180 IF K<>72 THEN CG(M)=CG(M)+12
2190 NEXT
2200 FOR K=62 TO 71:IF K>=65 AND K<=68 THEN 2260
2210 M=G(K):IF M<=0 THEN 2260
2220 ON DM(M) GOTO 2230,2240,2250,2260
2230 CG(M)=CG(M)-1:GOTO 2260
2240 CG(M)=CG(M)-12:GOTO 2260
2250 CG(M)=CG(M)+1
2260 NEXT
2270 FOR K=14 TO 23:GOSUB 2960:NEXT
2280 FOR K=26 TO 35:GOSUB 2960:NEXT
2290 FOR K=38 TO 41:GOSUB 2960:NEXT
2300 FOR K=44 TO 47:GOSUB 2960:NEXT
2310 FOR K=50 TO 52:GOSUB 2960:NEXT
2320 FOR K=57 TO 59:GOSUB 2960:NEXT
2330 FOR K=42 TO 43:M=G(K):IF M<=0 THEN 2400
2340 ON DM(M) GOTO 2350,2360,2370,2380
2350 CG(M)=CG(M)-1:GOTO 2400
2360 CG(M)=CG(M)-12:GOTO 2400
2370 CG(M)=CG(M)+1:GOTO 2400
2380 IF RND(1)>DA THEN 2400
2390 DK(M)=1
2400 NEXT
2410 FOR K=53 TO 65 STEP 12:M=G(K):IF M<=0 THEN 2480
2420 ON DM(M) GOTO 2430,2440,2450,2470
2430 CG(M)=CG(M)-1:GOTO 2480
2440 CG(M)=CG(M)-12:GOTO 2480
2450 IF RND(1)>DB THEN 2480
2460 DK(M)=1:GOTO 2480
2470 IF K=53 THEN CG(M)=CG(M)+12
2480 NEXT
2490 FOR K=56 TO 68 STEP 12:M=G(K):IF M<=0 THEN 2560
```

```
2500 ON DM(M) GOTO 2510,2530,2540,2550
2510 IF RND(1)>DB THEN 2560
2520 DK(M)=1:GOTO 2560
2530 CG(M)=CG(M)-12:GOTO 2560
2540 CG(M)=CG(M)+1:GOTO 2560
2550 IF K=56 THEN CG(M)=CG(M)+12
2560 NEXT
2570 FOR K=1 TO 72:G(K)=0:NEXT
2580 IF ND=0 THEN 2600
2590 FOR K=1 TO ND:G(GD(K))=-1:NEXT
2600 CD=1:CR=0:GOSUB 150
2610 FOR K=1 TO 4:PRINT A$:PRINT A$:PRINT CHR$(29):NEXT
2620 FOR K=1 TO 2:PRINT C$:PRINT C$:PRINT CHR$(29):NEXT
2630 IF ND=0 THEN 2650
2640 FOR K=1 TO ND:Q=GD(K):GOSUB 200:PRINT DB$:NEXT
2650 J=0
2660 J=J+1
2670 IF J>N THEN 2800
2680 Q=CG(J):IF G(Q)=0 THEN 2780
2690 IF G(Q)>0 THEN 2750
2700 W=W-30:IF TV(J)>2 THEN W=W-30
2710 GOSUB 3350:G(Q)=-2:GOSUB 3370
2720 CR=24:CD=23:GOSUB 150:PRINT LEFT$(SP$,15);:GOSUB 150
2730 PRINT CHR$(28);"SHIP IN DEBRIS";CC$;
2740 GOSUB 2930:GOTO 2670
2750 W=W-75:GOSUB 3350:GOSUB 3370:CR=24:CD=23:GOSUB 150
2760 PRINT LEFT$(SP$,15);:GOSUB 150
2770 PRINT CHR$(159);"COLLISION";CC$;:GOSUB 2930:GOTO 2670
2780 GOSUB 200:PRINT CV$(TV(J));T$(DK(J),DM(J));CC$
2790 G(Q)=J:GOTO 2660
2800 K=0
2810 K=K+1
2820 IF K>ND THEN 2870
2830 Q=GD(K):IF G(Q)=-2 THEN 2810
2840 IF RND(1)>DC THEN 2810
2850 GD(K)=GD(ND):G(Q)=0:ND=ND-1
2860 GOSUB 200:PRINT CL$:GOTO 2820
2870 NM=TI+MD:CD=20:CR=9:GOSUB 150:PRINT INT(NM/TF)
2880 GOTO 1330
2890 Q=1:IF TV(J)=3 OR TV(J)=4 THEN Q=3
2900 W=W+INT(Q*2000*TF/(TI-TE(J)))
2910 Q=K:GOSUB 200:PRINT CL$
2920 G(Q)=0:GOSUB 3330:GOSUB 3370:RETURN
2930 FOR C=1 TO 4:POKE 53265,PEEK(53265) AND 239
2940 FOR D=1 TO 50:NEXT:POKE 53265,PEEK(53265) OR 16
2950 FOR D=1 TO 50:NEXT:NEXT:RETURN
2960 M=G(K):IF M<=0 THEN RETURN
2970 ON DM(M) GOTO 2980,2990,3000,3010
2980 CG(M)=CG(M)-1:RETURN
2990 CG(M)=CG(M)-12:RETURN
3000 CG(M)=CG(M)+1:RETURN
3010 CG(M)=CG(M)+12:RETURN
```

```
3020 TB=TI+60
3030 IF TI>TB THEN 3130
3040 Q=INT(72*RND(1))+1
3050 IF Q=54 OR Q=55 OR Q=66 OR Q=67 THEN 3030
3060 IF G(Q)<>0 THEN 3030
3070 IF ND>=MX THEN 3130
3080 ND=ND+1:GD(ND)=Q:G(Q)=-2
3090 GOSUB 200:PRINT DB$
3100 CR=24:CD=22:GOSUB 150:PRINT LEFT$(SP$,15);:GOSUB 150
3110 PRINT CHR$(129);"NEW DEBRIS";CC$
3120 NB=TI+BD
3130 GOTO 1330
3140 IF N>=MX THEN 3310
3150 J=INT(6*RND(1))+1:K=J:IF RND(1)>0.5 THEN 3220
3160 K=K+1:IF K=7 THEN K=1
3170 IF G(PA(K))=0 THEN 3200
3180 IF K=J THEN 3310
3190 GOTO 3160
3200 Q=PA(K):N=N+1:TV(N)=2+2*INT(2*RND(1)):DM(N)=3
3210 GOTO 3270
3220 K=K+1:IF K=7 THEN K=1
3230 IF G(PB(K))=0 THEN 3260
3240 IF K=J THEN 3310
3250 GOTO 3220
3260 Q=PB(K):N=N+1:TV(N)=1+2*INT(2*RND(1)):DM(N)=1
3270 DK(N)=2:CG(N)=Q:G(Q)=N:TE(N)=TI
3280 CR=24:CD=24:GOSUB 150:PRINT LEFT$(SP$,15);:GOSUB 150
3290 PRINT CHR$(156);"NEW SHIP";CC$;
3300 GOSUB 200:PRINT CV$(TV(N));T$(DK(N),DM(N));CC$
3310 NV=TI+VD:CD=21:CR=9:GOSUB 150:PRINT INT(NV/TF)
3320 GOTO 1330
3330 TV(J)=TV(N):TE(J)=TE(N):DK(J)=DK(N)
3340 CG(J)=CG(N):DM(J)=DM(N):G(CG(N))=J:N=N-1:RETURN
3350 TV(J)=TV(N):TE(J)=TE(N):DK(J)=DK(N):CG(J)=CG(N)
3360 DM(J)=DM(N):N=N-1:RETURN
3370 CD=19:CR=31:GOSUB 150:PRINT W;AL$;Z$;Z$;Z$;Z$:RETURN
3380 GOSUB 2930:GOSUB 2930:GOSUB 2930
3390 GOSUB 210:CD=20:CR=25:GOSUB 150
3400 PRINT"RANK =";INT(W*100*TF/TI)
3410 GET R$:IF LEN(R$)>0 THEN 3410
3420 CD=22:CR=24:GOSUB 150:PRINT CHR$(158);"GAME IS OVER"
3430 CD=24:GOSUB 150:PRINT CHR$(30);"PRESS ANY KEY";CC$;
3440 GET R$:IF LEN(R$)=0 THEN 3440
3450 CLR:GOTO 130
3460 GET R$:IF LEN(R$)>0 THEN 3460
3470 PRINT CHR$(142);CHR$(8);CHR$(147)
3480 POKE 53280,3:POKE 53281,5:PRINT CHR$(5):PRINT:PRINT
3490 PRINT TAB(12);"A E R O J A M":PRINT:PRINT
3500 PRINT TAB(4);"COPYRIGHT 1984 DILITHIUM PRESS"
3510 PRINT:PRINT:PRINT:PRINT
3520 PRINT TAB(6);"FROM THE BOOK 'MIND MOVES'":PRINT
3530 PRINT:PRINT TAB(5);"BY TOM RUGG AND PHIL FELDMAN"
```

```
3540 FOR J=1 TO 6:PRINT:NEXT
3550 PRINT TAB(9);"PRESS A KEY TO BEGIN"
3560 GET R$:IF LEN(R$)=0 THEN 3560
3570 Q=RND(-TI):RETURN
3580 PRINT CHR$(9);CHR$(154);CHR$(147);:CLR
3590 POKE 53280,14:POKE 53281,6
9990 END

READY.
```

EASY CHANGES

The following Easy Changes affect the difficulty of the game by modifying standard settings. If you make any of these changes, select main menu option 2 (for a medium game) when beginning play. The other menu options may cancel some of your changes.

1. The three keys used to command ship rotation, ship zapping, and move updating are controlled by the variables YA$, YB$, and YC$ respectively. They are set in line 310. You may not like the function keys if you find yourself hitting the wrong one occasionally. To use the R, Z, and M keys for these functions, change line 310 to:

   310 YA$="R":YB$="Z":YC$= "M"

2. Some random cutoffs are used in the game. The variables DA and DB are used to control the chance of a successful refueling at the top ports and side ports respectively. Set them to one for a sure thing and anywhere down to zero for no chance. In a similar manner, the variable DC controls the probability of debris dissipating. Set them all to a 35 percent chance with:

   590 DA=0.35:DB=0.35:DC=0.35

3. All times are computed in seconds (earth seconds of course.) However, you can control relativity and adjust the speed of the clock (slower or faster) by changing the value of TF in line 600. To make it run faster and produce a more frantic game, try:

```
600 TF=30
```

4. Fixed time increments are set between forced moves, the appearance of new ships, and the appearance of random debris. These are set in the variables MD, VD, and BD respectively. Changing them dramatically affects the game play. For example, lowering the arrival time between new ships will clutter the playing field quickly. Try various combinations to find one you prefer. Making the time between forced moves long, but the time between new ships short, will get a game version with lots to do but more time to do it. You might try:

```
610 MD=60:VD=20:BD=30
```

5. The game length is set by the variable EV in line 620. It's currently set to 900 seconds (that's 15 minutes.) If you want a shorter game, like 10 minutes, try:

```
620 EV=600
```

6. Four spaceships appear at the beginning of the game. You can adjust this number up or down by changing line 1250. Simply change the 4 at the end of the line to any number from 1 to 12. To get six initial spacecraft, line 1250 becomes:

```
1250 FOR C=1 TO 6
```

7. Currently, a maximum of 30 ships are allowed in play at one time. This is more than can really be controlled. But if you're a glutton for punishment and can't resist allowing for more, make MX in line 250 higher. Allow for 50 ships with:

```
250 MX=50
```

PROGRAMMER'S NOTEBOOK

AEROJAM imposes the task of keeping track of the status of all ships — where they are, who they are, where they're going, if they've refueled, etc. This is done through the TV, TE, DK, CG, and DM arrays. The Main Variables section explains what information each one contains. The variable N is the number of active ships, and it is bumped by one when each new ship enters. The arrays of ship attributes are indexed from 1 to N. When a ship leaves, N must be decreased by one, but we still want the arrays to be indexed from 1 to the new N. Let's say ship J leaves, where J is greater than 1 but less than N. We simply move all the values associated with the Nth ship into the Jth position. Then N can be decreased by one, and the arrays remain tight, correctly containing only information for the active ships.

The playing field is divided into a 12-by-6 grid. Internally, these squares are numbered from 1 to 72 moving left to right and top to bottom. Squares 54, 55, 66, and 67 are not used or referenced since they would be inside the dock. We often need to identify what is in each square. The array G does this. G is dimensioned from 1 to 72. Its value is 0 (zero) when nothing is in the square, −2 when debris has just been created there, −1 when old debris is still there, and a positive number when a ship is there. This positive number is the ship's index from 1 to N. Thus if we know a ship is in square 5, let's say, we can quickly get at vital information with a double array reference. For example, DM(G(5)) will tell us in what direction the ship in square 5 is pointing.

The CG array tells us which grid square ship J occupies, where J can be from 1 to N. Thus CG acts as the opposite of the G array. If CG(J)=5, then G(5)=J. The GD array similarly tells us the grid location of debris, where the number of active debris squares runs from 1 to ND.

The program makes frequent use of the computer's internal clock timer with the system variables TI and TI$. At the beginning of the game, line 1240 resets the timer to zero. From then on, the variable TI tells us the current time. This time is measured in *jiffies*, where each jiffy is 1/60 of a second. At any time in the game, the value of TI/60 will be the number of seconds since the game began.

This is used to keep the console clock accurately timed. The variables NM, NV, and NB are used to trigger certain events at future times. For example, NV holds the time when the next ship is scheduled to appear. If we frequently monitor the value of TI, as soon as it gets larger than NV we know it's time to bring in the next ship. Lines 1360-1380 do this monitoring.

MAIN ROUTINES

130-140	Blanks screen and begins variable initialization.
150-190	Subroutine to position cursor CD rows down and CR columns to the right from the upper left corner.
200	Positions the command ball at grid square Q.
210-220	Clears message area in the lower right corner.
230-620	Variable initialization.
630	Calls the subroutine to display the title screen.
640-730	Displays option menu, branches to player's choice.
740-750	Sets variables for an easy game.
760-770	Sets variables for a hard game.
780-790	Calibrates the timing variables.
800-1190	Draws the playing board.
1200-1230	Defines strings used to blank out the play grid.
1240	Initializes the time counter.
1250-1320	Selects and displays the initial four spacecraft.
1330-1470	Draws command ball and detects user's requests.
1480-1650	Processes requests to move the command ball.

1660-1710	Rotates ship clockwise.
1720-1800	Sends ship to never-never land.
1810-2880	Processes a move request — updates collisions, ship positions, refueling, debris, etc.
2890-2920	Subroutine to adjust score upon a successful ship exit.
2930-2960	Subroutine to flash the screen.
2970-3010	Subroutine to update ship grid locations.
3020-3130	Brings in new debris randomly.
3140-3320	Brings in new ship.
3330-3360	Subroutines to update arrays when a ship leaves the field of play.
3370	Subroutine to display the player's score.
3380-3450	Updates the scoreboard at the game's conclusion.
3460-3570	Subroutine to display the introductory title.
3580-9990	Resets screen colors and ends the game.

MAIN VARIABLES

N	Number of active ships.
ND	Number of squares containing space debris.
TF	Time calibration constant.
NM, NV, NB	Time of next automatic move, new ship arrival, and new debris.
EV	Time for last ship to arrive.
MD, VD, BD	Time increments to update NM, NV, and NB.
P	Grid square of command ball.
MX	Maximum number of ships allowed in play at once.
W	Player's current score.
DA, DB, DC	Random value cutoffs for successful docking at top ports and side ports and for debris removal.
CD, CR	Values to position cursor down and right.
C, J, K, Q, M	Loop, index, and work variables.
XP, YP	Arrays of X and Y positions of each grid square.
G	Array of what's in each grid square (neg = debris, 0 = nothing, pos = ship).

CG	Array of each ship's current grid square.
TV	Array of each ship's classification (1 = westbound drone, 2 = eastbound drone, 3 = westbound manned, 4 = eastbound manned).
DK	Array of each ship's fueling status (1 = fueled, 2 = needs fuel).
TE	Array of each ship's time of entry.
DM	Array of each ship's direction of movement (1 = left, 2 = up, 3 = right, 4 = down).
GD	Array of each debris' grid square.
PA, PB	Arrays of possible entry squares for eastbound and westbound ships.
HM$	String to home the cursor.
CR$, CD$, SP$	Strings of 40 cursor rights, 40 cursor downs, and 40 blank spaces.
R$	Player input string.
AU$, AR$, AD$, AL$	Strings to request command ball movement up, right, down, and left.
Z$	String of one blank character.
CC$	String of default character color for printing.
YA$, YB$, YC$	Strings to request ship turning, ship zapping, and movement updates.
RV$, RF$	Strings to turn reverse characters on and off.
A$, B$, C$, D$, FF$	Work strings.
DB$	String to draw debris.
CL$	String to blank out a grid square.
CV$	String array to display color for each ship classification.
T$	String array to display each ship's classification and orientation.

SUGGESTED PROJECTS

1. Create a multiplayer version of the game. Perhaps two can play, one controlling the westbound ships and the other the eastbound ships. Each player's decisions will affect the task of the other.

2. Implement a feature to prescibe a flight plan for each ship at its entry time. This would be its route to the dock and exit path with set times for each event. Then the ship would automatically move itself unless you updated its plan as changing conditions might warrant.
3. Add a detect feature that warns you when a ship-to-ship or debris collision is imminent.

2.

ESCAPADE

If you have been following the computer gaming scene, you know what an adventure game is. Most adventure games are extremely complex simulations in which you, the player, try to explore a complicated area (maybe a labyrinth of caves, rooms, or dungeons) looking for treasures while trying to avoid and/or kill monsters. For the player's moves, the games generally accept simple English commands, such as "go north" and "throw bomb." Newer versions of these games keep coming up with larger vocabularies, requiring more obscure commands to overcome the obstacles and monsters. These games give the hard-core adventure freaks new challenges, but the beginning adventurer becomes hopelessly lost while trying to figure out incredibly complex games. How is a beginner going to figure out that he needs to juggle the enchanted pineapples in front of the yellow dwarf in order to scare away the poisonous wombat?

Don't despair, inexpert adventurers! ESCAPADE is for you. It has a small vocabulary and a straightforward scenario. You can get an enjoyable taste of what adventure games are like, then move on to the hard-core stuff after you become comfortable with this game.

RULES

In this game, you start from the front yard of a mysterious old mansion. It is next to a cemetery, although this has nothing to do with the game other than to indicate your isolation during the adventure, and to add credibility to the legends that the house is haunted. In addition, there are stories that numerous gangs of crooks have used the house as a hideout over the years, but they always have disappeared without a trace.

Throwing caution to the wind, you have decided for once in your life to go on a dangerous and exciting escapade. You are going to explore this deserted old house, trying to visit all the rooms. And, you are going to look for any loot left behind by the crooks. If you find any, of course, you'll carry it with you back out to the front yard, where you will drop it while you go inside to look for more.

That's the scenario and the object of the game. Details of exactly how to do these things follow.

HOW TO USE IT

If you are one of those people who likes to try to assemble the new bicycle (or lawn chair or computer system) without reading the instructions, then stop reading right now and begin playing the game. The rest of you, read on.

ESCAPADE starts by displaying its title screen and copyright notice, and waiting for you to press a key to begin. After a brief delay, the program displays its option menu. Choose option 1 to play the standard game of ESCAPADE. This game will always be the same, so you can play it numerous times until you have mastered it. The creatures, weapons, valuables, rooms, and doors are always in the same places, and the same weapons defeat the same creatures each time.

Option 2 gives you a random game. This means that some things change from game to game, although many things

stay the same. As currently implemented, the random option causes the weapons and valuables to be randomly placed (within limitations) in the house. Other things (creatures, rooms, doors) stay the same.

Option 3 displays a screen of brief instructions, and option 4 ends the program.

As we said earlier, this game uses a pretty simple command format. Enter your commands on your computer keyboard as either one or two words, pressing [RETURN] to conclude each command entry. Any words beyond two are ignored. For two-word commands, the first word is a verb and the second is either a noun or a direction. Some examples: GET LANTERN, USE WATER, DROP RUBY, GO NORTH.

Single-word commands are either simply a direction (NORTH, SOUTH, EAST, WEST, UP, DOWN) or one of the verbs that requires no object, which are: SCORE, LOOK (or INVENTORY), QUIT, and HELP.

If you prefer, you can enter the first letter of any of the directions (N, S, E, W, U, D) rather than the whole word. In summary, to move north you can choose from these four commands:

GO NORTH, GO N, NORTH, N.

For an object that is described with a phrase rather than a single word, use the last word of the phrase as the second word in your command. For example, to pick up the ring of keys, say GET KEYS instead of GET RING or GET RING OF KEYS. For objects with names longer than five characters, you need only enter the first five characters. You can say TAKE LANTE instead of TAKE LANTERN if you like.

While you wander through the house, you are carrying a bag with you. When you get (or take) something, it goes into your bag. Like all other bags we know of, this bag is not infinitely large. It has room for no more than five objects.

Scoring for the game is based on two factors — the point values of the valuables that you have removed from the house and dropped in the front yard, and the number of

rooms you have visited. There are 29 rooms and a total of 180 points for the five valuables, so the highest score you can achieve is 209 points. Use the SCORE command at any time to see how you are doing so far.

To see a list of commands you can use, enter the HELP command. To end the game, say QUIT. Ideally, you do this after dropping all the valuables in the front yard, but you can do it at any time.

You may find it helpful to draw a map as you explore the rooms, showing where objects, monsters, and doorways are. This will make it easier to improve your score next time. Soon you'll score 209 points and be rated as an Elite Escapader.

SAMPLE RUN

```
E S C A P A D E  FROM 'MIND MOVES'

OPTIONS:
 1 - PLAY STANDARD GAME.
 2 - PLAY RANDOM GAME.
 3 - INSTRUCTIONS FOR PLAYING.
 4 - END PROGRAM.

PRESS 1, 2, 3, OR 4
1
```

The program displays its option menu, and the player picks option 1 to play the standard game.

```
E S C A P A D E  FROM 'MIND MOVES'

YOU ARE STANDING IN THE FRONT YARD IN
FRONT OF A MYSTERIOUS OLD MANSION NEAR
A CEMETERY.  NO ONE IS AROUND FOR
MILES.  YOU'VE HEARD LEGENDS ABOUT
THIS PLACE -- THAT IT'S HAUNTED, AND
THAT CROOKS USED IT AS A HIDEOUT UNTIL
THEY MYSTERIOUSLY DISAPPEARED.

YOU'VE DECIDED TO EXPLORE THIS PLACE.
MAYBE YOU CAN FIND SOME LOOT LEFT
BEHIND BY THE CROOKS.  EVEN IF YOU
DON'T, THIS WILL BE AN EXCITING . . .

        E S C A P A D E!

YOU ARE IN THE FRONT YARD
IN FRONT OF A FRIGHTENING OLD HOUSE

YOU CAN GO: NORTH

COMMAND? N
```

After reading the introduction, the player types N to go north.

```
YOU CAN GO: NORTH

COMMAND? N

YOU ARE IN THE FRONT PORCH
JUST OUTSIDE THE FRONT DOOR

THERE IS A RING OF KEYS HERE.
THERE IS A LANTERN HERE.

YOU CAN GO: NORTH SOUTH

COMMAND? GET LANTERN

GOT IT -- IT'S NOW IN YOUR BAG

YOU ARE IN THE FRONT PORCH

THERE IS A RING OF KEYS HERE.

YOU CAN GO: NORTH SOUTH

COMMAND? N
```

This moves the player from the front yard to the front porch, where he or she decides to pick up the lantern before continuing north.

PROGRAM LISTING

```
100 REM: ESCAPADE
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY TOM RUGG AND PHIL FELDMAN
130 GOSUB 2910:GOSUB 1990:GOSUB 1480
140 PRINT CHR$(147)
150 PRINT TT$;SPC(2);"FROM 'MIND MOVES'"
160 GOSUB 1330
170 GOSUB 210:GOSUB 260:GOSUB 330
180 GOSUB 430:GOSUB 640
190 IF M=5 THEN 180
200 GOTO 170
210 PRINT
220 PRINT"YOU ARE IN THE";SPC(1);RS$(RN)
230 IF TR(RN)<>0 THEN 250
240 TR(RN)=1:PRINT RL$(RN)
250 RETURN
260 PRINT
270 FOR J=1 TO NW:W=TW(J):GOSUB 320
280 IF W<>RN THEN 300
290 PRINT"THERE IS A ";DW$(J);" HERE."
300 NEXT
310 RETURN
320 T=INT(W/100):W=W-T*100:RETURN
330 FOR J=1 TO NC:W=TC(J):GOSUB 320
340 IF W<>RN THEN 370
350 PRINT:PRINT"THERE IS A";SPC(1);DC$(J)
360 PRINT"BLOCKING THE DOOR GOING ";CM$(T+7);"!"
370 NEXT
380 PRINT:PRINT"YOU CAN GO:";
390 FOR J=0 TO 5:IF TM(RN,J)=0 OR TM(RN,J)=RN THEN 410
400 PRINT SPC(1);CM$(J+7);
410 NEXT:PRINT
420 RETURN
430 R$="":PRINT
440 INPUT"COMMAND";R$:IF LEN(R$)=0 THEN 440
450 R$=R$+CHR$(32)
460 L=LEN(R$):PF=1:WA$="":WB$="":FOR J=1 TO L
470 IF ASC(MID$(R$,J,1))=32 THEN 490
480 NEXT:GOTO 530
490 IF J=PF THEN PF=PF+1:GOTO 480
500 IF LEN(WA$)=0 THEN WA$=MID$(R$,PF,J-1):GOTO 520
510 IF LEN(WB$)=0 THEN WB$=MID$(R$,PF,J-PF)
520 PF=J+1:GOTO 480
530 IF LEN(WA$)=0 THEN 610
540 WA$=LEFT$(WA$,5):WB$=LEFT$(WB$,5)
550 VA=0:VB=0:FOR J=1 TO NA:IF VA<>0 AND VB<>0 THEN 580
560 IF CM$(J)=WA$ THEN VA=CM(J)
570 IF CM$(J)=WB$ THEN VB=CM(J)
580 NEXT:IF VA=0 THEN 610
590 IF VA>10000 THEN 610
600 RETURN
```

```
610 M=1:GOSUB 620:GOTO 430
620 PRINT:PRINT M$(M):IF M=4 THEN PRINT DC$(J)
630 PRINT:RETURN
640 W=VA:GOSUB 320
650 IF W=2 THEN VA=VB:W=1:GOTO 640
660 ON W GOSUB 680,680,770,870,1010,1070,1160,1280
670 RETURN
680 FL=T:FR=W:M=0
690 IF TM(RN,FL)=0 THEN M=2:GOSUB 620:GOTO 760
700 FOR J=1 TO NC:IF TC(J)<0 THEN 740
710 W=TC(J):GOSUB 320:IF RN<>W THEN 740
720 IF FL<>T THEN 740
730 M=4:GOSUB 620
740 NEXT:IF M=4 THEN 760
750 RN=TM(RN,FL):MN=MN+1
760 RETURN
770 W=VB:IF W=0 THEN M=5:GOTO 860
780 GOSUB 320
790 T=T-100:IF T<1 THEN M=5:GOTO 860
800 W=TW(T):J=T:GOSUB 320
810 IF W=0 THEN M=3:GOTO 860
820 IF W<>RN THEN M=18:GOTO 860
830 IF NB>=5 THEN M=22:GOTO 860
840 TW(J)=T*100
850 M=19:NB=NB+1
860 GOSUB 620:RETURN
870 W=VB:IF W=0 THEN M=5:GOTO 1000
880 GOSUB 320
890 T=T-100:IF T<1 THEN M=5:GOTO 1000
900 WR=W:W=TW(T):WL=T:GOSUB 320
910 IF W<>0 THEN M=20:GOTO 1000
920 FL=T:FR=W:IF WL>NC THEN M=21:GOTO 990
930 K=0:FOR J=1 TO NC:W=TC(J):GOSUB 320
940 IF W=RN THEN K=J
950 NEXT
960 IF K=0 THEN M=21:GOTO 990
970 IF K<>FL THEN M=WR:GOTO 990
980 M=WL+11:TC(WL)=0
990 TW(WL)=FL*100+RN:NB=NB-1
1000 GOSUB 620:RETURN
1010 K=0:FOR J=1 TO NW:W=TW(J):GOSUB 320
1020 IF W<>1 THEN 1040
1030 IF T>10 AND T<90 THEN K=K+T
1040 NEXT:FOR J=1 TO NR:K=K+TR(J):NEXT
1050 PRINT:PRINT"SCORE =";K;"POINTS.":PRINT
1060 RETURN
1070 TR(RN)=0
1080 PRINT:PRINT"YOU ARE CARRYING:"
1090 K=0:FOR J=1 TO NW:W=TW(J):GOSUB 320
1100 IF W<>0 THEN 1130
1110 IF POS(0)+LEN(DW$(J))>38 THEN PRINT
1120 K=K+1:PRINT DW$(J);SPC(2);
1130 NEXT:IF K=0 THEN PRINT"NOTHING";
1140 PRINT:PRINT
```

```
1150 RETURN
1160 GOSUB 1010:K=0
1170 FOR J=1 TO NR:K=K+TR(J):NEXT
1180 PRINT"YOU VISITED";K;"OF";NR;"ROOMS."
1190 PRINT:PRINT"SCORING SCALE:":PRINT
1200 PRINT"  1- 50 = BEGINNING BOZO"
1210 PRINT" 51-100 = HALF-WIT HAMBURGER"
1220 PRINT"101-150 = MID-LEVEL MEATBALL"
1230 PRINT"151-200 = TREMENDOUS TURKEY"
1240 PRINT"  200   = ELITE ESCAPADER"
1250 PRINT:PRINT"PRESS A KEY TO GO ON"
1260 GET R$:IF LEN(R$)=0 THEN 1260
1270 CLR:GOTO 130
1280 PRINT:PRINT"AVAILABLE VERBS ARE:":PRINT
1290 PRINT"GO, GET (TAKE), USE (DROP, THROW),"
1300 PRINT"SCORE, LOOK (INVEN), QUIT, HELP"
1310 PRINT
1320 RETURN
1330 PRINT
1340 PRINT"YOU ARE STANDING IN THE FRONT YARD IN"
1350 PRINT"FRONT OF A MYSTERIOUS OLD MANSION NEAR"
1360 PRINT"A CEMETERY.  NO ONE IS AROUND FOR"
1370 PRINT"MILES.  YOU'VE HEARD LEGENDS ABOUT"
1380 PRINT"THIS PLACE -- THAT IT'S HAUNTED, AND"
1390 PRINT"THAT CROOKS USED IT AS A HIDEOUT UNTIL"
1400 PRINT"THEY MYSTERIOUSLY DISAPPEARED."
1410 PRINT
1420 PRINT"YOU'VE DECIDED TO EXPLORE THIS PLACE."
1430 PRINT"MAYBE YOU CAN FIND SOME LOOT LEFT"
1440 PRINT"BEHIND BY THE CROOKS.  EVEN IF YOU"
1450 PRINT"DON'T, THIS WILL BE AN EXCITING . . ."
1460 PRINT:PRINT TAB(8);TT$;"!"
1470 RETURN
1480 PRINT CHR$(147)
1490 PRINT TT$;SPC(2);"FROM 'MIND MOVES'"
1500 PRINT:PRINT
1510 PRINT"OPTIONS:":PRINT
1520 PRINT" 1 - PLAY STANDARD GAME."
1530 PRINT" 2 - PLAY RANDOM GAME."
1540 PRINT" 3 - INSTRUCTIONS FOR PLAYING."
1550 PRINT" 4 - END PROGRAM."
1560 PRINT
1570 PRINT"PRESS 1, 2, 3, OR 4"
1580 GET R$:IF LEN(R$)=0 THEN 1580
1590 IF R$<"1" OR R$>"4" THEN 1580
1600 PRINT R$
1610 ON VAL(R$) GOTO 1630,1640,1730,1980
1620 STOP
1630 RETURN
1640 FOR J=1 TO NW:W=TW(J):GOSUB 320
1650 W=W+INT(5*RND(1))-2
1660 IF W>NR THEN W=NR
1670 IF W<1 THEN W=1
1680 TW(J)=T*100+W
```

```
1690 NEXT
1700 PRINT:PRINT"RANDOM GAME NOW SET UP."
1710 FOR J=1 TO 3000:NEXT
1720 RETURN
1730 PRINT:PRINT
1740 PRINT TT$;SPC(2);"MINI-INSTRUCTIONS"
1750 PRINT
1760 PRINT"THIS IS AN ADVENTURE-TYPE GAME."
1770 PRINT"ENTER COMMANDS AS ONE OR TWO WORDS."
1780 PRINT"THE FIRST WORD IS USUALLY A VERB, SUCH"
1790 PRINT"AS 'GET' OR 'GO' OR 'USE'.  THE SECOND"
1800 PRINT"WORD IS A NOUN OR A DIRECTION."
1810 PRINT
1820 PRINT"YOU CAN ENTER THE COMMAND 'HELP' TO"
1830 PRINT"FIND OUT WHAT VERBS YOU CAN USE."
1840 PRINT
1850 PRINT"TO MOVE IN A DIRECTION, YOU CAN SIMPLY"
1860 PRINT"GIVE A ONE WORD COMMAND LIKE 'EAST' OR"
1870 PRINT"JUST 'E', OR YOU CAN SAY 'GO EAST'."
1880 PRINT
1890 PRINT"THE OBJECT OF ESCAPADE IS TO EXPLORE"
1900 PRINT"ALL ROOMS AND DROP ALL VALUABLES IN"
1910 PRINT"THE FRONT YARD."
1920 PRINT
1930 PRINT"FULL DETAILS ARE IN 'MIND MOVES'"
1940 PRINT"FROM DILITHIUM PRESS."
1950 PRINT:PRINT"PRESS A KEY TO CONTINUE.";
1960 GET R$:IF LEN(R$)=0 THEN 1960
1970 GOTO 1480
1980 GOTO 3120
1990 PRINT CHR$(147):PRINT"ONE MOMENT PLEASE..."
2000 RN=1:NW=15:NC=6:NR=29:NA=39
2010 DIM TM(NR,5),TR(NR),TC(NC),TW(NW),CM(NA)
2020 DIM RS$(NR),RL$(NR),DC$(NC),DW$(NW),CM$(NA)
2030 DIM M$(25)
2040 FOR J=1 TO NR:FOR K=0 TO 5:READ TM(J,K):NEXT:NEXT
2050 DATA 2,1,1,1,0,0, 3,2,1,2,0,0, 7,5,2,4,19,0
2060 DATA 0,3,0,0,0,0, 0,6,0,3,0,0, 0,0,0,5,0,0
2070 DATA 0,8,3,0,0,0, 0,9,0,7,0,0, 12,10,0,8,0,0
2080 DATA 0,0,11,9,0,0, 10,0,0,0,0,0, 15,0,9,13,0,0
2090 DATA 0,12,0,14,0,16, 0,13,0,0,0,0, 0,0,12,0,0,0
2100 DATA 0,17,0,0,13,0, 0,0,18,16,0,0, 17,0,0,0,0,0
2110 DATA 23,25,0,20,0,3, 22,19,0,21,0,0, 0,20,0,0,0,0
2120 DATA 0,0,20,0,0,0, 24,0,19,0,0,0, 0,0,23,0,0,0
2130 DATA 26,0,0,19,0,0, 27,29,25,0,0,0, 0,28,26,0,0,0
2140 DATA 0,0,0,27,0,0, 0,0,0,26,0,0
2150 FOR J=1 TO NA:READ CM$(J),CM(J):NEXT
2160 DATA N,1,E,101,S,201,W,301,U,401,D,501,NORTH,1
2170 DATA EAST,101,SOUTH,201,WEST,301,UP,401,DOWN,501
2180 DATA GO,2,GET,3,TAKE,3,THROW,4,DROP,4,USE,4
2190 DATA SCORE,5,LOOK,6,INVEN,6,QUIT,7,HELP,8
2200 DATA GUN,10106,WATER,10206,SWORD,10307,GAS,10407
2210 DATA HANDC,10508,CUFFS,10508,DART,10607,BOOME,10707
2220 DATA ROPE,10807,KEYS,10909,LANTE,11010,RUBY,11111
```

```
2230 DATA COINS,11211,BAR,11311,CHAIN,11411,RING,11511
2240 FOR J=1 TO NW:READ DW$(J),TW(J):NEXT
2250 DATA SQUIRT GUN,104,BOTTLE OF WATER,207
2260 DATA RUBBER SWORD,304,VIAL OF MYSTERY GAS,428
2270 DATA PAIR OF HANDCUFFS,514,TRANQUILIZER DART,606
2280 DATA SHINY BOOMERANG,4,NYLON ROPE,20
2290 DATA RING OF KEYS,9902,LANTERN,9802
2300 DATA LARGE RED RUBY,5018,BAG OF COINS,3011
2310 DATA SILVER BAR,4024,GOLD CHAIN,4029
2320 DATA SMALL DIAMOND RING,2008
2330 FOR J=1 TO NC:READ DC$(J),TC(J):NEXT
2340 DATA HOBBLING GOBLIN,105,FAT BLACK CAT,210
2350 DATA GROSS GHOST,107,MUTANT BANDITO MOSQUITO,312
2360 DATA DROOLING GHOUL,19,FIGHTER SPIDER,18
2370 FOR J=1 TO NR:READ RS$(J),RL$(J):NEXT
2380 DATA FRONT YARD,IN FRONT OF A FRIGHTENING OLD HOUSE
2390 DATA FRONT PORCH,JUST OUTSIDE THE FRONT DOOR
2400 DATA FRONT HALL,WITH A BALCONY ABOVE AND MANY DOORS
2410 DATA FRONT CLOSET,A SMALL ROOM WITH A SINGLE DOOR
2420 DATA SITTING ROOM,WHICH HAS A VIEW OF THE FRONT YARD
2430 DATA DOWNSTAIRS BATH,A ONE-DOOR ROOM
2440 DATA LIVING ROOM,DUSTY FROM YEARS OF NEGLECT
2450 DATA PARLOR,THE CENTRAL DOWNSTAIRS ROOM
2460 DATA DINING ROOM,DO YOU DARE GO ANY FARTHER?
2470 DATA BACK CLOSET,A SMALL ROOM AND PASSAGEWAY
2480 DATA SEWING ROOM,BUT THEN SEW WHAT?
2490 DATA KITCHEN,OVERLOOKING AN ENCLOSED BACK YARD
2500 DATA PANTRY,A SMALL ROOM WITH EMPTY SHELVES
2510 DATA REAR CLOSET,NOT A VERY EXCITING PLACE TO BE
2520 DATA BACK YARD,COMPLETELY ENCLOSED BY FENCES
2530 DATA BASEMENT,DAMP AND DANK AND DUSTY
2540 DATA WINE CELLAR,LONG AGO EMPTIED
2550 DATA STORAGE ROOM,AN EXTREMELY UNPLEASANT ROOM
2560 DATA BALCONY,THE UPSTAIRS CROSSROADS
2570 DATA SIDE HALL,A SMALL AND NARROW ROOM
2580 DATA SMALL BATH,ABOUT WHICH NOTHING MORE CAN BE SAID
2590 DATA SMALL BEDROOM,TOO SMALL EVEN FOR A GHOUL
2600 DATA MASTER BEDROOM,ONCE LUXURIOUS BUT NO LONGER
2610 DATA MASTER BATH,WHICH HAS ONLY ONE DOOR
2620 DATA UPSTAIRS HALL,A LONG NARROW ROOM
2630 DATA GREEN ROOM,EVEN IF YOU HAVE A BLACK & WHITE TV
2640 DATA GUEST ROOM,FROM WHICH THE GUESTS ARE LONG GONE
2650 DATA GUEST BATH,SUFFERING FROM TERMINAL BATHTUB RING
2660 DATA STUDY,ALMOST AS DUSTY AS THE BASEMENT
2670 FOR J=1 TO 22:READ M$(J):NEXT
2680 DATA I DON'T UNDERSTAND YOU
2690 DATA YOU CAN'T GO THAT WAY
2700 DATA YOU ALREADY HAVE THAT
2710 DATA THAT WAY IS BLOCKED BY A
2720 DATA I CAN'T DO THAT
2730 DATA IT'S WET BUT UNHARMED
2740 DATA IT ISN'T AFFECTED
2750 DATA THEY DON'T FIT IT -- IT IGNORES THEM
```

```
2760 DATA THEY BOUNCE OFF HARMLESSLY
2770 DATA IT'S IRRITATED BUT UNHARMED
2780 DATA "EASY COME, EASY GO"
2790 DATA IT RUNS AWAY FRIGHTENED
2800 DATA "SOAKED, IT RUNS AWAY"
2810 DATA IT VAPORIZES IN SURPRISE
2820 DATA IT FLIES AWAY COUGHING
2830 DATA "FEARING CAPTURE, IT DISAPPEARS"
2840 DATA IT CLIMBS TO THE CEILING AND SLEEPS
2850 DATA NOTHING LIKE THAT IS HERE
2860 DATA GOT IT -- IT'S NOW IN YOUR BAG
2870 DATA YOU DON'T HAVE SUCH A THING
2880 DATA IT FALLS TO THE FLOOR
2890 DATA YOU CAN'T CARRY THAT MUCH AT ONCE
2900 RETURN
2910 PRINT CHR$(147);CHR$(142);CHR$(8);CHR$(158)
2920 POKE 53269,0:POKE 53280,6:POKE 53281,6
2930 TT$="E S C A P A D E"
2940 PRINT:PRINT
2950 PRINT TAB(12);TT$
2960 PRINT:PRINT
2970 PRINT TAB(4);"COPYRIGHT 1984 DILITHIUM PRESS"
2980 PRINT:PRINT:PRINT:PRINT
2990 PRINT TAB(6);"FROM THE BOOK 'MIND MOVES'"
3000 PRINT:PRINT
3010 PRINT TAB(5);"BY TOM RUGG AND PHIL FELDMAN"
3020 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
3030 PRINT TAB(9);"PRESS A KEY TO BEGIN"
3040 GET R$:IF LEN(R$)=0 THEN 3040
3050 K=RND(-TI)
3060 J=1:W=0:T=0:I=1
3070 RETURN
3080 PRINT
3090 PRINT TAB(6);"PRESS A KEY TO CONTINUE"
3100 GET R$:IF LEN(R$)=0 THEN 3100
3110 RETURN
3120 POKE 53280,14:POKE 53281,6
3130 PRINT CHR$(147);CHR$(9);CHR$(154)
9990 END

READY.
```

EASY CHANGES

1. You can scatter the objects among a wider range of rooms when you choose the random game (option 2) by changing this line:

```
1650 W=W+INT(9*RND(1))-4
```

Be aware that this increases the chances that you will not be able to score the maximum point total, because a creature may end up blocking the path to the room containing the weapon needed to neutralize that creature.

2. Change the names and descriptions of the rooms by changing lines 2380 to 2660. Each line has the name of the room, then a comma, then a short descriptive phrase that is displayed the first time the room is entered. Don't include any commas in the descriptive phrase unless you enclose the entire phrase in quotation marks. For example, suppose you want to rename the front hall to be the entryway, and you want to describe it as a dusty, dark room. Make this change:

```
2400 DATA ENTRYWAY,"A DUSTY, DARK
     ROOM"
```

3. Make a more visible separation on the screen between one command and the next by inserting:

```
435 FOR J=1 TO 36:PRINT" + ";:NEXT:PRINT
```

4. Change the number of objects your bag can hold by changing the 5 in line 830. For example, to limit the bag's capacity to three items, make this change:

```
830 IF NB>=3 THEN M=22:GOTO 860
```

PROGRAMMER'S NOTEBOOK

The dominant feature of the design of ESCAPADE is that it is table-driven. Nearly all the critical information is in numeric and string tables (arrays), all of which are initialized from DATA statements by the routines in lines 1990 to 2900.

When a command is entered, it gets broken into two words, if necessary. The command table (CM$) is searched to see if the command is a legal one. If not, an error message is displayed. The command table has corresponding numeric values (the CM array) that indicate which processing routine handles that command (rightmost two digits) and other data about that command, if necessary (left digits).

The weapons and valuables are described in a similar way using DW$ and TW. The rightmost two digits in each TW array entry show which room the object is in. The left digits indicate the creature that the weapon is effective against if less than 10, or the value of the object in scoring points if between 10 and 90. Values over 90 are included for use in doing the first Suggested Project.

The key to what movement is allowed from room to room is in the table of movement (TM). Each room (1 to 29) has six values (indexed as 0 through 5), corresponding with north, east, south, west, up, and down. Moving east from a room, for example, puts the player into the room indicated by the east value for the originating room. A value of 0 means there is no door in that direction. A value the same as the current room number means the player remains in the same room.

MAIN ROUTINES

130-200	Main loop. Calls other main subroutines.
210-250	Displays which room the player is now in.
260-310	Displays which objects are in the room.
320	Splits value in W into rightmost two digits (replaces W) and left digits (T).
330-420	Displays any creatures in the room, and indicates which directions doors lead.
430-610	Accepts command from player. Splits words. Checks words for legality.
620-630	Displays message number M.
640-670	Goes to the subroutine corresponding to the command entered.

680-760	Processes commands 1 and 2 (direction command without and with the word go).
770-860	Processes command 3 (get/take).
870-1000	Processes command 4 (use/drop/throw).
1010-1060	Processes command 5 (score).
1070-1150	Processes command 6 (look/inven).
1160-1270	Processes command 7 (quit).
1280-1320	Processes command 8 (help).
1330-1470	Displays introduction before player's first move.
1480-1600	Displays option menu and accepts response.
1610-1630	Branches to the proper routine to handle the menu response. Returns for standard game (option 1).
1640-1720	Sets up random game (option 2).
1730-1970	Displays instructions (option 3).
1980	Branches to the routine to end the program.
1990-2900	Subroutine to initialize variables and read DATA statements into arrays.
2910-3070	Subroutine to display title screen and initialize some variables.
3080-3110	Subroutine to wait for a key to be pressed.
3120-9990	Resets screen colors and ends the program.

MAIN VARIABLES

RN	Room number the player is currently in.
M	Message number to be displayed.
NW	Number of objects (weapons, valuables) in the game.
NC	Number of creatures in the game.
NR	Number of rooms.
NA	Number of command words and synonyms.
J, K	Loop and work variables.
L	Length of input command.
T, W	Temporary work variables.
PF	Pointer to the first character in a command word.
FL, FR	Left and right fields that a number is split into.
WL, WR	Work values of left and right fields of a number.
VA, VB	Command values corresponding with WA$ and WB$.
NB	Number of objects in player's bag.
MN	Move number.
I	Constant one.
TM	Table of movement.

TC	Table of creatures.
TW	Table of weapons, valuables, and other objects.
TR	Table of rooms, for determining which have been visited.
CM	Table of command numerical values (associated with each CM$).
RS$	Array of room names (short).
RL$	Array of room descriptions (long).
DW$	Array of weapon names.
DC$	Array of creature names.
CM$	Array of commands (first five characters).
M$	Array of messages.
R$	Reply from player (full command or single key pressed).
WA$, WB$	First and second words of player's command.
TT$	Title of the game (with alternate spaces).

SUGGESTED PROJECTS

1. Add obstacles to the game and commands to overcome them. For example, the lantern and the ring of keys are already included for such future uses in the game (as well as to have a few useless objects in the game to confuse the player a little). Put locks on some doors and require the player to "unlock door" before going through it. Require the player to carry the lantern when traveling in the basement, since it's dark down there.
2. When the player wants to play a random game, change the room layout slightly, not just the locations of the objects. Start with a change like this, which randomly (50 percent chance) eliminates the door between the front hall and the living room, but creates one between the sitting room and the parlor:

```
1692 IF RND(1)>.5 THEN TM(3,0)=0:TM(7,2)=0:
     TM(5,0)=8:TM(8,2)=5
```

3.

HOTSHOT

HOTSHOT brings the challenging action game of billiards into your living room (or wherever your computer happens to be). You have free reign at the billiards table, complete control over each shot's direction and speed. Compete against a friend, the computer, or just practice by yourself. Game parameters can be freely adjusted to match your level of skill.

Now please don't confuse billiards with the game of pool. Pool (more formally known as pocket billiards) is played on a table with pockets, usually with 16 balls. The more elegant game of billiards uses only three balls and no pockets. The objective in billiards is to strike your cue ball and have it hit each of the other two balls. Carefully planned geometric shots must be executed. In HOTSHOT, you choose among several variants of billiards, including the difficult three-cushion billiards.

Dynamic color graphics are used to display the excitement of the moving balls and their collisions. Eager for success, you plan each shot then watch it unfold. And, of course, you can make trick shots. After all, what's a game of billiards without trick shots? You can put various types of spin, or *English*, on your cue ball to cause special effects — just what's needed for those tough shots you seem to face so often.

OK, hustler, it's your shot!

RULES

This computer game closely simulates the real game of billiards. Three balls are used — red, white, and blue (very patriotic if you're American.) Each player shoots one of these balls, called his *cue ball*, and tries to make it hit both of the other balls, called *object balls*. One player uses the white ball as a cue ball and the other player uses the red ball as a cue ball.

In regular billiards, a shot is successful if the cue ball hits both object balls. By contrast, the game of three-cushion billiards requires additionally that the cue ball bounce off at least three cushions before it hits the second object ball. It is not necessary to hit three different cushions, and it is even possible to hit the same cushion twice in a row. It's okay if more than three cushions are hit before the second object ball is struck. Also, the cue ball can hit the first object ball before it strikes a cushion or after it strikes one cushion or more. If the two object balls are close together, a common strategy is to hit the cue ball around the table striking (at least) three cushions before colliding into the two object balls. However, most shots are attempted by hitting an object ball and deflecting the cue ball around the table into the second object ball. A shot is unsuccessful if the two object balls are never struck by the cue ball or if they are struck before the three-cushion requirement is met.

To repeat, a successful shot in regular billiards requires only that the cue ball hit each object ball. This is zero-cushion billiards, if you will. With HOTSHOT, you can play zero-, one-, two-, or three-cushion billiards. We recommend starting with zero-cushion billiards and increasing the cushion requirement as your skill progresses.

The game is played for a prescribed number of innings. During an inning, each player gets one turn. He may continue shooting as long as he continues to make successful shots. Each successful shot scores one point. When he

misses, it becomes his opponent's turn. After the prescribed number of innings is completed, the player with the higher score wins.

HOW TO USE IT

The program begins by displaying its title screen. You hit any key to start the option menu. First select who the players will be. Option 1 means that you will be competing by yourself (and using the white ball as the cue ball). Option 2 is for two human players, while option 3 pits you against the computer. Option 4 ends the game after resetting computer parameters. Simply press 1, 2, 3, or 4 to indicate your choice.

If you select option 2, you are asked for the names of the two players. Type in each name followed by [RETURN]. Limit each name to eight characters, or only the first eight characters you type will be used. The first player will get the white cue ball and the second player the red. If you choose to play against the computer, you will be asked who shoots first, and thus gets the white cue ball. Simply press 1 or 2 to indicate this choice.

The next option is to choose the size of the table. We've given you three choices cleverly called small, medium, and large. The difficulty of the game increases as the table size increases, so again we recommend starting with the small table and progressing to the larger ones as your skill improves. Simply press 1, 2, or 3 to indicate your choice.

Now you choose the number of cushions required for a successful shot. The toughest game is three-cushion, but you may also choose two-, one-, or zero-cushion (regular billiards) as explained above. Press 0, 1, 2, or 3 to select your preference.

Lastly, choose the number of innings for the game. You may pick any number between 1 and 99. Hit [RETURN] after making your choice.

Now the screen will clear and the game will begin. The computer will ask you to wait a moment while it initializes game variables. The three balls will appear on the table placed randomly for the first shot. The lower part of the screen is used for the scoreboard and information center. In the lower left corner is a message board indicating what action is expected next or what the result of the last shot was.

To take a shot, you must select three things — the angle or direction to strike your cue ball, the speed with which you strike it, and whether to strike it with any special spin or English. You will be prompted for these by messages in the lower left corner. The messages will be in white when white is shooting and in red when red is shooting. When the turn changes to the next player, his cue ball will flash on the table. Also, a white or red ball is displayed on the scoreboard in front of the current shooter's name.

Now let's discuss how a shot is made. First you must indicate the direction of the shot. The message board says ALIGN SHOT and indicates that the space bar is used to do this. On the upper left corner of the perimeter of the table are displayed cross hairs of the same color as that of the player shooting. When you press the space bar, these cross hairs move clockwise around the table. If you hold down the [SHIFT] key as well as the space bar, they move counterclockwise. You may hit the space bar in single strokes or hold it down to get repeated movement. Also, holding down the lower right cursor movement key instead of the space bar causes a rapid movement of the cross hairs around the table. The direction of your shot is considered to be from the center of the cue ball to the center of the cross hairs. Typically, you use the cursor key to get the cross hairs in the general area you want, then use the space bar or [SHIFT] space bar to fine-tune your shot direction. When the cross hairs are positioned properly, press [RETURN] to move on.

Now you are asked if a SPECIAL SHOT is desired, and you are told to use the cursor keys for this. These trick shots are done with the aid of the special shot box in the lower right corner of the screen. You will notice cross hairs in the

center of this box. By using the four cursor movement keys (the two keys in the lower right corner of the keyboard either SHIFTed or not), you can move the cross hairs around the box. This box is to be viewed as the front plane of the ball as the player strikes it with the cue stick. That is, if the cross hairs are to the left of center, the player is striking the ball to the left of center and thus imparting left spin or English on the ball. Similarly, right English can be imparted. When the cue ball hits its first *rail* (another word for cushion), these spins will cause it to bounce off with exaggerated movement in the direction of the spin. The further the cross hairs are from the center, the stronger is the spin and the stronger is its effect. If the cross hairs are placed below center, the ball is spinning backwards as it is struck. This effect, called *draw*, causes the cue ball to move back in the direction it came from upon striking an object ball. The more the object ball is struck *square on* and the further below center the cross hairs are placed, the greater the effect of draw. Finally, if the cross hairs are placed above center, top spin will be imparted to the cue ball. This effect, called *follow*, causes it to continue forward with accelerated movement upon collision with an object ball. Just as in real billiards, any up or down spin on the cue ball dissipates after a collision with an object ball, but side spin remains. After a cue ball bounces off a cushion, all spins are considered to dissipate. When you have selected any desired spin with the cross hairs, indicate this by hitting the [RETURN] key. Most of the time, you will not need any special shot and will hit the [RETURN] key immediately. By discriminating use of these special shots, all kinds of otherwise impossible shots can be made.

Lastly, the speed at which the cue ball is struck must be selected. You can choose any number from 1 to 9, with the larger numbers being stronger shots. Simply press a number from 1 to 9, and the shot will occur.

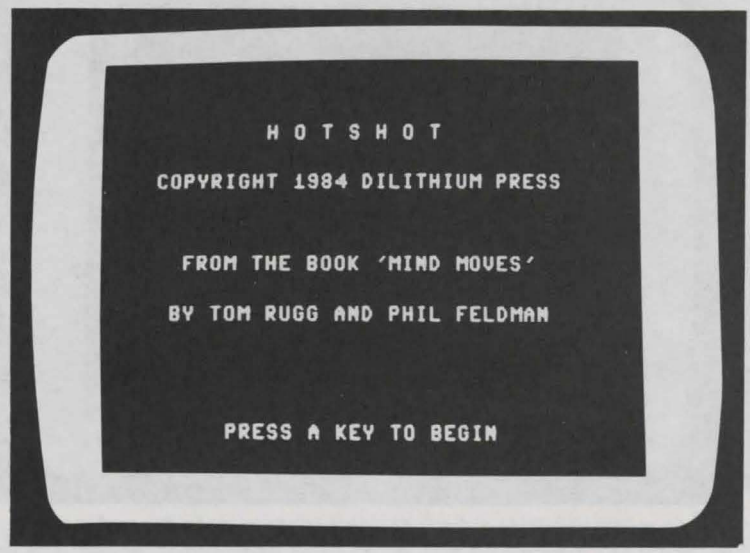
Each shot is shown graphically as it develops. You will have fun watching the shot progress and seeing if you've planned it correctly. It is shown in a kind of slow motion so you can see exactly what's happening. The display of

collisions is especially slow. We're tempted to say that this is intentional so you can study the exact physics and collision angles better. However, the truth is that the slower movement occurs because of the time it takes the computer's BASIC program to perform the calculations involved in determining the results of a collision. By the way, at faster speeds, and especially with exaggerated spins, the shot dynamics become a little unpredictable. You may see some funny looking collisions or even possibly a ball moving through another one. This mirrors real life in that the stronger, most exaggerated shots are the hardest to control successfully. And don't worry if you occasionally find yourself using a little *body English* to nudge the balls. Though it can't really help, it can't really hurt, either.

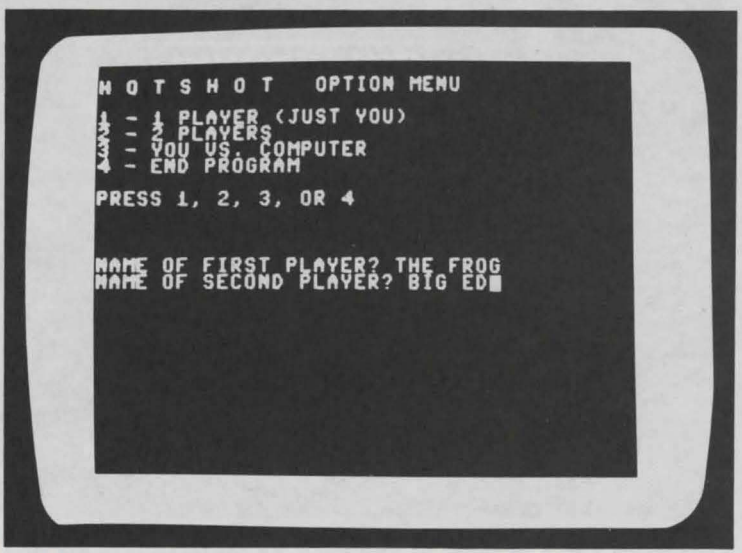
When the shot finishes, its result is displayed in the message area. If successful, the computer says NICE SHOT, the player's score is incremented, and he is prompted to shoot again. If unsuccessful, the player is told why with one of two messages. He may may see the message BOTH NOT HIT meaning that the cue ball never struck both object balls. The other possible message is # RAILS ONLY 1, meaning that the cue ball had hit only one rail at the time the second object ball was struck and more rails were required. Of course, this latter message may indicate that only zero, or perhaps only two rails were hit, as appropriate. When the computer shoots, the message MY SHOT is displayed, and the computer takes its turn. You'll find the computer an unpredictable opponent. Sometimes it'll make many good shots in succession, and sometimes it'll have its long dry spells, too.

The game continues with players taking turns until the prescribed number of innings is completed. The computer acknowledges the winner (or indicates a tie) then asks that any key be hit to proceed. When this is done, the title screen reappears. Hit a key again for the option menu. Now you may continue with another game of HOTSHOT or end the program to return to the real world.

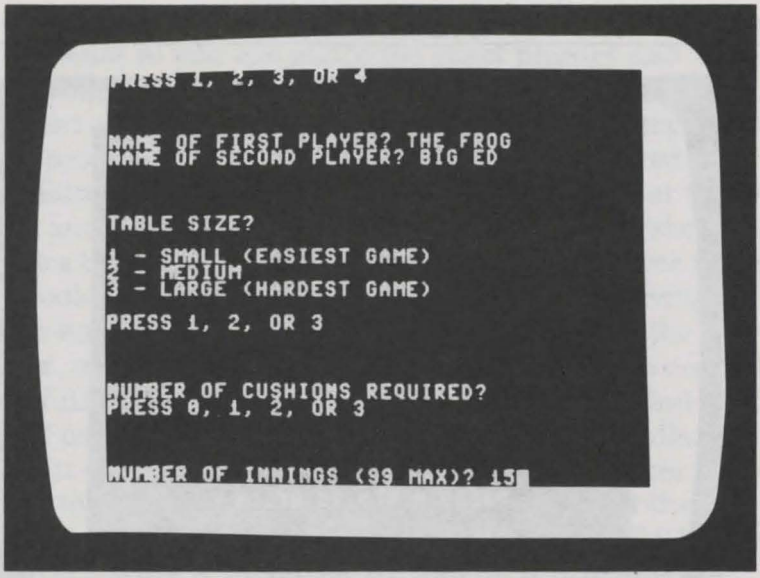
SAMPLE RUN



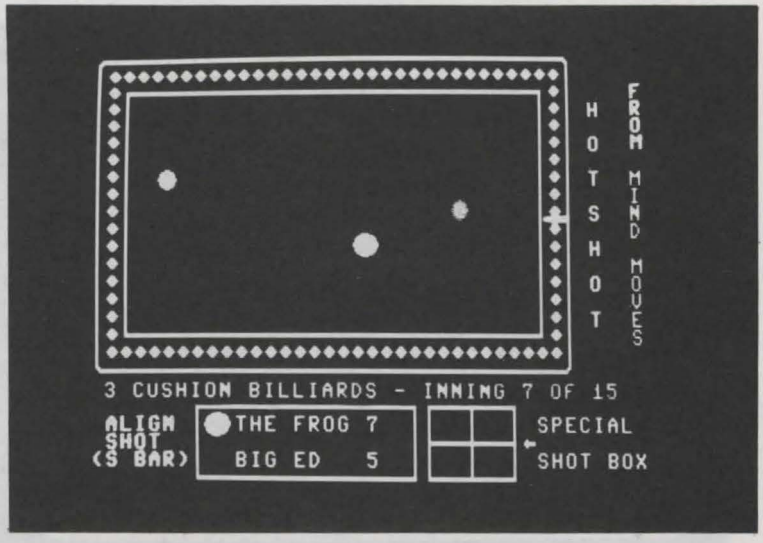
The title screen is displayed, and a key must be hit to begin the option menu.



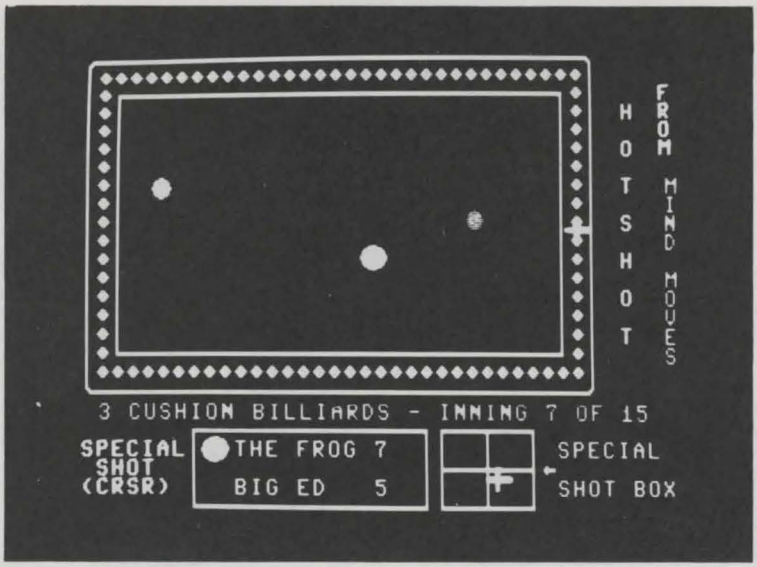
A 2 is pressed for a two-player game. The players' names are THE FROG and BIG ED.



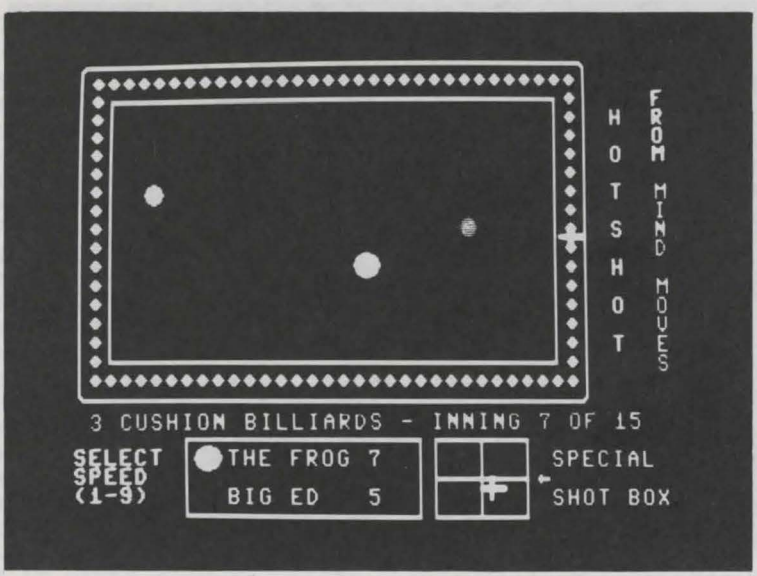
Being experienced players, they press 3 for both the table size and the number of cushions required. They decide to play a 15-inning game.



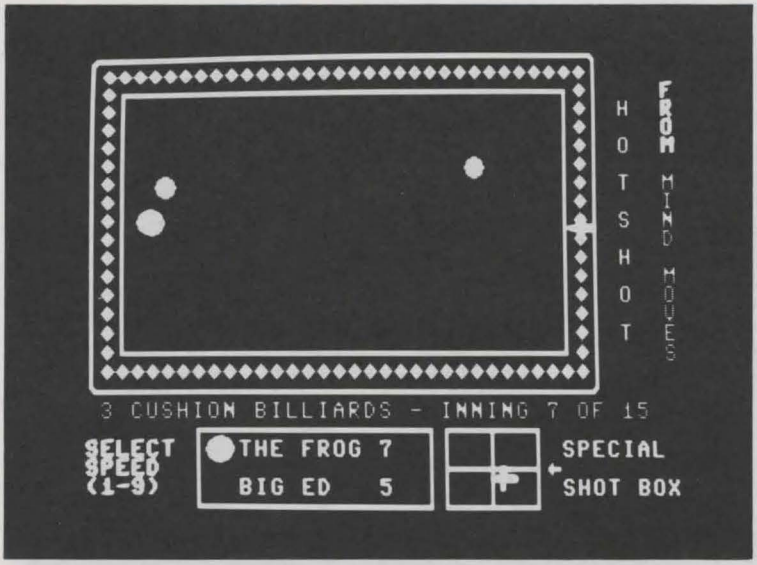
With the score 7 to 5 in favor of The Frog, he contemplates his shot to begin the seventh inning. He positions the cross hairs near the center of the right cushion to aim his white cue ball toward the blue object ball.



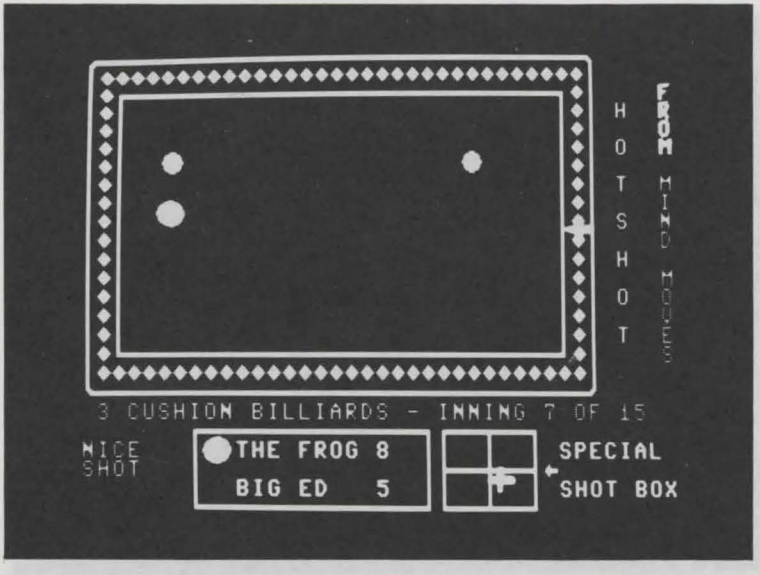
The Frog selects moderate right English for his shot. He hopes to hit the blue ball below center then to have his cue ball strike the right, bottom, and left cushions before colliding with the other (red) object ball.



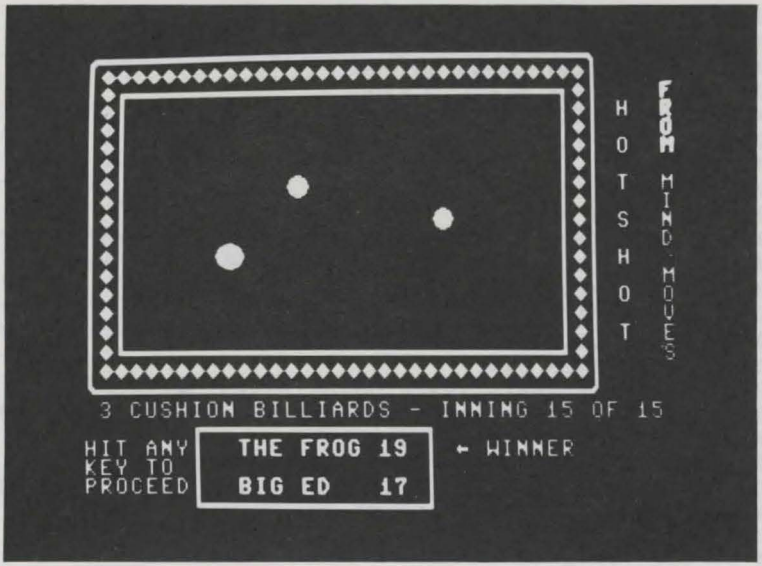
A speed of 4 will be used. When 4 is pressed, the shot is displayed graphically.



The shot is shown in mid-action. The blue ball has been hit and the white cue ball is heading toward the left rail after striking the right and bottom rails.



The Frog pulls it off. The computer acknowledges the nice shot. It will still be The Frog's turn in the seventh inning.



After a hard-fought, grueling match, The Frog wins by a score of 19 to 17. Big Ed is ready to hit a key to reinitialize the game for a rematch.

PROGRAM LISTING

```
100 REM: HOTSHOT
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY PHIL FELDMAN AND TOM RUGG
130 GOTO 830
140 DD=D3-D4:EE=E3-E4:U=SQR(D3*D3+D4*D4+E3*E3+E4*E4)
150 IF X4=X3 THEN AL=SGN(Y4-Y3)*HP:GOTO 170
160 AL=ATN((Y4-Y3)/(X4-X3))
170 IF DD=0 THEN GA=SGN(EE)*HP:GOTO 190
180 GA=ATN(EE/DD)
190 TH=AL-GA:IF TH>HP THEN TH=TH-PI
200 IF TH<-HP THEN TH=TH+PI
210 SI=SIN(TH):CI=COS(TH):SA=SIN(AL):CA=COS(AL)
220 G3=U*SI*SA+D4:H3=-U*SI*CA+E4
230 G4=U*CI*CA+D4:H4=U*CI*SA+E4
240 RA=SQR(G3*G3+G4*G4+H3*H3+H4*H4)/U
250 G3=G3/RA:H3=H3/RA:G4=G4/RA:H4=H4/RA
260 RETURN
270 IF XA<LE THEN XA=LE:DA=-DA:CC=CC+1:EA=EA-SE:SE=0:TE=0
280 IF XA>RE THEN XA=RE:DA=-DA:CC=CC+1:EA=EA+SE:SE=0:TE=0
290 IF YA<UE THEN YA=UE:EA=-EA:CC=CC+1:DA=DA+SE:SE=0:TE=0
300 IF YA>BE THEN YA=BE:EA=-EA:CC=CC+1:DA=DA-SE:SE=0:TE=0
310 IF XB<LE THEN XB=LE:DB=-DB
320 IF XB>RE THEN XB=RE:DB=-DB
```

```
330 IF YB<UE THEN YB=UE:EB=-EB
340 IF YB>BE THEN YB=BE:EB=-EB
350 IF XC<LE THEN XC=LE:DC=-DC
360 IF XC>RE THEN XC=RE:DC=-DC
370 IF YC<UE THEN YC=UE:EC=-EC
380 IF YC>BE THEN YC=BE:EC=-EC
390 RETURN
400 CC=0:BH=0:FH=-1
410 FOR K=1 TO NN
420 XA=XA+DA:YA=YA+EA:XB=XB+DB:YB=YB+EB:XC=XC+DC:YC=YC+EC
430 GOSUB 270
440 X=XA-XB:Y=YA-YB
450 IF X*X+Y*Y>C THEN 550
460 BH=BH OR 1:IF BH=3 THEN FH=CC:BH=4
470 IF XA<XB THEN 510
480 X3=XB:Y3=YB:X4=XA:Y4=YA:D3=DB:E3=EB:D4=DA:E4=EA
490 GOSUB 140:DA=G4:EA=H4:DB=G3:EB=H3:IF TE=0 THEN 540
500 DA=DA+TE*D4:EA=EA+TE*E4:TE=0:GOTO 540
510 X3=XA:Y3=YA:X4=XB:Y4=YB:D3=DA:E3=EA:D4=DB:E4=EB
520 GOSUB 140:DB=G4:EB=H4:DA=G3:EA=H3:IF TE=0 THEN 540
530 DA=DA+TE*D3:EA=EA+TE*E3:TE=0
540 GOSUB 780:X=XA-XB:Y=YA-YB:IF X*X+Y*Y<=C THEN 540
550 X=XA-XC:Y=YA-YC
560 IF X*X+Y*Y>C THEN 660
570 BH=BH OR 2:IF BH=3 THEN FH=CC:BH=4
580 IF XA<XC THEN 620
590 X3=XC:Y3=YC:X4=XA:Y4=YA:D3=DC:E3=EC:D4=DA:E4=EA
600 GOSUB 140:DA=G4:EA=H4:DC=G3:EC=H3:IF TE=0 THEN 650
610 DA=DA+TE*D4:EA=EA+TE*E4:TE=0:GOTO 650
620 X3=XA:Y3=YA:X4=XC:Y4=YC:D3=DA:E3=EA:D4=DC:E4=EC
630 GOSUB 140:DC=G4:EC=H4:DA=G3:EA=H3:IF TE=0 THEN 650
640 DA=DA+TE*D3:EA=EA+TE*E3:TE=0
650 GOSUB 780:X=XA-XC:Y=YA-YC:IF X*X+Y*Y<=C THEN 650
660 X=XB-XC:Y=YB-YC
670 IF X*X+Y*Y>C THEN 740
680 IF XB<XC THEN 710
690 X3=XC:Y3=YC:X4=XB:Y4=YB:D3=DC:E3=EC:D4=DB:E4=EB
700 GOSUB 140:DB=G4:EB=H4:DC=G3:EC=H3:GOTO 730
710 X3=XB:Y3=YB:X4=XC:Y4=YC:D3=DB:E3=EB:D4=DC:E4=EC
720 GOSUB 140:DC=G4:EC=H4:DB=G3:EB=H3
730 GOSUB 780:X=XB-XC:Y=YB-YC:IF X*X+Y*Y<=C THEN 730
740 POKE P3,XA:POKE P4,YA:POKE P5,XB:POKE P6,YB
750 POKE P7,XC:POKE P8,YC
760 NEXT:GOSUB 270
770 RETURN
780 POKE P3,XA:POKE P4,YA:POKE P5,XB:POKE P6,YB
790 POKE P7,XC:POKE P8,YC
800 XA=XA+DA:YA=YA+EA:XB=XB+DB:YB=YB+EB:XC=XC+DC:YC=YC+EC
810 GOSUB 270
820 RETURN
830 XA=0:XB=0:XC=0:YA=0:YB=0:YC=0:X=0:Y=0
840 DA=0:DB=0:DC=0:EA=0:EB=0:EC=0
850 LE=0:RE=0:UE=0:BE=0
860 D3=0:D4=0:E3=0:E4=0
```

```
870 X3=0:X4=0:Y3=0:Y4=0:G3=0:G4=0:H3=0:H4=0
880 DD=0:EE=0:U=0:AL=0:GA=0:TH=0:RA=0
890 SI=0:CI=0:SA=0:CA=0
900 C=99:PI=3.14159265:HP=PI/2
910 P3=0:P4=0:P5=0:P6=0:P7=0:P8=0:V=53248
920 FH=0:BH=0:CC=0:TE=0:SE=0
930 K=0:NN=0
940 ZA=0:ZB=0
950 HC$=CHR$(147):HM$=CHR$(19):B$=CHR$(32)
960 RR$=CHR$(28):BB$=CHR$(31):WW$=CHR$(5):LB$=CHR$(154)
970 FF=1
980 GOSUB 1160:GOSUB 3420:GOSUB 2990:GOSUB 2500:GOSUB 2260
990 B7$=B$+B$:B7$=B7$+B7$+B7$+B$
1000 GOSUB 2910
1010 P3=V+4:P4=V+5:P5=V+6:P6=V+7:P7=V+8:P8=V+9
1020 GOSUB 1330
1030 GET R$:IF R$<>"" THEN 1030
1040 POKE V+21,28:GOSUB 2910:PRINT BB$
1050 CR=23:FOR CD=19 TO 23:GOSUB 2940
1060 PRINT B7$;B7$;B$;B$;:NEXT
1070 CD=21:CR=24:IF FL>1 THEN 1090
1080 GOSUB 2940:PRINT"GAME OVER":GOTO 1130
1090 CD=20:IF ZA>ZB THEN 1120
1100 IF ZB>ZA THEN CD=22:GOTO 1120
1110 CD=21:GOSUB 2940:PRINT"TIE GAME":GOTO 1130
1120 GOSUB 2940:PRINT CHR$(95);B$;"WINNER"
1130 GOSUB 2910:PRINT"HIT ANY":PRINT"KEY TO":PRINT"PROCEED"
1140 GET R$:IF R$="" THEN 1140
1150 CLR:GOTO 130
1160 POKE V+21,0:RESTORE
1170 POKE V+23,0:POKE V+28,0:POKE V+29,0:POKE V+16,0
1180 ADD=832:PAGE=ADD/64
1190 FOR J=ADD TO ADD+29:READ BYTE:POKE J,BYTE:NEXT
1200 FOR J=ADD+30 TO ADD+63:POKE J,0:NEXT
1210 DATA 0,60,0,0,255,0,1,255,128,1,255,128,3,255,192
1220 DATA 3,255,192,1,255,128,1,255,128,0,255,0,0,60,0
1230 FOR J=2042 TO 2046:POKE J,PAGE:NEXT
1240 ADD=896:PAGE=ADD/64
1250 FOR J=ADD TO ADD+29:READ BYTE:POKE J,BYTE:NEXT
1260 FOR J=ADD+30 TO ADD+63:POKE J,0:NEXT
1270 DATA 0,24,0,0,24,0,0,24,0,0,24,0,3,255,192
1280 DATA 3,255,192,0,24,0,0,24,0,0,24,0,0,24,0
1290 POKE 2040,PAGE:POKE 2041,PAGE
1300 FOR J=1 TO 5:POKE V+40+J,VAL(MID$("12612",J,1)):NEXT
1310 POKE V+10,82:POKE V+11,217:POKE V+12,82:POKE V+13,233
1320 RETURN
1330 XB=(RE-LE)/4+LE:XA=XB+(RE-LE)/4:XC=XA+(RE-LE)/4
1340 Q=INT((BE+UE)/2)-20
1350 YA=Q+RND(1)*40:YB=Q+RND(1)*40:YC=Q+RND(1)*40
1360 POKE V+4,XA:POKE V+5,YA:POKE V+6,XB:POKE V+7,YB
1370 POKE V+8,XC:POKE V+9,YC
1380 FOR M=1 TO NI
1390 CD=18:CR=21:PRINT BB$;:GOSUB 2940
1400 PRINT B7$;B7$:GOSUB 2940
```

```
1410 PRINT"- INNING";M;"OF";NI
1420 GOSUB 2910:DB=0:EB=0:DC=0:EC=0
1430 POKE V+21,0:POKE V+39,1:POKE V+40,1:PRINT WW$;
1440 IF FL>1 AND M>1 THEN GOSUB 2020
1450 POKE V+21,60
1460 IF FF=0 THEN 1500
1470 FOR Q=1 TO 5:FOR J=1 TO 300:NEXT
1480 POKE V+21,56:FOR J=1 TO 300:NEXT
1490 POKE V+21,60:NEXT
1500 Q=4:GOSUB 1720
1510 IF FH<NC THEN 1550
1520 PRINT"NICE":PRINT"SHOT":FOR Q=1 TO 3000:NEXT:ZA=ZA+1
1530 CD=20:CR=18:PRINT LB$:GOSUB 2940:PRINT ZA;WW$;
1540 GOSUB 2910:DB=0:EB=0:DC=0:EC=0:POKE V+21,60:GOTO 1500
1550 GOSUB 1860
1560 IF FL=1 THEN 1700
1570 GOSUB 2910:DB=0:EB=0:DC=0:EC=0
1580 POKE V+21,0:POKE V+39,2:POKE V+40,2:PRINT RR$;
1590 GOSUB 2020:POKE V+21,92
1600 IF FF=0 THEN 1640
1610 FOR Q=1 TO 5:FOR J=1 TO 300:NEXT
1620 POKE V+21,88:FOR J=1 TO 300:NEXT
1630 POKE V+21,92:NEXT
1640 Q=3:GOSUB 1720
1650 IF FH<NC THEN 1690
1660 PRINT"NICE":PRINT"SHOT":FOR Q=1 TO 3000:NEXT:ZB=ZB+1
1670 CD=22:CR=18:PRINT LB$:GOSUB 2940:PRINT ZB;RR$;
1680 GOSUB 2910:DB=0:EB=0:DC=0:EC=0:POKE V+21,92:GOTO 1640
1690 GOSUB 1860
1700 NEXT
1710 RETURN
1720 GET R$:IF R$<>"" THEN 1720
1730 IF FL=Q THEN GOSUB 2070:GOTO 1840
1740 PRINT B$;"ALIGN":PRINT B$;"SHOT":PRINT"(S BAR)"
1750 GOSUB 2130:GOSUB 2910
1760 PRINT"SPECIAL":PRINT B$;"SHOT":PRINT"(CRSR)"
1770 GOSUB 1910
1780 GET R$:IF R$<>"" THEN 1780
1790 GOSUB 2910:PRINT"SELECT":PRINT"SPEED":PRINT"(1-9)"
1800 GET R$:IF R$="" THEN 1800
1810 IF R$<"1" OR R$>"9" THEN 1800
1820 NN=20+15*VAL(R$):U=2+VAL(R$)*0.7
1830 RA=SQR(DA*DA+EA*EA)/U:DA=DA/RA:EA=EA/RA
1840 GOSUB 400:GOSUB 2910:PRINT BB$;
1850 RETURN
1860 PRINT"NO GOOD";LB$
1870 IF FH<0 THEN PRINT B$;"BOTH":PRINT"NOT HIT":GOTO 1890
1880 PRINT"# RAILS":PRINT"ONLY";FH
1890 FOR Q=1 TO 3000:NEXT
1900 RETURN
1910 X=224:Y=225:POKE V+2,X:POKE V+3,Y
1920 POKE V+21,PEEK(V+21) OR 2
1930 GET R$:IF R$="" THEN 1930
1940 IF R$=CHR$(157) AND X>200 THEN X=X-1
```

```
1950 IF R$=CHR$(29) AND X<248 THEN X=X+1
1960 IF R$=CHR$(145) AND Y>209 THEN Y=Y-1
1970 IF R$=CHR$(17) AND Y<241 THEN Y=Y+1
1980 IF R$=CHR$(13) THEN 2000
1990 POKE V+2,X:POKE V+3,Y:GOTO 1930
2000 SE=(X-224)/4:TE=(225-Y)/8
2010 RETURN
2020 Q=XA:XA=XB:XB=Q:Q=YA:YA=YB:YB=Q
2030 Q=PEEK(V+41):POKE V+41,PEEK(V+42):POKE V+42,Q
2040 Q=PEEK(V+4):POKE V+4,PEEK(V+6):POKE V+6,Q
2050 Q=PEEK(V+5):POKE V+5,PEEK(V+7):POKE V+7,Q
2060 RETURN
2070 PRINT"MY SHOT":SE=0:TE=0:DA=XB-XA:EA=YB-YA
2080 FOR K=1 TO 1500:NEXT
2090 NN=50+RND(1)*100:U=4+3*RND(1)
2100 RA=SQR(DA*DA+EA*EA)/U:DA=DA/RA:EA=EA/RA
2110 DA=DA*(0.75+0.5*RND(1)):EA=EA*(0.75+0.5*RND(1))
2120 RETURN
2130 J=1:SS$=CHR$(160):RT$=CHR$(13)
2140 POKE V+21,PEEK(V+21) OR 1
2150 POKE V,XP(J):POKE V+1,YP(J):POKE V+16,ZP(J)
2160 GET R$:IF R$="" THEN 2160
2170 IF R$=B$ THEN J=J+1
2180 IF R$=SS$ THEN J=J-1
2190 IF R$=RT$ THEN 2240
2200 IF R$=CHR$(29) THEN J=J+20
2210 IF J>NP THEN J=1
2220 IF J<1 THEN J=NP
2230 GOTO 2150
2240 DA=XP(J)+256*ZP(J)-XA:EA=YP(J)-YA
2250 RETURN
2260 ON TS GOTO 2270,2340,2410
2270 LE=40:RE=193:UE=71:BE=138
2280 NP=558:DIM XP(NP),YP(NP),ZP(NP)
2290 FOR J=1 TO 184:XP(J)=24+J:YP(J)=56:ZP(J)=0:NEXT
2300 FOR J=185 TO 280:XP(J)=208:YP(J)=J-128:ZP(J)=0:NEXT
2310 FOR J=281 TO 463:XP(J)=488-J:YP(J)=152:ZP(J)=0:NEXT
2320 FOR J=464 TO 558:XP(J)=25:YP(J)=615-J:ZP(J)=0:NEXT
2330 GOTO 2490
2340 LE=40:RE=224:UE=71:BE=154
2350 NP=654:DIM XP(NP),YP(NP),ZP(NP)
2360 FOR J=1 TO 216:XP(J)=24+J:YP(J)=56:ZP(J)=0:NEXT
2370 FOR J=217 TO 328:XP(J)=240:YP(J)=J-160:ZP(J)=0:NEXT
2380 FOR J=329 TO 543:XP(J)=568-J:YP(J)=168:ZP(J)=0:NEXT
2390 FOR J=544 TO 654:XP(J)=25:YP(J)=711-J:ZP(J)=0:NEXT
2400 GOTO 2490
2410 LE=40:RE=255:UE=71:BE=170
2420 NP=750:DIM XP(NP),YP(NP),ZP(NP)
2430 FOR J=1 TO 231:XP(J)=24+J:YP(J)=56:ZP(J)=0:NEXT
2440 FOR J=232 TO 248:XP(J)=J-232:YP(J)=56:ZP(J)=1:NEXT
2450 FOR J=249 TO 376:XP(J)=16:YP(J)=J-192:ZP(J)=1:NEXT
2460 FOR J=377 TO 392:XP(J)=392-J:YP(J)=184:ZP(J)=1:NEXT
2470 FOR J=393 TO 623:XP(J)=648-J:YP(J)=184:ZP(J)=0:NEXT
2480 FOR J=624 TO 750:XP(J)=25:YP(J)=807-J:ZP(J)=0:NEXT
```

```
2490 RETURN
2500 POKE 53280,13:POKE 53281,13:PRINT LB$;HC$;
2510 V$=CHR$(98):D$=CHR$(122):H$=CHR$(99)
2520 CA$=CHR$(176):CB$=CHR$(174):CC$=CHR$(173):CD$=CHR$(189)
2530 S$="":HH$="":DD$=""
2540 FOR J=1 TO 4:S$=S$+H$:NEXT
2550 FOR J=1 TO (16+4*TS):HH$=HH$+H$:NEXT
2560 FOR J=1 TO (20+4*TS):DD$=DD$+D$:NEXT
2570 PRINT CHR$(117);HH$;S$;CHR$(105)
2580 PRINT V$;DD$;V$
2590 PRINT V$;D$;CA$;HH$;CB$;D$;V$
2600 FOR J=1 TO (7+2*TS)
2610 PRINT V$;D$;V$;SPC(16+4*TS);V$;D$;V$:NEXT
2620 PRINT V$;D$;CC$;HH$;CD$;D$;V$
2630 PRINT V$;DD$;V$
2640 PRINT CHR$(106);HH$;S$;CHR$(107)
2650 HH$=H$:FOR J=1 TO 13:HH$=HH$+H$:NEXT
2660 S$=H$+H$:DD$=V$+B$+B$+V$+B$+B$+V$
2670 CR=0:CD=19:GOSUB 2940
2680 PRINT SPC(7);CA$;HH$;CB$;CA$;S$;CHR$(178);S$;CB$
2690 PRINT SPC(7);V$;SPC(14);V$;DD$;B$;"SPECIAL"
2700 PRINT SPC(7);V$;SPC(14);V$;CHR$(171);S$;CHR$(123);
2710 PRINT S$;CHR$(179);CHR$(95)
2720 PRINT SPC(7);V$;SPC(14);V$;DD$;B$;"SHOT BOX"
2730 PRINT SPC(7);CC$;HH$;CD$;CC$;S$;CHR$(177);S$;CD$;
2740 CD=20:CR=10:GOSUB 2940
2750 PRINT NA$;TAB(18);ZA
2760 CD=22:GOSUB 2940
2770 IF FL>1 THEN PRINT NB$;TAB(18);ZB
2780 CD=2:CR=35:PRINT RR$;:GOSUB 2940
2790 DD$=CHR$(157)+CHR$(17)+CHR$(17)
2800 FOR J=1 TO 7:PRINT MID$("HOTSHOT",J,1);DD$;:NEXT
2810 CD=1:CR=38:PRINT WW$:GOSUB 2940
2820 DD$=CHR$(157)+CHR$(17):S$="FROM"+B$
2830 FOR J=1 TO 5:PRINT MID$(S$,J,1);DD$;:NEXT
2840 PRINT BB$;
2850 FOR J=1 TO 10:PRINT MID$("MIND MOVES",J,1);DD$;:NEXT
2860 CD=18:CR=0:GOSUB 2940
2870 PRINT NC;"CUSHION BILLIARDS"
2880 GOSUB 2910:PRINT RR$;"JUST A"
2890 PRINT"MOMENT":PRINT"PLEASE"
2900 RETURN
2910 CR=0:CD=20:GOSUB 2940
2920 FOR J=1 TO 3:PRINT B7$:NEXT
2930 GOSUB 2940:RETURN
2940 PRINT HM$:IF CD=0 THEN 2960
2950 FOR K=1 TO CD:PRINT CHR$(17);:NEXT
2960 IF CR=0 THEN 2980
2970 FOR K=1 TO CR:PRINT CHR$(29);:NEXT
2980 RETURN
2990 PRINT HC$
3000 PRINT TT$;SPC(3);"OPTION MENU":PRINT
3010 PRINT"1 - 1 PLAYER (JUST YOU)"
3020 PRINT"2 - 2 PLAYERS"
```

```
3030 PRINT"3 - YOU VS. COMPUTER"
3040 PRINT"4 - END PROGRAM"
3050 PRINT:PRINT"PRESS 1, 2, 3, OR 4"
3060 GET R$:IF LEN(R$)=0 THEN 3060
3070 IF R$<"1" OR R$>"4" THEN 3060
3080 Q=VAL(R$):ON Q GOTO 3090,3100,3170,3570
3090 NA$="YOU":NB$="":FL=1:GOTO 3260
3100 NA$="":NB$="":PRINT:PRINT:PRINT
3110 INPUT"NAME OF FIRST PLAYER";NA$
3120 IF LEN(NA$)=0 THEN 3110
3130 INPUT"NAME OF SECOND PLAYER";NB$
3140 IF LEN(NB$)=0 THEN 3130
3150 NA$=LEFT$(NA$,8):NB$=LEFT$(NB$,8)
3160 FL=2:GOTO 3260
3170 PRINT:PRINT:PRINT:PRINT"WHO SHOOTS FIRST?":PRINT
3180 PRINT"1 - YOU"
3190 PRINT"2 - COMPUTER"
3200 PRINT:PRINT"PRESS 1 OR 2"
3210 GET R$:IF LEN(R$)=0 THEN 3210
3220 IF R$<"1" OR R$>"2" THEN 3210
3230 Q=VAL(R$):ON Q GOTO 3240,3250
3240 NA$="YOU":NB$="COMPUTER":FL=3:GOTO 3260
3250 NA$="COMPUTER":NB$="YOU":FL=4
3260 PRINT:PRINT:PRINT:PRINT"TABLE SIZE?":PRINT
3270 PRINT"1 - SMALL (EASIEST GAME)"
3280 PRINT"2 - MEDIUM
3290 PRINT"3 - LARGE (HARDEST GAME)"
3300 PRINT:PRINT"PRESS 1, 2, OR 3"
3310 GET R$:IF LEN(R$)=0 THEN 3310
3320 IF R$<"1" OR R$>"3" THEN 3310
3330 TS=VAL(R$):PRINT:PRINT:PRINT
3340 PRINT"NUMBER OF CUSHIONS REQUIRED?"
3350 PRINT"PRESS 0, 1, 2, OR 3"
3360 GET R$:IF LEN(R$)=0 THEN 3360
3370 IF R$<"0" OR R$>"3" THEN 3360
3380 NC=VAL(R$):PRINT:PRINT:PRINT
3390 INPUT"NUMBER OF INNINGS (99 MAX)";NI
3400 NI=INT(NI):IF NI<1 OR NI>99 THEN 3390
3410 RETURN
3420 GET R$:IF R$<>"" THEN 3420
3430 PRINT CHR$(142);CHR$(8);HC$
3440 POKE 53280,14:POKE 53281,15:PRINT WW$
3450 TT$="H O T S H O T"
3460 PRINT:PRINT:PRINT TAB(12);TT$:PRINT:PRINT
3470 PRINT TAB(4);"COPYRIGHT 1984 DILITHIUM PRESS"
3480 PRINT:PRINT:PRINT:PRINT
3490 PRINT TAB(6);"FROM THE BOOK 'MIND MOVES'"
3500 PRINT:PRINT
3510 PRINT TAB(5);"BY TOM RUGG AND PHIL FELDMAN"
3520 FOR J=1 TO 6:PRINT:NEXT
3530 PRINT TAB(9);"PRESS A KEY TO BEGIN"
3540 GET R$:IF LEN(R$)=0 THEN 3540
3550 Q=RND(-TI)
3560 RETURN
```

```
3570 POKE V+21,0
3580 PRINT CHR$(142);CHR$(9);CHR$(154);HC$;:CLR
3590 POKE 53280,14:POKE 53281,6:GOTO 9990
9990 END

READY.
```

EASY CHANGES

1. If you prefer not to have the cue balls flash at the beginning of each player's turn, change the value of FF to zero in line 970 as follows:

   970 FF=0

2. The speed of the shot is controlled by the variables NN and U set in line 1820 after you enter your speed number from 1 to 9. If you'd like to experiment with very fast shot speeds, you can change the formulas for these variables. NN controls the number of time slices for each shot, and U the distance the ball moves during one time slice. To get really fast movement for high speed numbers, try changing U in line 1820 as follows:

   1820 NN=20+15*VAL(R$):U=1.3+VAL(R$)*1.4

   You might also want to increase NN in conjunction with this as follows:

   1820 NN=10+25*VAL(R$):U=1.3+VAL(R$)*1.4

   If you would like to slow down the animation to more precisely view the shots, try:

   1820 NN=25+10*VAL(R$):U=2+VAL(R$)*0.3

   Experiment with different formulas, if you wish.

3. Green is used for the color of the table, since billiard tables traditionally are surfaced in green felt. But, if you would like another color, the background colors are set by the POKE statements in line 2500. For a yellow table and border try:

```
2500 POKE 53280,7:POKE 53281,7:PRINT
     LB$;HC$;
```

   Consult your reference manual for the POKE arguments needed to generate different colors.

4. If you want to try an extremely difficult game variation, increase the cushion requirement up to nine. Make these changes to allow anything from zero- to nine-cushion billiards:

```
3350 PRINT"PRESS 0-9"
3370 IF R$<"0" OR R$>"9" THEN 3360
```

PROGRAMMER'S NOTEBOOK

The action graphics in HOTSHOT take advantage of the sprite graphics capabilities of the Commodore 64. Only two unique sprite shapes are used, one for the balls and one for the cross hairs. Sprites 0 and 1 are the cross hairs used during shot selection, sprites 2-4 are the three billiard balls, and sprites 5-6 are the two balls used in the scoreboard to indicate the current shooter.

Animation is accomplished by a time-slicing technique. Once the speed of the shot is determined, the variable NN is set to the number of time slices. The loop at lines 415-760 processes these time slices. During a time slice, each ball's position is incremented, collisions with the other balls and the table cushions are checked and processed. The three balls are denoted with the suffixes A, B and C. The current

cue ball (white or red) is the A ball, the other player's cue ball is the B ball, and the blue ball is the C ball. Thus, the positions and colors of the A and B balls must be switched each time the shooter changes. This is done by the subroutine at 2020.

Let's take a look at each time slice. First the X and Y positions of each ball are updated in line 420. Then the subroutine at 270 is called. This checks whether any balls are out of bounds and computes their new positions and velocities if one bounces off a rail. Special care is required to keep track of the number of cushions the A ball has hit (CC) and to implement special spins (SE, TE), if any. Now a check must be made to see if any two balls are colliding. Instead of using the sprite register (V+30) for this, the geometric relation typified by lines 440-450 is used to check for each possible collision (A with B, A with C, or B with C).

We don't have space to go into all the gory details of how new velocities are computed after a collision, but here's a brief explanation. We'll consider a collision between the A and B balls as an example. The ball with the smaller X position is denoted with the numerical suffix 3, and the other ball with the suffix 4. Their X, Y positions and velocities are set in either line 480 or 510, depending upon which ball is leftmost. Then the collision routine at line 140 is called. This moves the problem into the frame of reference of a motionless 4 ball. That is, the X and Y velocities of the 4 ball (D4, E4) are subtracted from the 3 ball. The total kinetic energy (U) is retained, since it must be conserved during the collision. Three key angles are computed: AL (the angle between the centers of the balls and the horizontal), GA (the angle between the new computed direction of the 3 ball and the horizontal), and TH (the difference between these two angles, which becomes the angle of collision). That is, if TH is near zero, the balls are hitting square-on and, as TH gets further from 0, the balls are just nicking each other. The new X and Y velocities of the colliding balls

are functions of these angles and are computed in lines 220-230. They are then normalized in lines 240-250 to conserve the total kinetic energy.

After returning from the collision routine, the 3 and 4 velocities (G3, H3, G4, H4) are given to the A and B balls as appropriate. This general technique is used for possible collisions between any two balls. After a collision, the subroutine at line 780 draws the ball locations and updates their positions.

When the A ball is involved in a collision, some special care is needed. The Boolean variable BH is used to keep track of the balls with which it has collided. If BH becomes 3, the cue ball has hit both object balls and the variable FH is then set to the number of cushions the cue ball has hit. If topspin (follow or draw) is present, it is added in line 500 then is set to zero for the remainder of the shot.

After each possible collision is processed, the sprite position vectors are updated in lines 740-750. This causes the balls to move on the screen. Then one time slice is completed and the loop at 410 is begun again for the next time slice.

Some random notes of interest. When BASIC encounters any variable in a program, it must look it up in its tables. Variables at the beginning of these tables will be found slightly faster than those farther down. Since smoother, faster motion is desired in HOTSHOT, the variables used in the collision and time-slice routines need to be defined first. Thus the routine at line 830 simply mentions these variables to force them high on the variable list. That is the only use of lines 830-890. At the beginning of the game, the Y position of the three balls is set randomly in line 1350. During the game, the [SHIFT] [COMMODORE] key combination that toggles between uppercase and lowercase is disabled in line 3430. After a normal game termination, it is re-enabled in line 3580 with PRINT CHR$(9). If you stop the game abnormally, you may need to rectify this by hand, so

to speak. If you want to add some of your own coding for the Suggested Projects, you can do it between lines 3590 and 9990. Line 3590 already branches to 9990 as required.

MAIN ROUTINES

140-260	Computes new velocities for ball-to-ball collisions.
270-390	Computes new velocities for ball-to-cushion collisions.
400-770	Portrays each shot graphically.
780-820	Updates ball positions and displays them.
830-970	Initializes variables.
980-1150	Mainline routine — calls routines to get game parameters, initializes screen, plays game, acknowledges winner, and inquires about a new game.
1160-1230	Sprite data for the balls.
1240-1290	Sprite data for the cross hairs.
1300-1320	Sets sprite colors.
1330-1710	Sets up first shot, then controls all shots for the entire game.
1720-1850	Controls the routines to get each player's shot selection (alignment, special shots, and speed).
1860-1900	Prints message when shot is no good.
1910-2010	Subroutine to get trick shot information.
2020-2060	Switches sprite position and color values for the red and white balls.
2070-2120	Subroutine to determine computer's shots.
2130-2250	Subroutine to get shot alignment position.
2260-2490	Sets variable information depending on table size.
2500-2900	Displays the screen containing the billiards table and scoreboard.
2910-2930	Clears the lower left message area and prepares to print messages there.
2940-2980	Moves cursor CR columns right and CD rows down.
2990-3410	Gets game parameters from the user.
3420-3560	Displays introductory title and waits for a player to hit a key to continue.

3570-3590	Resets screen colors and computer parameters at conclusion of the game.
9990	Ends program.

MAIN VARIABLES

XA, XB, XC	X position of the cue ball and two object balls.
YA, YB, YC	Y position of the cue ball and two object balls.
DA, DB, DC	X velocities of the three balls.
EA, EB, EC	Y velocities of the three balls.
X3, Y3, D3, E3	X, Y position and X, Y velocity of leftmost ball in a collision.
X4, Y4, D4, E4	As above for the rightmost ball.
G3, H3, G4, H4	Resultant X, Y velocities for leftmost, rightmost balls after a collision.
X, Y	Temporary X, Y position.
J, K, M, Q	Loop and work variables.
LE, RE, UE, BE	Positions of left, right, upper, bottom edges of table.
C	Collision window (i.e., ball diameter squared).
DD, EE	Difference of X, Y velocities of two colliding balls.
U	Total kinetic energy (or velocity).
PI, HP	Value of pi and pi/2.
AL	Angle between the centers of two colliding balls.
GA	Angle of moving ball colliding with stationary one.
TH	Collision angle (AL-GA).
SI, CI, SA, CA	Sine, cosine of TH and AL.
RA	Ratio with which velocities are normalized.
V	Memory address of VIC graphics chip (53248).
NN	Duration (number of time slices) for a given shot.
NI, NC	Number of innings, number of cushions required.
TS	Table size (1 = small, 2 = medium, 3 = large).

BH	Object balls hit (0 = none, 1 = one only, 2 = other only, 3 = both just hit, 4 = both already hit).
CC	Cushion count (number of rails hit by cue ball).
FH	Number of rails hit by cue ball when second object ball is struck (set to −1 if both object balls not hit).
TE	Top English (0 = none, neg = draw, pos = follow).
SE	Side English (0 = none, neg = left, pos = right).
ZA, ZB	Scores of player 1 and player 2.
FL	Flag (1 = one player game, 2 = two humans, 3 = computer plays second, 4 = computer plays first).
FF	Flash flag (0 = flashing off, 1 = flashing on).
P3, P4, P5, P6, P7	Poke locations for balls (sprites 2-6).
XP, YP, ZP	Arrays of (X, Y, Extended X) values for the sprite locations of the alignment cross hairs.
NP	Number of values in above arrays.
ADD, PAGE, BYTE	Address, memory page, byte value of sprite data.
CD, CR	Values to move cursor down, right.
TI	Internal system timer for random number seed.
R$	User reply string.
B$,B7$	Strings of one, seven blank characters.
RT$	CHR$ argument for carriage return.
TT$	String of program name.
HM$, HC$	CHR$ argument for home, clear screen.
NA$, NB$	String names of player 1, player 2.
BB$, LB$	CHR$ arguments for text colors dark blue, light blue.
RR$, WW$	CHR$ arguments for text colors red, white.
S$, SS$, V$, D$ H$, CA$, CB$, CC$, CD$, HH$, DD$	Temporary strings.

SUGGESTED PROJECTS

1. If you want to speed up the animation, try writing the collision and time-slice routines in machine language.

Or perhaps try some simplifications in BASIC to these routines. This will make them run faster but probably at the expense of less realism in the collision angles and velocities.

2. Allow for a third player who will use the blue ball as his cue ball.
3. Try to improve the subroutine that determines the computer's shots. See if you can compute what a shot will look like and adjust how the computer selects the angle and speed for its shots. Also, you might allow it to use English on its shots, which is not done in the current subroutine.
4. You might enjoy implementing some different scoring algorithms. You could insist that, for a shot to score, the number of required cushions be met exactly. Perhaps more points would be awarded the closer a player comes to the number of cushions required. Another possibility is to have the shooter predict the number of cushions his shot will hit, then score him according to how close he comes to his prediction.

4.

QUIKWIT

Who was known as the "Schnozzola"? Is there an animal with both tusks and horns? Which chess piece cannot move backwards?

If questions like these stimulate your brain, you'll have a ball playing QUIKWIT. Compete against a friend in this race to answer trivia-type multiple-choice questions. Play with teams when there are more than two of you. And if you're by yourself, you can have a whale of time playing a special solo version against the computer.

This game has many features, including the ability to adjust the difficulty of the questions and to handicap the players when their skill levels are unequal. We provide questions in four categories: Entertainment, Sports, Science, and General Information. You can choose which ones you want each time you play — even using several categories at once if you desire. And we'll show you how to make up and enter your own questions, so you can challenge your friends with your own devilish creations.

EXCITING NEWS! The beauty of QUIKWIT is the quality of the questions. They can be clever, entertaining, even funny, but they're always interesting. We commissioned Penguin Dave, a legendary popular culture and trivia expert, to write them. And now, when you use up all the questions that come with the program disk, you can order more written by the master. These new disks each contain over a thousand questions at a bargain price. They are

oriented toward both adults and children. If you're interested, please see the enclosed order form.

By the way, the answers to the three original questions: Jimmy Durante, no, and the pawn.

RULES

QUIKWIT is essentially a race to correctly answer questions as fast as possible. Let's discuss the basic two-player game first, then we'll describe team play and single-player play.

The object of the game is to score goals. The first player to get five wins. A goal is scored by tugging (some say sucking) the white game ball to your end of the court. The players have only a fixed amount of time to answer each question before the next one is presented. When you answer a question correctly, the game ball moves toward your end of the court. The distance of this movement is greater the faster the question is answered. If you answer incorrectly, the ball moves toward your opponent's end a fixed penalty distance (independent of time).

The questions are displayed in multiple-choice form. You can choose to have a maximum of five possible answers displayed with each question. Only one answer is correct.

When a player answers incorrectly, the timer continues and the other player is given a chance to answer the question. Strategy enters: should you risk an unsure guess for the potential profit? Sometimes you must make a guess quickly to prevent your opponent from scoring a goal if the ball is near his end of the court.

Team play is a lot of fun and can be done in several different ways. Some aficionados like a noisy, lively game in which any team member can speak up or yell out an answer for the captain to enter. Some allow any player on the team to hit an answer key if he thinks he knows the right answer. Others prefer a more subdued game in which team players rotate to answer questions. Usually then, teammates are not allowed to help.

In single-player action, the computer takes one end of the court. Of course, when a question is asked, only you can answer. (After all, the computer knows the right answer.) Thus a new wrinkle is added. While you're thinking, the game ball moves toward the computer's end. This forces you to answer some questions right or you'll find the computer scoring goals by default. The speed of the movement toward the computer's goal is set by the difficulty level you select. No rest for the weary.

HOW TO USE IT

The following instructions apply to the disk version of QUIKWIT. Appendix B gives you additional instructions for the cassette version that you can type in yourself. As we explained in telling you How to Use This Book, this is the one program that may work less reliably with a cassette drive than with a disk drive.

By now, you probably realize that the program starts by displaying its title screen. Press a key and you will be served the QUIKWIT option menu, which has six selections for your command. Notice that the screen display is now in uppercase/lowercase mode and, indeed, will stay there for the remainder of the game.

Briefly, here's what each option does. Option 1 initiates the one-player game that pits you against the computer. Option 2 selects the game version for two players (or teams) playing against each other. Option 3 is for choosing other data files (or combinations of them) from the disk. Option 4 is to quickly adjust the difficulty of the game to suit your taste (and skill level). Option 5 shows you where you are in each active data file so that you can continue from that point in the next game session if you wish. And good old option 6 ends the game so someone else (your parent, child, friend, teacher, whoever) can get at the computer or so you can go to sleep. Now let's discuss each option more thoroughly.

When you press 1 to start the one-player game, the QUIKWIT game board appears, the disk whirs, the computer tells you that it's initializing data files and reading in the first set of questions.

Note the thin rectangular court with the three balls on it. This is the basic play area. You are on the left side in red while the computer is on the right side in blue. You try to get the white game ball onto your red ball; the computer tries to get it onto his blue ball. A goal is scored whenever one of you succeeds, and the first to get five goals wins the game.

On the right side of the board is a vertical strip labeled TIME BALL with another white ball near the top. When a question is asked, this ball starts falling. You have until it hits bottom to answer.

Questions are presented with a list of up to five possible answer choices. Only one is the correct answer. Your task, of course, is to decide as quickly as possible which one that is. Use the number keys 1 to 5 at the upper left of the keyboard to make your guess. (Hopefully you'll *know* the answer, not *guess* it.)

A special rule comes up here. Some questions have a possible answer like, "all of the other choices" or "none of the other choices." If one of these can be the correct answer, it takes precedence over the other choices. For example, if the question is "Which of these countries is in Europe?" and the four possible answers are "1 — France, 2 — Spain, 3 — All of the other choices, 4 — Belgium," then 3 is the correct answer.

As each question appears, the background flashes black, you're told to get ready, and the background flashes back to gray. The question appears instantly along with its category and its number within that category. You now have to answer the question as accurately and quickly as possible. Note that the time ball is falling and that the white game ball is moving toward the computer's end. If you do nothing, the computer can score without your even hazarding a guess.

When you think you know the answer (or want to risk a guess), hit the appropriate number key. Immediately the

message line below the question indicates which key is pressed and whether or not the answer is correct. The number of the correct answer is outlined in black on the question board. If you answer right, you hear a cheerful two-note beep, and the game ball moves toward your end. The faster you answer correctly, the more it moves. If you answer incorrectly, you hear a short razzing noise, and the game ball jumps away from you.

When someone scores a goal, the game ball flashes, a beeping sound occurs, the scorer is acknowledged, and the scoreboard is updated. Then the game ball moves back to midcourt for the next point. When one side scores five goals, the game ends. The winner is congratulated (hope it's you), a bell sound is played, and you're asked to hit any key to continue. When you do so, you find yourself right back at the option menu.

Selecting option 2 from the main menu gets you into the two-player game. Questions are asked as described in the one-player game, but the scoring and game play are somewhat different.

First you have to enter the name of each player (or team). Eight characters is the maximum length; the names are trimmed to eight if you enter more.

One player takes the red or left side of the board, and the other player gets the blue or right side. This game is a race; each player tries to answer questions correctly sooner than his opponent. The red side uses the five number keys as explained in the one-player game. The blue side, however, needs five different keys. We chose the four function keys (on the right of the keyboard) plus the cursor right key at the extreme lower right of the keyboard. The function keys are unfortunately labeled f1, f3, f5, and f7, but we'll use them to represent 1, 2, 3, and 4 respectively. And we chose the cursor key to represent 5. If you have trouble visualizing this, try taping little paper labels on the appropriate keys with the numerals from 1 to 5. We find most people have no problem after a few games.

The white game ball does not move as the time ball falls, as it does in the one-player game. When a player makes a

guess, the computer indicates who made the guess and what that guess was. If the guess is right, the game ball moves toward a goal for him.

But if the guess is wrong, the time ball continues to fall, a tone sounds, and the other player gets a shot at answering the question. This tone is a razzing noise if red misses and a continuous note if blue misses. All penalties for missing and rewards for correct answers are like those described in the one-player game. If the time ball reaches the bottom, no further scoring is possible during that question.

Again, the game is to five goals and is in other respects like the one-player game.

You use options 3 and 4 on the main menu before you select one of the two game versions just described.

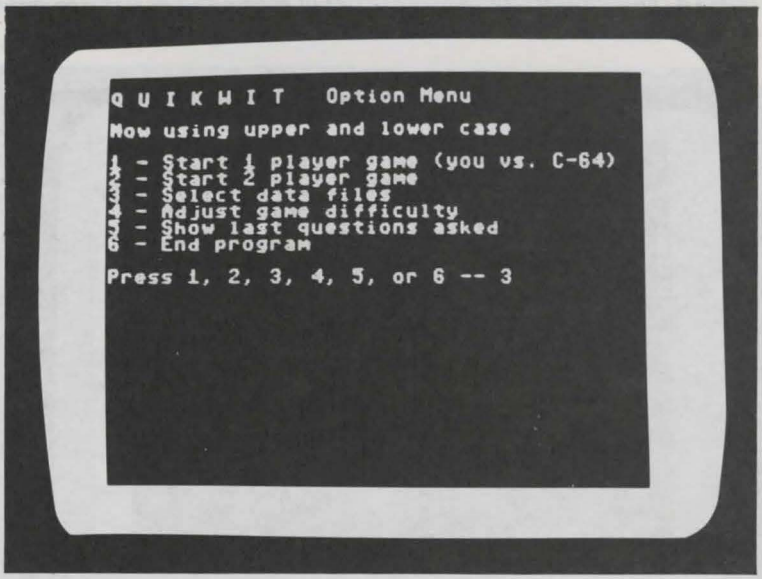
With option 3, you select which data files the questions are drawn from during the game, and you choose which questions are asked from that file. When you select this option, you're shown a list of the available data files. The computer wants to know which files to turn on and where to start in each one. So you're asked file by file what question number you want to start with. Simply input the desired question number, then press the [RETURN] key. (See option 5 for a discussion of how you might know which choices to make under option 3.) If you want to use the file but don't care where the questions start, input −1, and the computer will start randomly within that file. If you don't want to use the file at all, input 0 (zero), and that file will be turned off. Note that you must turn on at least one file and can have up to three files active at once.

You use option 4 to quickly adjust the difficulty level of the game. By pressing 1, 2, or 3, you get an easy, medium, or hard version of the game. These settings are only relative, of course, and in the Easy Changes we'll show you how to make many more and much finer-tuned difficulty-level adjustments. Option 4 changes three things — the number of possible answer choices displayed, the speed at which the time ball falls, and the speed at which the game ball moves during the one-player game.

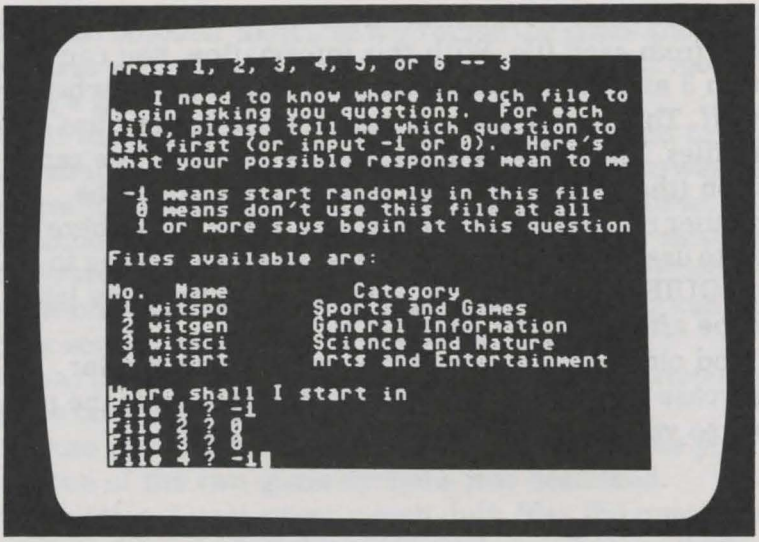
Option 5 from the main menu shows you the last question asked from each file. With this information, you can use option 3 at a subsequent session to start exactly where you left off. This way, there will be no waste in your use of the data files. When different games are played in the same session (that is without ever ending QUIKWIT), the computer remembers where it was in each file, so there is no need to use this option. It is used when you are going to exit from QUIKWIT entirely but intend to resume play later (maybe after a night's sleep.)

Good old option 6 restores the machine to regular uppercase mode with standard colors, then ends the program so you can get that good night's sleep.

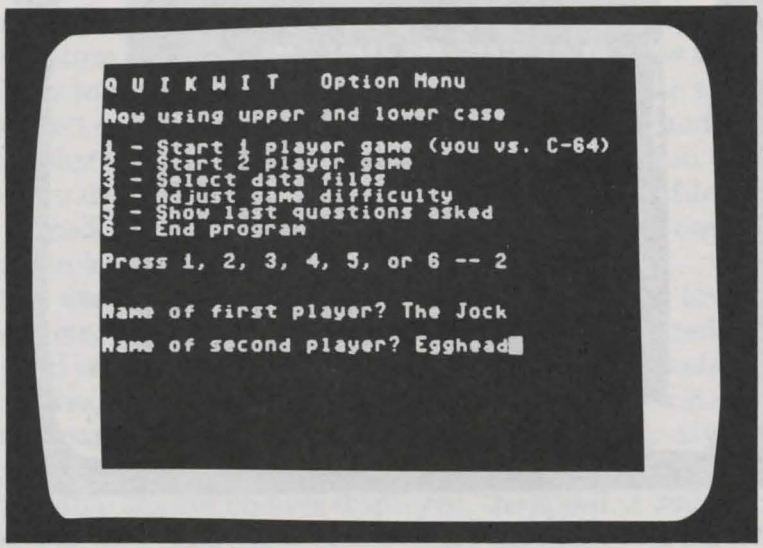
SAMPLE RUN



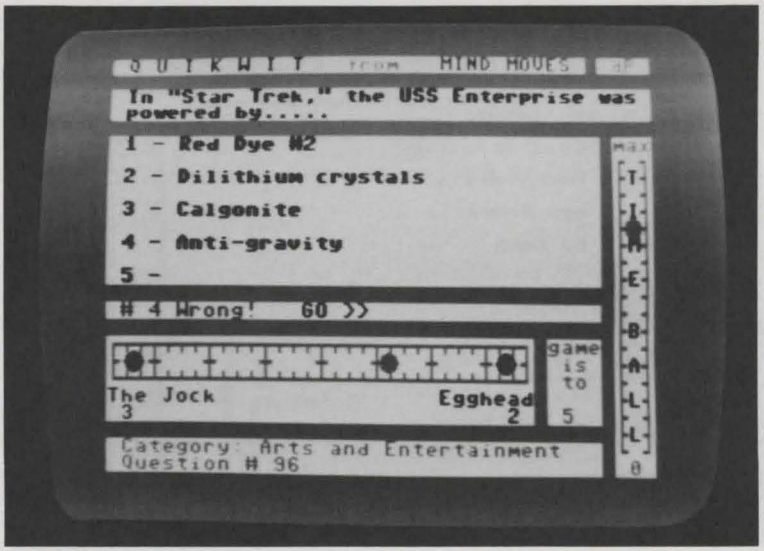
After pressing a key to see the option menu, the two players select option 3 to control the data files for their game.



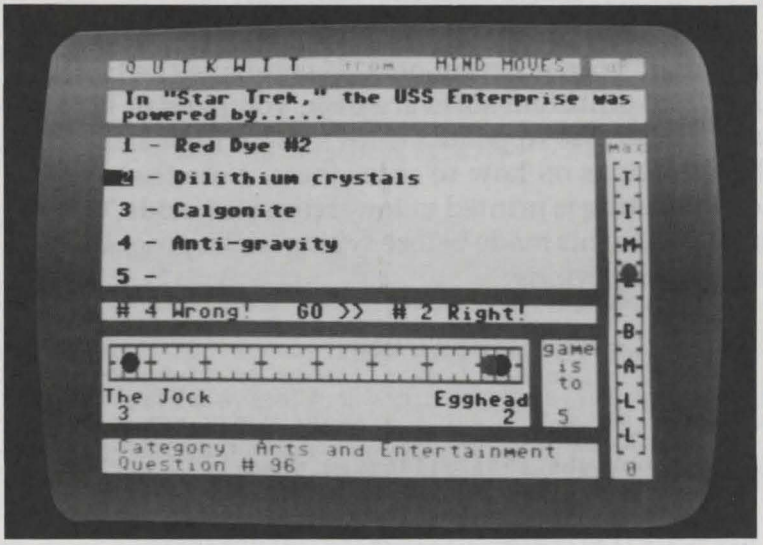
One player is a sports nut and the other a movie and TV fan. They compromise and decide to use questions from each of their two specialties. They turn off the questions on science and general information and select a random start for the categories of sports and entertainment.



The two-player version of QUIKWIT is selected, and the players enter their nicknames in preparation for play.



The score is 3 to 2 in favor of The Jock when an entertainment question appears. The Jock tries a quick answer of choice 4 (the time ball has barely fallen), but he is incorrect. The Jock is penalized as the game ball gets closer to a score for Egghead, and the computer signals Egghead that he has a guess coming.



A short time later, Egghead guesses choice 2, which the computer shows to be correct. The game ball gets closer to Egghead's side and he is now close to a goal.

Eventually Egghead wins the match 5 goals to 4. The computer waits for a key to be pressed to return to the option menu.

PROGRAM LISTING

This listing is the program only. If you have the book without the dilithium software disk, you have to type in data files also. See Appendix C for a list of typical questions and instructions on how to enter them. Note that the following listing is printed in lowercase text mode. Get your computer into this mode before typing in the program. You can do this by typing:

PRINT CHR$(14) [RETURN]

```
100 rem: quikwit
110 rem: copyright 1984 dilithium press
120 rem: by phil feldman and tom rugg
130 a1$="1":a2$="2":a3$="3":a4$="4":a5$="5"
140 b1$=chr$(133):b2$=chr$(134):b3$=chr$(135)
150 b4$=chr$(136):b5$=chr$(29)
160 sd=54276:v=53248
170 sf=1
```

```
180 dt=0.8
190 dx=0.22
200 py=30
210 xc=136
220 fr=0.5
230 hc$=chr$(147):hm$=chr$(19):bk$=chr$(144)
240 ww$=chr$(5):rr$=chr$(28):bb$=chr$(31):pr$=chr$(156)
250 fz=100
260 dim nn(24),cn(24)
270 nf=4
280 dim f$(nf),c$(nf),dc(nf),qn(nf),aq(nf)
290 dim a$(2,24),b$(5,24),aa(5)
300 f$(1)="witspo"
310 f$(2)="witgen"
320 f$(3)="witsci"
330 f$(4)="witart"
340 for k=1 to nf:dc(k)=0:aq(k)=-1:next
350 dc(1)=0:dc(2)=-1:dc(3)=0:dc(4)=0
360 nc=4
370 x$=chr$(32):n$="":cr$=n$:cd$=n$:sp$=n$
380 for k=1 to 40:cr$=cr$+chr$(29):cd$=cd$+chr$(17)
390 sp$=sp$+x$:next
400 sa=0:sb=0
410 sg=5
420 ar$=chr$(94)+chr$(94)+chr$(94)
430 for j=0 to 24:poke sd-4,0:next
440 if sf=0 then 470
450 poke sd-4,0:poke sd-3,18:poke sd-2,12:poke sd-1,7
460 poke sd+1,9:poke sd+2,169:poke sd+20,15:poke sd,0
470 for k=1 to 9:close k:close 15
480 gosub 570:gosub 670:gosub 4070
490 gosub 3440:gosub 1780:q=fre(0):gosub 3230
500 gosub 2940:qq=0
510 qq=qq+1:if qq>24 then 500
520 gosub 2700:on np gosub 710,1010
530 q=fre(0)
540 for p=1 to 1000:next
550 if sa>=sg or sb>=sg then 2270
560 goto 510
570 poke v+21,0:restore
580 poke v+23,0:poke v+28,0:poke v+29,0:poke v+16,0
590 add=832:page=add/64
600 for j=add to add+29:read byte:poke j,byte:next
610 for j=add+30 to add+63:poke j,0:next
620 data 0,60,0,0,255,0,1,255,128,1,255,128,3,255,192
630 data 3,255,192,1,255,128,1,255,128,0,255;0,0,60,0
640 for j=2040 to 2043:poke j,page:next
650 for j=1 to 4:poke v+38+j,val(mid$("1126",j,1)):next
660 return
670 dim pv(24):for j=0 to 24:read pv(j):next
680 data 200,151,0,0,0,9,249,0,0,0,0,0,0,0,0
690 data 255,200,0,0,17,153,255,1,255,81,159
700 return
710 a=0
```

```
720 y=y+dt:x=x+dx:get r$
730 if r$=a1$ then a=1:goto 810
740 if r$=a2$ then a=2:goto 810
750 if r$=a3$ then a=3:goto 810
760 if r$=a4$ then a=4:goto 810
770 if r$=a5$ then a=5:goto 810
780 poke v+1,y:if y>233 then 900
790 poke v+2,x:if x>=244 then 920
800 goto 720
810 if ca=a then 860
820 poke sd,129:print rr$;"#";a;"Wrong!":gosub 2460
830 x=x+py:if x>244 then x=244
840 poke v+2,x:if x>=244 then 920
850 goto 1740
860 print rr$;"#";a;"Right!":gosub 2490:gosub 2460
870 x=x-(250-y)*fr:if x<28 then x=28
880 poke v+2,x:if x<=28 then 970
890 goto 1740
900 cd=15:cr=13:gosub 3180
910 print ww$;"Time's up":gosub 2460:goto 1740
920 x=244:poke v+2,x:cd=15:cr=13:gosub 3180
930 print ww$;"I score":gosub 2460:gosub 1750
940 gosub 1750
950 sb=sb+1:cd=21:cr=30-len(str$(sb))
960 gosub 3180:print bb$;sb:x=xc:poke v+2,x:goto 1740
970 x=28:poke v+2,x:cd=15:cr=13:gosub 3180
980 print ww$;"You score":gosub 1750
990 sa=sa+1:cd=21:cr=0:gosub 3180
1000 print rr$;sa:x=xc:poke v+2,x:goto 1740
1010 a=0:b=0
1020 y=y+dt:get r$
1030 if r$=a1$ then a=1:goto 1150
1040 if r$=a2$ then a=2:goto 1150
1050 if r$=a3$ then a=3:goto 1150
1060 if r$=a4$ then a=4:goto 1150
1070 if r$=a5$ then a=5:goto 1150
1080 if r$=b1$ then b=1:goto 1380
1090 if r$=b2$ then b=2:goto 1380
1100 if r$=b3$ then b=3:goto 1380
1110 if r$=b4$ then b=4:goto 1380
1120 if r$=b5$ then b=5:goto 1380
1130 poke v+1,y:if y>233 then 900
1140 goto 1020
1150 if ca=a then 1580
1160 poke sd,129
1170 print rr$;"#";a;"Wrong!";bb$;left$(sp$,3);
1180 print"GO >>";left$(sp$,2);
1190 x=x+py:if x>244 then x=244
1200 poke v+2,x:if x>=244 then 1680
1210 y=y+dt:get r$
1220 if r$=b1$ then b=1:goto 1290
1230 if r$=b2$ then b=2:goto 1290
1240 if r$=b3$ then b=3:goto 1290
1250 if r$=b4$ then b=4:goto 1290
```

```
1260 if r$=b5$ then b=5:goto 1290
1270 poke v+1,y:if y>233 then 910
1280 goto 1210
1290 if ca=b then 1340
1300 poke sd,17:print bb$;"#";b;"Wrong!"
1310 x=x-py:if x<28 then x=28
1320 poke v+2,x:if x<=28 then 1620
1330 gosub 2460:goto 1740
1340 print bb$;"#";b;"Right!":gosub 2490
1350 x=x+(233-y)*fr:if x>244 then x=244
1360 poke v+2,x:if x>=244 then 1680
1370 gosub 2460:goto 1740
1380 if ca=b then print left$(sp$,20);:goto 1340
1390 poke sd,17:print rr$;left$(sp$,12);"<< GO";
1400 print bb$;left$(sp$,3);"#";b;"Wrong!"
1410 x=x-py:if x<28 then x=28
1420 poke v+2,x:if x<=28 then gosub 2460:goto 1620
1430 y=y+dt:get r$
1440 if r$=a1$ then a=1:goto 1520
1450 if r$=a2$ then a=2:goto 1520
1460 if r$=a3$ then a=3:goto 1520
1470 if r$=a4$ then a=4:goto 1520
1480 if r$=a5$ then a=5:goto 1520
1490 poke v+1,y
1500 if y>233 then cd=15:cr=1:gosub 3180:goto 910
1510 goto 1430
1520 cd=15:cr=1:gosub 3180
1530 if ca=a then 1580
1540 poke sd,129:print rr$;"#";a;"Wrong!"
1550 x=x+py:if x>244 then x=244
1560 poke v+2,x:if x>=244 then gosub 2460:goto 1680
1570 gosub 2460:goto 1740
1580 print rr$;"#";a;"Right!":gosub 2490
1590 x=x-(233-y)*fr:if x<28 then x=28
1600 poke v+2,x:if x<=28 then gosub 2460:goto 1620
1610 gosub 2460:goto 1740
1620 x=28:poke v+2,x:gosub 2460
1630 sa=sa+1:cd=21:cr=0:gosub 3180
1640 print rr$;sa;left$(sp$,4);"<< Score"
1650 gosub 1750
1660 cd=21:cr=7:gosub 3180:print left$(sp$,11)
1670 x=xc:poke v+2,x:goto 1740
1680 x=244:poke v+2,x:gosub 2460
1690 sb=sb+1:cd=21:cr=16:gosub 3180
1700 print bb$;"Score >>":cr=30-len(str$(sb))
1710 gosub 3180:print sb:gosub 1750
1720 cd=21:cr=15:gosub 3180:print left$(sp$,9)
1730 x=xc:poke v+2,x:goto 1740
1740 return
1750 for j=1 to 5:poke sd,33:poke v+21,13
1760 for p=1 to 300:next:poke v+21,15:poke sd,0
1770 for p=1 to 300:next:next:return
1780 poke 53280,8:poke 53281,15:print chr$(14);hc$;
1790 print x$;bb$;"Q U I K W I T";
```

```
1800 print ww$;left$(sp$,3);"from";left$(sp$,3);
1810 print rr$;"MIND MOVES":print chr$(129);
1820 gosub 2520:print:print:gosub 2520
1830 for j=1 to 9:print:next
1840 print chr$(18);left$(sp$,36)
1850 print:print chr$(18);left$(sp$,36)
1860 print chr$(19);chr$(18);left$(cd$,5);left$(cr$,36);
1870 for j=1 to 19:print chr$(18);x$;chr$(157);
1880 print chr$(17);:next
1890 print chr$(18);x$;chr$(19)
1900 print left$(cd$,4);left$(cr$,37);
1910 p$=chr$(157)+chr$(157)+chr$(157)+chr$(17)
1920 print chr$(156);"max";p$;
1930 print chr$(176);x$;chr$(174);p$;
1940 for j=1 to 16:print chr$(171);x$;chr$(179);p$;:next
1950 print chr$(173);x$;chr$(189);p$;
1960 print x$;"0";chr$(157);chr$(145);
1970 for j=1 to 16:print chr$(145);:next
1980 print chr$(31);
1990 for j=1 to 4:print mid$("TIME",j,1);chr$(157);
2000 print left$(cd$,2);:next:print chr$(17);
2010 for j=1 to 4:print mid$("BALL",j,1);chr$(157);
2020 print left$(cd$,2);:next
2030 print chr$(19);left$(cd$,17);chr$(156);chr$(176);
2040 for j=1 to 29:print chr$(178);:next:print chr$(174)
2050 print chr$(171);x$;x$;chr$(123);
2060 for j=1 to 6:print x$;x$;x$;chr$(123);:next
2070 print x$;x$;chr$(179):print chr$(145);chr$(173);
2080 for j=1 to 29:print chr$(177);:next:print chr$(189)
2090 print rr$;na$;left$(cr$,31-len(nb$)-len(na$));bb$;nb$
2100 print chr$(145);rr$;sa;
2110 cd=21:cr=30-len(str$(sb)):gosub 3180
2120 print bb$;sb
2130 print chr$(129);chr$(18);left$(sp$,36);chr$(19);
2140 print left$(cd$,17);left$(cr$,31);chr$(129);chr$(18);
2150 for j=1 to 5:print x$;chr$(157);chr$(17);:next
2160 print chr$(146);
2170 for j=1 to 5:print chr$(145);:next:print chr$(29);
2180 print chr$(5);"game":print spc(33);"is"
2190 print chr$(145);spc(33);"to"
2200 print spc(32);chr$(17);sg
2210 print chr$(19);spc(35):print chr$(129);chr$(18);
2220 print x$;chr$(146);chr$(156);" dP"
2230 x=xc:poke v+2,x:poke v+3,193:poke v+4,28:poke v+5,193
2240 poke v+6,244:poke v+7,193:poke v+16,1:y=97
2250 poke v+0,64:poke v+1,y:poke v+21,15
2260 return
2270 gosub 2530:cd=23:gosub 3180
2280 if sa>=sg then print rr$;ar$:print"Winner";
2290 if sb<sg then 2320
2300 print bb$;left$(sp$,28);ar$
2310 print left$(sp$,25);"Winner";
2320 get r$:if r$<>n$ then 2320
2330 gosub 2590:print ww$;"Game over";
```

```
2340 print bk$;x$;"-";x$;
2350 print pr$;"Hit any key to continue"
2360 for j=0 to 24:poke sd-4,pv(j):next
2370 poke sd,21:poke sd+14,17
2380 for j=1 to (50*rnd(1)):next
2390 poke sd,20:poke sd+14,16
2400 for j=1 to 300:next
2410 for j=1 to 9:close j:next:close 15
2420 get r$:if r$=n$ then 2370
2430 for j=1 to nf:if dc(j)=0 then 2450
2440 dc(j)=aq(j)+1:if dc(j)<1 then dc(j)=1
2450 next:poke v+21,0:sa=0:sb=0:goto 490
2460 cd=3+ca+ca:cr=0:gosub 3180
2470 print bk$;chr$(18);ca;chr$(146)
2480 poke sd,0:return
2490 for p=1 to 2
2500 poke sd,33:for j=1 to 100:next:poke sd,0
2510 for j=1 to 100:next:next:return
2520 print chr$(18);sp$;chr$(146);:return
2530 for q=23 to 24:cd=q:cr=0:gosub 3180
2540 print left$(sp$,36);:next:return
2550 cr=0:for cd=2 to 3:gosub 3180
2560 print left$(sp$,40);:next
2570 for cd=5 to 13:gosub 3180
2580 print left$(sp$,36);:next:return
2590 cd=15:cr=0:gosub 3180:print left$(sp$,33)
2600 cr=1:gosub 3180:return
2610 gosub 3670
2620 print:print"No.";tab(5);"Name";
2630 print tab(14);"Last Question Asked"
2640 for k=1 to nf:print k;f$(k);tab(15);
2650 if aq(k)>=0 then print aq(k):goto 2670
2660 print"File not used yet"
2670 next:print:print"Hit any key to continue"
2680 get r$:if r$=n$ then 2680
2690 goto 3440
2700 poke 53281,0:y=97:poke v+1,y:gosub 2590
2710 print left$(sp$,10);ww$;"Get ready !"
2720 gosub 2530:cd=23:cr=1:gosub 3180
2730 print pr$;"Category: ";c$(cn(qq))
2740 cd=24:gosub 3180
2750 print ww$;"Question #";nn(qq);
2760 aq(cn(qq))=nn(qq):gosub 2550
2770 cd=2:cr=1:gosub 3180:t$=a$(1,qq):gosub 3080
2780 print bk$;tt$
2790 cd=3:gosub 3180:t$=a$(2,qq):gosub 3080:print tt$
2800 mc=0:for k=1 to 5
2810 if asc(left$(b$(k,qq),1))<>104 then mc=k
2820 next:if nc<mc then mc=nc
2830 for k=1 to 5:aa(k)=k:next
2840 for k=mc to 2 step -1
2850 j=int(k*rnd(1))+1
2860 q=aa(k):aa(k)=aa(j):aa(j)=q
2870 next:for k=1 to mc:if aa(k)=1 then ca=k
```

```
2880 next:cr=0
2890 for k=1 to 5:t$=b$(aa(k),qq):gosub 3080
2900 if k>mc then tt$=n$
2910 cd=3+k+k:gosub 3180:print k;"-";x$;tt$:next
2920 get r$:if r$<>n$ then 2920
2930 gosub 2590:poke 53281,15:return
2940 gosub 2530:cd=23:cr=1:gosub 3180
2950 print bk$;"Excuse me for a few seconds,"
2960 cd=24:gosub 3180
2970 print"I'm reading in questions.";
2980 k=0:for q=1 to 24
2990 k=k+1:if k>nf then k=0:goto 2990
3000 if dc(k)=0 then 2990
3010 input#k,a$(1,q):input#k,a$(2,q)
3020 for j=1 to 5:input#k,b$(j,q):next
3030 qn(k)=qn(k)+1:nn(q)=qn(k):cn(q)=k
3040 if st=0 then 3070
3050 close k:open k,8,k+2,"0:"+f$(k)+",s,r":input#k,r$
3060 qn(k)=0
3070 next:gosub 2530:return
3080 tt$=n$:q=asc(left$(t$,1)):if q=104 then 3170
3090 if q<>105 then tt$=t$:goto 3170
3100 for p=2 to len(t$):q=asc(mid$(t$,p,1))
3110 if q=100 then tt$=tt$+",":goto 3160
3120 if q=101 then tt$=tt$+":":goto 3160
3130 if q=102 then tt$=tt$+";":goto 3160
3140 if q=103 then tt$=tt$+chr$(34):goto 3160
3150 tt$=tt$+chr$(q)
3160 next
3170 return
3180 print hm$;:if cd=0 then 3200
3190 print left$(cd$,cd);
3200 if cr=0 then 3220
3210 print left$(cr$,cr);
3220 return
3230 cd=23:cr=1:gosub 3180
3240 print bk$;"Just a moment please, I'm"
3250 cd=24:gosub 3180
3260 print"initializing my disk files.";
3270 for j=1 to 9:close j:next:close15
3280 open 15,8,15:close 15:open 15,8,15
3290 for k=1 to nf:if dc(k)=0 then 3430
3300 q=dc(k):if q<0 then q=int(fz*rnd(1))+1
3310 open k,8,k+2,"0:"+f$(k)+",s,r"
3320 input#15,w$,r$:if w$="00" then 3360
3330 cd=15:cr=0:gosub 3180:print bk$;"Fatal error, file";
3340 print k;"-";x$;r$
3350 gosub 2530:cd=21:cr=0:gosub 3180:end
3360 input#k,c$(k):if q=1 then qn(k)=0:goto 3430
3370 qn(k)=q-1:p=0:for j=1 to qn(k):p=p+1
3380 for m=1 to 7:input#k,r$:next
3390 if st=0 then 3420
3400 close k:open k,8,k+2,"0:"+f$(k)+",s,r":input#k,r$
3410 p=0
```

```
3420 next:qn(k)=p
3430 next:return
3440 print hc$;ww$;chr$(14):poke 53280,4:poke 53281,6
3450 print"Q U I K W I T";spc(3);"Option Menu":print
3460 print"Now using upper and lower case":print
3470 print"1 - Start 1 player game (you vs. C-64)"
3480 print"2 - Start 2 player game"
3490 print"3 - Select data files"
3500 print"4 - Adjust game difficulty"
3510 print"5 - Show last questions asked"
3520 print"6 - End program"
3530 print:print"Press 1, 2, 3, 4, 5, or 6";
3540 get r$:if len(r$)=0 then 3540
3550 if r$<"1" or r$>"6" then 3540
3560 q=val(r$):print x$;"--";q
3570 on q goto 3580,3590,3760,3930,2610,4210
3580 na$="You":nb$="Computer":np=1:goto 3660
3590 np=2:na$=n$:nb$=n$:print:print
3600 input"Name of first player";na$
3610 if len(na$)=0 then 3600
3620 print
3630 input"Name of second player";nb$
3640 if len(nb$)=0 then 3630
3650 na$=left$(na$,8):nb$=left$(nb$,8)
3660 return
3670 print:print"Files available are:":print
3680 print"No.";tab(5);"Name";tab(18);"Category"
3690 close2:close 15:open 15,8,15:for k=1 to nf
3700 open 2,8,2,"0:"+f$(k)+",s,r"
3710 input#15,w$,r$:if w$="00" then 3740
3720 print:print"Fatal error on file";k
3730 print r$:end
3740 input#2,c$(k):print k;f$(k);tab(15);c$(k)
3750 close2:next:print:close 15:return
3760 print:print spc(3);"I need to know where in each file to"
3770 print "begin asking you questions.  For each"
3780 print"file, please tell me which question to"
3790 print"ask first (or input -1 or 0).  Here's"
3800 print"what your possible responses mean to me":print
3810 print" -1 means start randomly in this file"
3820 print"  0 means don't use this file at all"
3830 print"  1 or more says begin at this question"
3840 gosub 3670:print"Where shall I start in"
3850 for k=1 to nf:print"File";k;:input dc(k):next
3860 q=0:for k=1 to nf:if dc(k)<>0 then q=q+1
3870 next:if q>=1 then 3900
3880 print:print"Redo this, you've turned all files off!"
3890 print:goto 3810
3900 if q<=3 then 4060
3910 print:print"Redo this, 3 files maximum!"
3920 print:goto 3810
3930 get r$:if r$<>n$ then 3930
3940 print:print
3950 print"Select the game difficulty you prefer":print
```

```
3960 print"1 - Easy"
3970 print"2 - Medium"
3980 print"3 - Hard"
3990 print:print"Which would you like (1, 2, or 3)?"
4000 get r$:if r$=n$ then 4000
4010 if r$<"1" or r$>"3" then 4000
4020 q=val(r$):on q goto 4030,4040,4050
4030 nc=3:dt=0.7:dx=0.19:goto 4060
4040 nc=4:dt=0.8:dx=0.22:goto 4060
4050 nc=5:dt=0.9:dx=0.25:goto 4060
4060 goto 3440
4070 get r$:if len(r$)>0 then 4070
4080 print chr$(142);chr$(8);hc$
4090 poke 53280,6:poke 53281,4:print ww$:print:print
4100 print tab(12);"q u i k w i t":print:print
4110 print tab(4);"copyright 1984 dilithium press"
4120 print:print:print:print
4130 print tab(6);"from the book 'mind moves'"
4140 print:print
4150 print tab(5);"by tom rugg and phil feldman"
4160 for j=1 to 6:print:next
4170 print tab(9);"press a key to begin"
4180 get r$:if len(r$)=0 then 4180
4190 q=rnd(-ti)
4200 return
4210 poke v+21,0
4220 print chr$(142);chr$(9);chr$(154);hc$;:clr
4230 poke 53280,14:poke 53281,6
4240 for j=1 to 9:close j:next:close15
9990 end

ready.
```

EASY CHANGES

1. The keys pressed to indicate each player's guesses can be changed if you desire. The five keys used for player A (the red or left player) are set by the variables a1$ to a5$ in line 130. Similarly, the five keys for player B (the blue or right player) are set by the variables b1$ to b5$ in lines 140 and 150. You might want to change the keys for player B if you find using the function and cursor control keys awkward. One choice would be to use the k, l, :, ;, and = keys. Look at the keyboard to see why this would be logical. (You can tape labels 1 to 5 on

these keys if you want.) To implement this, change lines 140 and 150 as follows:

```
140 b1$="k":b2$="l":b3$=":"
150 b4$=";":b5$=" ="
```

2. You can remove all sound effects from the program (some people find sound distracting) by changing line 170 to:

```
170 sf=0
```

3. The speed at which the time ball falls is controlled by the variable dt in line 180. Make it larger to get a faster fall or smaller to get a slower one. A fairly fast fall (approximately eight seconds) can be achieved with:

```
180 dt=1.2
```

4. In the one-player game, the speed at which the game ball moves toward the computer's side is controlled by the variable dx in line 190. Make it larger to get a tougher game or smaller to get an easier one. You can get a much easier game by slowing down the movement with:

```
190 dx=0.1
```

5. The maximum number of possible answer alternatives displayed with each question can be set anywhere from two to five. Its value is controlled by the variable nc in line 360. Adjusting nc greatly affects the difficulty of the questions and the time needed to answer them correctly. If you don't intervene, the program sets nc to four. Make this change to increase the number of answer possiblities to the maximum of five:

```
360 nc=5
```

6. The penalty movement of the game ball for a wrong answer is controlled by the variable py in line 200. Increase its value to make risking unsure guesses more hazardous. Turn it off completely to encourage faster guessing with:

```
200 py=0
```

7. You can adjust the starting location of the game ball after each goal is scored by altering the value of xc in line 210. Use this as a handicapping tool for the two-player game or to adjust the degree of game difficulty for the one-player game. A smaller value of xc will make the one-player game easier and will give an advantage to the red (or left) player in the two-player game. A larger value of xc will, of course, have the opposite effects. Try raising or lowering its current value of 136 by any amount up to 70 points. If the blue player is a little better than the red player in the two-player game, try:

```
210 xc=100
```

8. The distance the game ball moves when a correct answer is given is controlled by the variable fr in line 220. Make it larger to get rewarded more and smaller to get rewarded less. Be careful how much you change it, since the game ball movement is sensitive to small changes in fr. The game action changes greatly as fr is changed. Larger values cause goals to be scored much faster and smaller values correspondingly less often. Experiment to find your preference. Some people prefer the quick-paced game achieved with:

```
220 fr=0.9
```

9. The number of goals required to win is controlled by the variable sg in line 410. To make it twelve goals, use:

```
410 sg=12
```

10. The time delay between questions is controlled by the loop in line 540. If you want a shorter delay, reduce the 1000 in this line to a smaller value. Similarly, you can make the 1000 larger to lengthen the delay.

11. In the two-player game, when one player guesses incorrectly, his guess is displayed on the board. You might prefer not seeing this so the other player cannot benefit from the information. To accomplish this, change these four lines as follows:

```
1170 print rr$;"Wrong!";bb$;left$(sp$,3);
1300 poke sd,17:print bb$;"Wrong!"
1400 print bb$;left$(sp$,3);"Wrong!"
1540 poke sd,129:print rr$;"Wrong!"
```

PROGRAMMER'S NOTEBOOK

Essential to the operation of QUIKWIT is the use of data files to store the questions used by the program. We use sequential files (instead of random or relative files) because they are simple to implement, put less demand on machine memory and disk storage, and are easily explained when telling you how to create your own data files for use with QUIKWIT.

Each data file is a series of string text lines. The first line contains the category of questions on that file — Arts and Entertainment, for example. The questions follow, each one consisting of seven lines. The first two lines hold the question itself, and the remaining five hold up to five possible answers.

The subroutine at line 3230 is called to initialize the disk files for use by the program. BASIC allows up to four files to be opened at once. One of these files should be the error (or communication) channel that can be interrogated to

make sure disk operations are going smoothly and to tell what's wrong when they're not. This leaves a maximum of three data files that can be open simultaneously. We adopt the Commodore standard of using file 15 for the error channel, and it is opened conventionally in line 3280. Line 3310 opens the sequential data files for the current game. The array dc (for disposition code) contains the information about which files are to be opened and where the reading of information is to begin in these opened files. If dc = 0, it means that file is not to be opened. A dc value of −1 means the file is to be opened and the first question is to be read from it randomly. A positive value of dc means the file is to be opened and the first question is to be the one with the same number as the value of dc. Remember, sequential files must be read sequentially (no jumping around). So once opened and initialized to a certain point, the questions will subsequently be read in order.

In line 3310, each file is given a file number of k, assigned a distinguishing channel number (k + 2), and identified by its file name from the f$ array. The "s, r" at the end of this line means the file will be sequential (s) and opened for reading (r) as opposed to writing. In the next line, the error channel is interrogated to make sure all is in order and each file is found to be okay. If not, an error message is printed.

Line 3360 reads the category name into the c$ array. Now q comes from line 3300 and holds the number of the first question to be asked. That means q–1 questions must be read from the file to position it to the desired spot. This is accomplished with the coding from 3360-3430. Note that the qn array is set to the last question read. The thing we have to be careful about here is reading past the end of the file. In line 3390, the machine variable st (for status) is checked after each read. If it is zero, we have not reached the end of the file. If it is not zero, the program will read the necessary remaining questions from the beginning of the file. This requires closing and reopening the file and reading past the category line at the beginning. This chore is accomplished in line 3400.

Questions are read from the disk into the computer's memory 24 at a time. QUIKWIT then gets these questions from main memory. Another 24 are read from the disk as necessary by the subroutine at line 2940. More than one file can be open for this reading and, if so, we want to take questions from each open category alternatively. In lines 2990 and 3000, k is looped over the range of possible files, checking dc(k) each time to see if that file is being used. If so, a question is taken from it. This looping process continues until 24 questions have been read. As a question is read, the information is stored in the a$ and b$ arrays in lines 3010 and 3020. The a$ array holds the two lines of each question, and the b$ array the five lines of possible answers. The first of these five lines always holds the correct answer, but the choices are shuffled randomly before they are presented on the screen. The arrays qn and nn contain the current question number within the file; qn with respect to the file argument k, and nn with respect to q, the question position from 1 to 24. The array cn(q) holds the file number k for that question, so the category name can be recalled quickly when the question is given on the screen. In lines 3040-3060, the variable st is checked to see if the end of the file has been reached. If so, the file is wrapped around by being closed and reopened so the next question read from it will be the first on the file.

The input# statement is used to read information from the disk into memory. This BASIC command treats certain characters as special delimiters, causing undesired complications for our use. These special characters are the comma, colon, semicolon, and quotation mark. But we found ourselves needing them in the text of the questions and possible answers. What to do?

We adopted a special encoding scheme to change them to other characters when the disk was created, then to decode them back after the disk is read. We use chr$(100), chr$(101), chr$(102), and chr$(103) respectively for the four special characters. In addition, a blank line is represented with chr$(104) and, if a line contains any of the four special

characters, a chr$(105) is placed at the beginning of the line to signal this. These chr$ values were chosen because they are not needed for any normal uppercase and lowercase text.

Thus, the question decoding subroutine works as follows. A question is about to be displayed on the screen with the subroutine at line 2700. In lines 2770, 2790, and 2890 the temporary string t$ is set to one line of data as it is read from the disk. Then the subroutine at line 3080 is called. This subroutine creates the corrected string tt$, which is t$ with any special characters converted back to their desired values.

As a question is being asked, we want to shuffle randomly the appropriate number of possible answers. This is done in lines 2800-2880. First, in lines 2800-2810, mc is set to the number of possible answers provided with the question. Most questions have five answers provided but some, like true/false, have as few as two. If nc (the number of choices you wish to present with each question) is less than mc, then mc is set to nc in 2820. The aa array, containing the integers from 1 to mc, is shuffled randomly in lines 2830-2870. This ensures the correct answer will be present, and a maximum of nc possibilities will be shown with each question.

It seems like all this was easier done than said!

MAIN ROUTINES

130-470	Initializes variables.
480-560	Mainline routine — calls other subroutines and controls game flow.
510-660	Subroutine to store sprite information.
670-700	Subroutine to store the bell sound information.
710-1740	Subroutine to ask the questions, get the replies, evaluate and display the results.
1750-1770	Subroutine to emit sound when goal is scored.
1780-2260	Subroutine to display the game board.
2270-2450	Processes end of game.
2460-2480	Subroutine to mark the correct answer on the screen.
2490-2510	Subroutine to make sound when a question is answered correctly.

2520	Subroutine to draw one blank orange line.
2530-2540	Subroutine to clear the two-line message area.
2550-2580	Subroutine to clear area where question appears.
2590-2600	Subroutine to clear the one-line message area and move cursor there for subsequent message.
2610-2690	Shows last question asked on each file (option 5).
2700-2930	Subroutine to display each question.
2940-3070	Subroutine to read next 24 questions into memory.
3080-3170	Subroutine to decode t$ string into tt$.
3180-3220	Subroutine to position cursor cd rows down and cr columns to the right from the upper left corner.
3230-3430	Subroutine to initialize the disk data files.
3440-3570	Subroutine to display the main option menu and branch to the option chosen.
3580	Initializes the one-player game (option 1).
3590-3660	Initializes the two-player game (option 2).
3670-3750	Subroutine to display the name and category of questions on each disk data file.
3760-3920	Provides for selection and positioning of the disk data files (option 3).
3930-4060	Provides for adjusting the difficulty level of the game (option 4).
4070-4200	Displays introductory title and waits until a key is pressed before continuing.
4210-9990	Ends program (option 6).

MAIN VARIABLES

sa, sb	Current score of player A and player B.
sg	Score required to win the game.
qq	Current question number in memory (from 1-24).
np	Number of players (1 or 2).
add, page, byte	Address, memory page, byte value of sprite data.
a, b	Value of guess for player A and player B.
y	Vertical location of time ball.
x	Horizontal position of game ball.
ca	Correct answer to question (from 1-5).
cd, cr	Values to position cursor down and right.
nc	Number of answer possiblities to display with each question (from 3-5).

mc	Number of answer possibilities available with the current question (from 2-nc).
sd	Memory address of SID sound chip (54276).
v	Memory address of VIC graphics chip (53248).
sf	Sound flag (1 = use sound, 0 = don't).
dt	Time ball descent rate.
dx	Game ball movement rate in one-player game.
py	Penalty movement of game ball for wrong answer.
xc	Starting location of game ball.
fr	Factor ratio to determine game ball movement reward for a correct answer.
fz	Range from which to choose random question in each file.
nf	Number of data files available on disk.
j, k, m, p, q	Loop and work variables.
x$	String of one blank space.
n$	String of one null character.
cr$, cd$, sp$	Strings of 40 cursor rights, 40 cursor downs, and 40 blank spaces.
ar$	String of 3 up arrows used to show winner.
r$	User input string.
p$	Formatting string.
na$, nb$	String names of player A and player B.
t$, tt$	Data string read from disk before conversion and after conversion.
w$	String read from error channel on the disk.
a1$-a5$	Strings of the five answer keys for player A.
b1$-b5$	Strings of the five answer keys for player B.
hc$, hm$	Strings to clear screen and home cursor.
bk$, ww$, rr$, bb$, pr$	Strings to cause screen printing color to be black, white, red, dark blue, and purple.
f$	String array of the names of the nf disk files.
c$	String array of the question categories for the nf disk data files.
a$(2, 24)	String array of the sets of two-line questions for the recent 24 questions read from disk.
b$(5, 24)	String array of the sets of five answer choices for the recent 24 questions read from disk.
nn(24), cn(24)	Question number and file number (from 1 – nf) in the original disk file.
qn(nf)	Question number in the disk file.

aq(nf)	Last question asked in each of the nf disk files.
dc(nf)	Disposition code of the nf disk files (− 1 = random start, 0 = turn off file, + = start at this number).
aa(5)	Random order to shuffle answer possiblities.
pv	Array of poke values to produce the bell sound.

SUGGESTED PROJECTS

1. Allow for up to four players (or teams) trying to answer the questions at the same time.
2. After a session with QUIKWIT, save the position in each data file on the disk itself. Then when you start the program the next time you can automatically continue where you left off.

5.

QUIKWORD

QUIKWORD is one of those games situated squarely in the big gray area that lies between entertainment and education. Is it a game, or does it improve your mind? We claim there's no reason why the answer can't be yes to both. Fun and learning should go together as often as possible, and word games are among the best ways for that to happen.

One to four players can play this new game, competing to create words from the letters on the screen. You score points based on the letters you use. Common letters (vowels and common consonants) score one point each. Rarer consonants score three or five points, depending on how rare they are. As in some forms of anagrams, your starting point is the word your opponent just created. However, you also have five letters of your own (your *hand* of letters) plus two *free* letters to choose from. Unlike in many word games, you very seldom find that you can't create any word at all. It's just a matter of trying to pick the best one before your time runs out, to score as many points as possible, while leaving the next player with limited scoring potential. If the game sounds a bit complicated when you read through the rules below, don't worry about it. Keep in mind that the nine-year-old daughter of one of the authors learned to play it within minutes. She's pretty good at it, too!

RULES

The game is played using a letter pool with 104 letters, made up of the following: 10 each of A, E, I, and O; five each of D, H, L, N, R, S, T, and U; two each of B, C, F, G, K, M, P, V, W, and Y; and one each of J, Q, X, and Z. We know there are some skeptics among you, so we'll save you some work:

$$(10 \times 4) + (5 \times 8) + (2 \times 10) + (1 \times 4) = 104$$

and

$$4 + 8 + 10 + 4 = 26$$

Letters that are most common (with 10 or five of each) score one point when used in a word. Letters that are least common (one of each) score five points, and the letters in the middle (two of each) score three points.

The letters in the letter pool are shuffled, and five are given (face up) to each player. In addition, two letters are drawn as free letters and are available for your use on your turn. To start the game, three random letters are drawn as the *main* word at the top of the screen, even though they don't usually form a word.

The first player tries to create a legal new main word using the letters taken from the old main word, the five letters in his hand, and the two free letters. Don't panic! You don't have to use *all* these letters — just some of them, as explained later. Before you start, you should decide what dictionary you will use to verify whether or not words are legal. We recommend the standard types of words allowed in most word games — no proper names, no slang, and so on, but that's up to you. When you create a word, you must use at least one letter from the main word, and at least one from your hand. Words must be at least three letters long, and no more than 10. You must use a given letter from your hand before you draw one of that same letter from the main word. This becomes important when the only letter your

new word would have drawn from the main word is duplicated in your hand.

If you can't create a word before your time limit of two minutes runs out, you can *pass*. You score no points, but you get to throw all your letters back into the pool so you can get five new ones. At the end of two minutes, you are forced to pass if you haven't yet created a legal word.

After you create a word, you add the point values of the letters to your score. Then your hand and the free letters are replenished with letters from the letter pool as necessary, and unused letters from the old main word are put aside, to be shuffled back into the letter pool when it becomes nearly empty.

The next player can decide to *challenge* your word if he does not believe it to be legal. If he does so, you have to consult your agreed-upon dictionary to try to find your word. If you find it to be a good (real) word, the challenger loses his turn and exchanges his letters, just as if he had decided to pass. If the word is bad, the challenger steals the points you would have scored with it, and continues with his turn.

You are not allowed to use a word that has previously been used in the game. Also, you cannot create a word that is simply the same main word as the last one with a suffix added (i.e., adding -s or -ed or -ing to the previous word).

The winner is the player with the most points at the end of a complete round in which someone has reached 100 points or more (or some other limit the players agree upon before the game).

HOW TO USE IT

The program starts, like the others, by displaying its title and copyright notice, and waiting for you to press a key. After a few seconds during which the program gets things ready for you, it displays an option menu with seven choices. Press a key from 1 to 7 to select the one you like.

The first four simply indicate the number of players in the game, 1 to 4. After you select one of these, you are asked for the names of the players (up to eight characters each), then the game begins. Option 5 causes some very brief instructions to be displayed. Option 7 (have patience, we'll get to option 6) ends the program after resetting the screen colors to their original hues.

With option 6, you can change the primary two settings of the game — the time limit per turn (choice 1) or the number of points it takes to win the game (choice 2). Simply press the 1 or 2 key, and the program will show you the current limit (120 seconds for choice 1 or 100 points for choice 2, unless previously changed). Then you can enter a new value. The time limit must be at least 10 seconds but no more than 600 seconds (10 minutes). The winning point score can be from 20 to 1000. To change both limits, choose option 6 twice.

After you finish with option 5 or 6, you are returned back to the menu so you can pick one of the first four options to start the game.

The game begins as explained in the Rules section. Drawn from the pool are letters for all of the players, plus two free letters, plus three for the first main word. The letters are displayed in different colors so you can quickly distinguish among the one-, three-, and five-point letters. The timer immediately starts counting down toward zero, so if you are the first player, you had better start trying to come up with a word. Simply type in the letters of the word you choose and press the [RETURN] key. The word will be shown in the *work* area as you type it and, after you press [RETURN], one of two things will happen.

If you type an acceptable word, you see the message WORD OK in the lower right corner, the score of the word appears in the work area. You are asked to either press the A key to accept the word (taking the points and ending your turn), or to press the R key to retry for a new word. For retry, the work area is cleared and you start over again (except the timer continues relentlessly counting down). If

the timer runs out before you press the A key, the word in the work area will be accepted automatically, even if it has been only partly entered and [RETURN] has not been pressed. If the partial word is not a real word, the next player can challenge it.

Keep in mind that this program uses no dictionary, so it only judges an acceptable word on the basis of whether the rules were followed in taking letters from the right places. It's up to the next player to challenge a player who uses a nonexistent word.

If you type an unacceptable word, the message *NO GOOD* is displayed in the lower right corner, with a brief explanation right below it to explain why. There are five reasons:

1. TOO SHORT	The word was not three or more letters long or time ran out with no word entered.
2. EXTRA:x	There was an extra letter x in the word you entered. Only the first extra letter is shown if more were found.
3. NO MAIN	No letter from the main word was used, after all letters possible were taken from the player's hand.
4. NO HAND	No letter from the player's hand was used.
5. DUPLICATE	The word duplicates a word used earlier.

At this point, the program waits for you to press either the R key to retry (for a new word), or the [f1] key to give up and pass your turn.

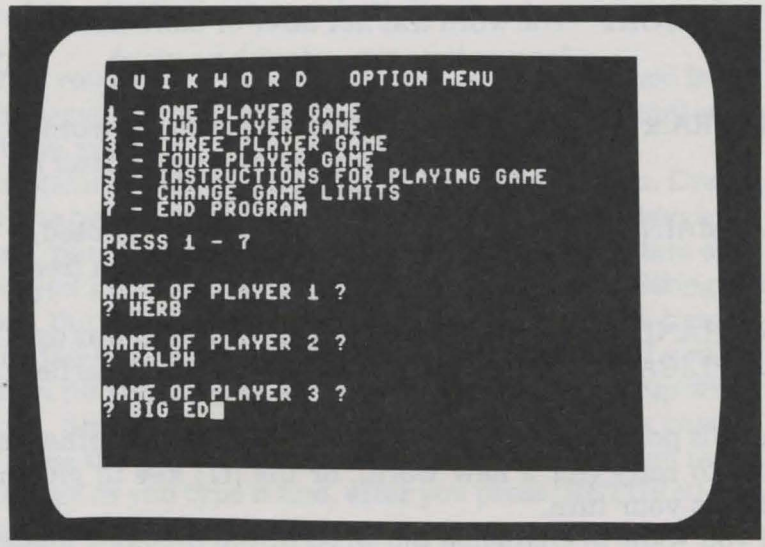
If you want to challenge the word of the previous player, press the [f7] key before your time limit expires. You are then asked to press either G (good) or B (bad) to indicate what you find out about the word when you check the dictionary. The program adjusts the scores accordingly. If the word is bad, the challenger's timer is reset to the full time limit, then his turn continues as usual. You can't challenge if the previous player scored no points, or if you

are the only player in the game. The [f7] key is ignored in these situations. You can only challenge the player immediately before you.

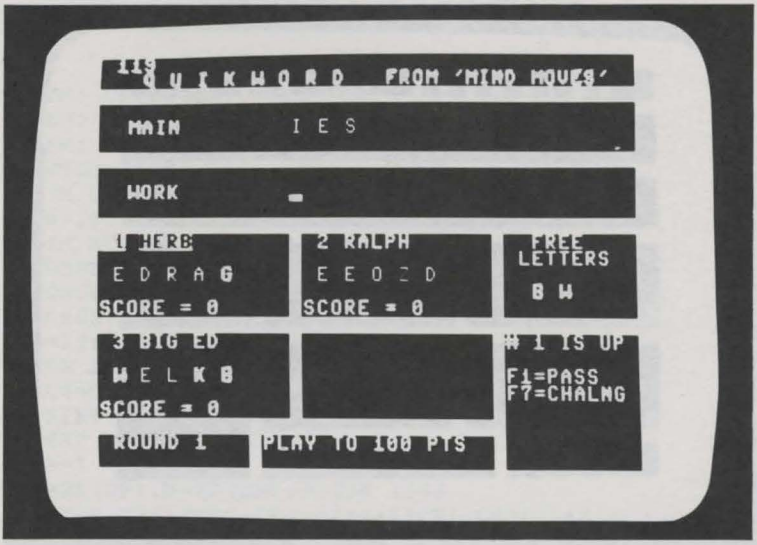
The program does not check if the new main word is simply the same main word as before with a suffix added. The next player must challenge the word to enforce the rule prohibiting such a word.

At the end of the game, the winner's name is shown in the message area in the lower right corner. Then you can press a key to restart the program.

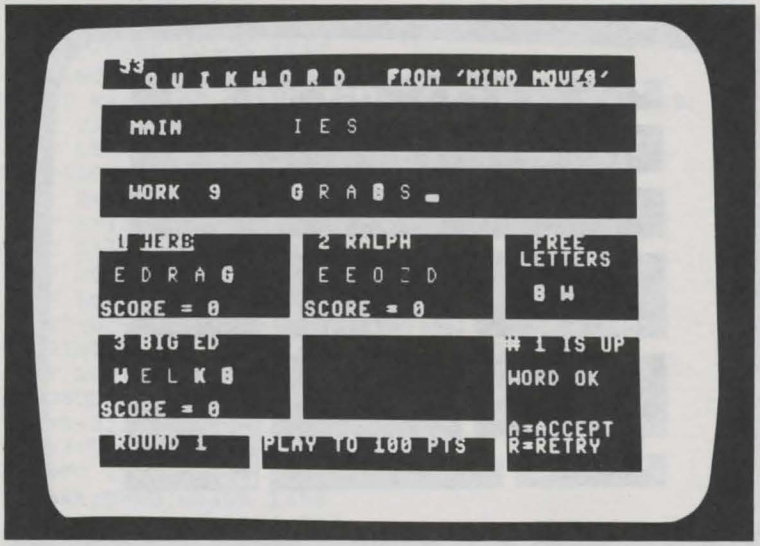
SAMPLE RUN



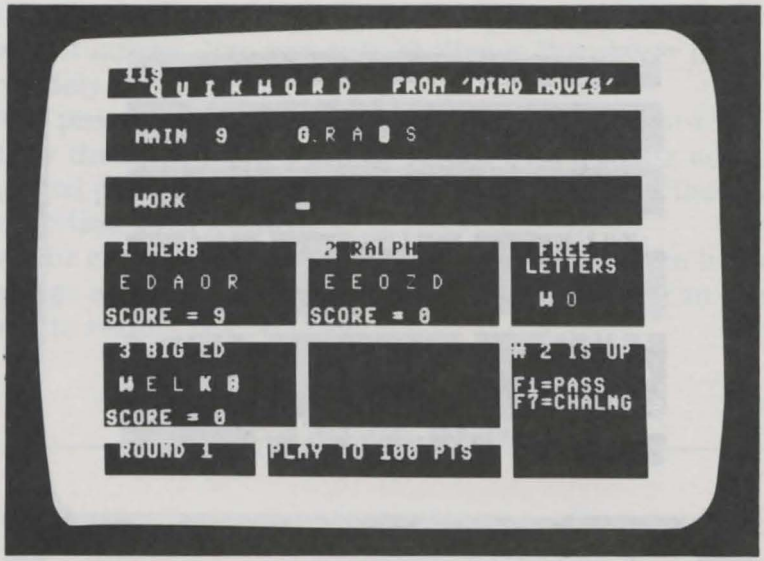
Herb, Ralph, and Big Ed decide to take on each other in a game of QUIKWORD. They choose option 3 and enter their names.



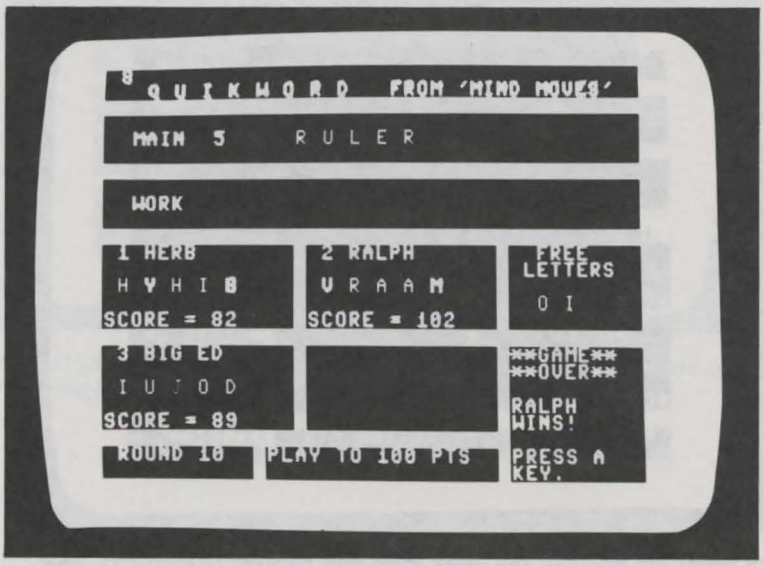
Herb has to create a word choosing letters from the five letters in his hand, the main word, and the two free letters. If the timer in the upper left corner reaches zero first, he scores no points. He must use at least one letter from both his hand and the main word.



Herb types his word in the work area and presses [RETURN]. The program responds by showing the points the word scores, and asking him to press A or R.



Herb presses A to accept the word. It becomes the main word, new letters are drawn for Herb, and it's Ralph's turn.



After many rounds of play, Ralph passes the 100-point mark to win the game.

PROGRAM LISTING

```
100 REM: QUIKWORD
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY TOM RUGG AND PHIL FELDMAN
130 GOSUB 3670
140 DIM LE$(4),NA$(4),SC(4),PA(4),PD(4),LC(5)
150 NW=200
160 DIM WD$(NW),AV(104),LS(26)
170 GOSUB 3290:GOSUB 2550:GOSUB 2280
180 GOSUB 1550:FOR CP=1 TO NP:GOSUB 1580:NEXT
190 GOSUB 1750:FOR CP=1 TO NP:GOSUB 1800:NEXT
200 A=13:D=4:GOSUB 1490
210 FOR J=1 TO 3:GOSUB 1610:MA$=MA$+R$
220 LC$=CHR$(LC(LS(ASC(R$)-64)))
230 PRINT SPC(1);LC$;R$;
240 NEXT
250 CP=1
260 A=PA(CP):D=PD(CP):GOSUB 1490
270 PRINT CT$;CHR$(18);CP;NA$(CP);CHR$(146);
280 TE=TI+TL+54
290 GOSUB 1720:A=MA:D=MD:GOSUB 1490
300 PRINT CT$;"#";CP;"IS UP";
310 D=D+2:GOSUB 1490:PRINT"F1=PASS";
320 D=D+1:GOSUB 1490:PRINT"F7=CHALNG";
330 A=WA-7:D=WD:GOSUB 1490
340 PRINT LEFT$(BL$,28)
350 SW=0:GOSUB 2090
360 WK$=""
370 A=WA:D=WD:GOSUB 1490
380 PRINT CC$;:A=A+1
390 GET R$:PRINT CHR$(19);INT((TE-TI)/60);CL$;B$;
400 IF TI>TE THEN 600
410 IF LEN(R$)=0 THEN 390
420 K=ASC(R$):IF K=13 THEN 600
430 IF K=20 THEN 500
440 IF K=133 THEN SW=0:GOTO 1100
450 IF K=136 THEN 2100
460 IF R$<"A" OR R$>"Z" THEN 390
470 IF LEN(WK$)>9 THEN 550
480 WK$=WK$+R$:GOSUB 1490
490 PRINT CL$;R$;CR$;CC$;:A=A+2:GOTO 390
500 IF LEN(WK$)=0 THEN 390
510 GOSUB 1490:PRINT CL$;B$;CL$;CL$;CL$;CC$;
520 A=A-2:IF LEN(WK$)=1 THEN 360
530 WK$=LEFT$(WK$,LEN(WK$)-1)
540 GOTO 390
550 A=WA:D=WD:GOSUB 1490
560 PRINT"** TOO LONG **"
570 FOR J=1 TO 1200:NEXT
580 A=WA:D=WD:GOSUB 1490:PRINT LEFT$(BL$,22)
590 GOTO 360
600 GOSUB 2090:GOSUB 1490:PRINT B$;
```

```
610 GOSUB 1170
620 A=MA:D=MD+2:GOSUB 1490:IF W=0 THEN 680
630 PRINT"*NO GOOD*";
640 D=D+1:GOSUB 1490:PRINT LEFT$(BL$,9);
650 GOSUB 1490:PRINT DM$;
660 D=D+2:GOSUB 1490:PRINT"F1=PASS";
670 GOTO 710
680 PRINT"WORD OK";
690 D=D+1:GOSUB 1490:PRINT LEFT$(BL$,9);
700 D=D+2:GOSUB 1490:PRINT"A=ACCEPT";
710 D=D+1:GOSUB 1490:PRINT"R=RETRY";
720 A=WA-7:D=WD:GOSUB 1490:PRINT SW;
730 A=WA:GOSUB 1490:PRINT T$;CT$;
740 GET R$:PRINT CHR$(19);INT((TE-TI)/60);CL$;B$;
750 IF TI<=TE THEN 780
760 IF SW=0 THEN 1100
770 GOTO 830
780 IF LEN(R$)=0 THEN 740
790 IF R$="R" THEN GOSUB 1720:GOTO 290
800 IF R$=CHR$(133) THEN SW=0:GOTO 1100
810 IF R$<>"A" THEN 740
820 IF SW=0 THEN 740
830 SC(CP)=SC(CP)+SW:PS=SW
840 PW=PW+1:IF PW>NW THEN W=1:GOTO 970
850 WD$(PW)=WK$
860 GOSUB 1900
870 GOSUB 1550:GOSUB 1750
880 GOSUB 1580:GOSUB 1800
890 J=FRE(0):CP=CP+1:IF CP<=NP THEN 260
900 J=0:W=0:T=-1:FOR CP=1 TO NP
910 IF SC(CP)=T THEN W=1
920 IF SC(CP)>T THEN T=SC(CP):J=CP
930 NEXT
940 IF T>=WS THEN 970
950 RN=RN+1:A=6:D=23:GOSUB 1490:PRINT RN;
960 GOTO 250
970 GOSUB 1720:A=MA:D=MD:GOSUB 1490
980 PRINT"**GAME**"
990 PRINT TAB(MA);"**OVER**":PRINT
1000 IF W=0 THEN 1030
1010 PRINT TAB(MA);"TIE GAME."
1020 GOTO 1050
1030 PRINT TAB(MA);NA$(J)
1040 PRINT TAB(MA);"WINS!"
1050 PRINT:PRINT TAB(MA);"PRESS A"
1060 PRINT TAB(MA);"KEY.";
1070 GOSUB 2090
1080 GET R$:IF LEN(R$)=0 THEN 1080
1090 CLR:GOTO 130
1100 W$=LE$(CP):LE$(CP)="":PS=0:GOSUB 1580
1110 FOR J=1 TO LEN(W$)
1120 IF AV(PR)=0 THEN 1150
1130 PR=PR+1:IF PR>NL THEN PR=1
1140 GOTO 1120
```

```
1150 AV(PR)=ASC(MID$(W$,J,1)):NEXT
1160 GOTO 870
1170 W=0:T$="":SW=0:FL=0:FM=0
1180 IF LEN(WK$)<3 THEN W=1:DM$="TOO SHORT":GOTO 1440
1190 FOR J=1 TO NE:UL(J)=ASC(MID$(LE$(CP),J,1)):NEXT
1200 FOR J=1 TO NF:UF(J)=ASC(MID$(FR$,J,1)):NEXT
1210 FOR J=1 TO LEN(MA$):UM(J)=ASC(MID$(MA$,J,1)):NEXT
1220 FOR J=1 TO LEN(WK$):IF W>0 THEN 1390
1230 R$=MID$(WK$,J,1):T=ASC(R$)
1240 FOR K=1 TO NE:IF UL(K)<>T THEN 1270
1250 IF T=0 THEN 1270
1260 UL(K)=0:T=0:FL=FL+1
1270 NEXT:IF T=0 THEN 1370
1280 FOR K=1 TO LEN(MA$):IF UM(K)<>T THEN 1310
1290 IF T=0 THEN 1310
1300 UM(K)=0:T=0:FM=FM+1
1310 NEXT:IF T=0 THEN 1370
1320 FOR K=1 TO NF:IF UF(K)<>T THEN 1350
1330 IF T=0 THEN 1350
1340 UF(K)=0:T=0
1350 NEXT
1360 IF T<>0 THEN W=2:DM$="EXTRA:"+CHR$(T):GOTO 1390
1370 LC$=CHR$(LC(LS(ASC(R$)-64))):SW=SW+LS(ASC(R$)-64)
1380 T$=T$+LC$+R$+B$
1390 NEXT:IF W>0 THEN SW=0:GOTO 1440
1400 IF FM<1 THEN W=3:DM$="NO MAIN"
1410 IF FL<1 THEN W=4:DM$="NO HAND"
1420 IF W=0 THEN GOSUB 1450
1430 IF W>0 THEN SW=0
1440 RETURN
1450 IF PW=0 THEN 1480
1460 FOR J=1 TO PW:IF WD$(J)=WK$ THEN W=5:DM$="DUPLICATE"
1470 NEXT
1480 RETURN
1490 PRINT CHR$(19);
1500 IF A=0 THEN 1520
1510 PRINT LEFT$(SR$,A);
1520 IF D=0 THEN 1540
1530 PRINT LEFT$(SD$,D);
1540 RETURN
1550 IF LEN(FR$)=NF THEN 1570
1560 GOSUB 1610:FR$=FR$+R$:GOTO 1550
1570 RETURN
1580 IF LEN(LE$(CP))=NE THEN 1600
1590 GOSUB 1610:LE$(CP)=LE$(CP)+R$:GOTO 1580
1600 RETURN
1610 R$="":IF AV(PP)=0 THEN 1630
1620 R$=CHR$(AV(PP)):AV(PP)=0
1630 PP=PP+1:IF PP+12>NL THEN GOSUB 1660
1640 IF LEN(R$)=0 THEN 1610
1650 RETURN
1660 A=30:D=21:GOSUB 1490:PRINT CT$;"SHUFFLING";
1670 FOR K=NL TO 2 STEP -1:W=INT(K*RND(1))+1
1680 T=AV(W):AV(W)=AV(K):AV(K)=T
```

```
1690 NEXT:PP=1:PR=1
1700 GOSUB 1490:PRINT LEFT$(BL$,9);
1710 RETURN
1720 FOR K=0 TO 6:A=MA:D=MD+K:GOSUB 1490
1730 PRINT LEFT$(BL$,9);:NEXT
1740 RETURN
1750 A=FA:D=FD:GOSUB 1490
1760 FOR J=1 TO NF:R$=MID$(FR$,J,1)
1770 LC$=CHR$(LC(LS(ASC(R$)-64)))
1780 PRINT SPC(1);LC$;R$;
1790 NEXT:RETURN
1800 A=PA(CP):D=PD(CP):GOSUB 1490
1810 PRINT CT$;CP;NA$(CP);
1820 D=D+2:GOSUB 1490
1830 FOR J=1 TO NE:R$=MID$(LE$(CP),J,1)
1840 LC$=CHR$(LC(LS(ASC(R$)-64)))
1850 PRINT SPC(1);LC$;R$;
1860 NEXT
1870 D=D+2:GOSUB 1490:PRINT LEFT$(BL$,12);:GOSUB 1490
1880 PRINT CT$;"SCORE =";SC(CP);
1890 RETURN
1900 FOR J=1 TO LEN(MA$)
1910 IF UM(J)=0 THEN 1960
1920 IF AV(PR)=0 THEN 1950
1930 PR=PR+1:IF PR>NL THEN PR=1
1940 GOTO 1920
1950 AV(PR)=ASC(MID$(MA$,J,1))
1960 NEXT
1970 MA$=WK$:LE$(CP)="":FR$=""
1980 FOR J=1 TO NE:IF UL(J)=0 THEN 2000
1990 LE$(CP)=LE$(CP)+CHR$(UL(J))
2000 NEXT
2010 FOR J=1 TO NF:IF UF(J)=0 THEN 2030
2020 FR$=FR$+CHR$(UF(J))
2030 NEXT
2040 A=7:D=4:GOSUB 1490:PRINT LEFT$(BL$,28)
2050 GOSUB 1490:PRINT SW;
2060 A=14:GOSUB 1490:PRINT T$;
2070 A=7:D=WD:GOSUB 1490:PRINT LEFT$(BL$,28)
2080 RETURN
2090 FOR J=1 TO 10:GET R$:NEXT:RETURN
2100 IF PS=0 OR NP=1 THEN 390
2110 GOSUB 1720
2120 A=MA:D=MD:GOSUB 1490:PRINT"CHALLENGE";
2130 D=D+1:GOSUB 1490:PRINT"---------";
2140 D=D+1:GOSUB 1490:PRINT"VERIFY";
2150 D=D+1:GOSUB 1490:PRINT"LAST WORD";
2160 D=D+2:GOSUB 1490:PRINT"G=GOOD";
2170 D=D+1:GOSUB 1490:PRINT"B=BAD";
2180 GOSUB 2090
2190 GET R$:IF LEN(R$)=0 THEN 2190
2200 IF R$="G" THEN 2270
2210 IF R$<>"B" THEN 2190
2220 J=CP-1:IF CP=1 THEN J=NP
```

```
2230 SC(J)=SC(J)-PS:IF SC(J)<0 THEN SC(J)=0
2240 SC(CP)=SC(CP)+PS:PS=0
2250 T=CP:CP=J:GOSUB 1800:CP=T:GOSUB 1800
2260 GOTO 260
2270 GOTO 1100
2280 PRINT CHR$(147)
2290 PRINT TAB(3);CHR$(5);TT$;SPC(3);
2300 PRINT"FROM 'MIND MOVES'"
2310 PRINT HL$
2320 PRINT TAB(2);CT$;"MAIN"
2330 PRINT:PRINT HL$
2340 PRINT TAB(2);CT$;"WORK"
2350 PRINT:PRINT HL$
2360 PRINT:PRINT:PRINT:PRINT
2370 PRINT HL$
2380 PRINT:PRINT:PRINT:PRINT
2390 PRINT LEFT$(HL$,31)
2400 A=0:D=11:GOSUB 1490
2410 FOR J=14 TO 29 STEP 15
2420 FOR K=11 TO 22
2430 A=J:D=K:GOSUB 1490:PRINT VP$;
2440 NEXT:NEXT
2450 PRINT
2460 PRINT TAB(11);VP$;TAB(29);VP$
2470 PRINT TAB(11);VP$;TAB(29);VP$;
2480 A=32:D=11:GOSUB 1490
2490 PRINT CT$;"FREE"
2500 PRINT TAB(31);"LETTERS"
2510 A=0:D=23:GOSUB 1490
2520 PRINT TAB(1);"ROUND";RN;TAB(11);
2530 PRINT VP$;CT$;"PLAY TO";WS;"PTS"
2540 RETURN
2550 PRINT CHR$(147)
2560 PRINT TT$;SPC(3);"OPTION MENU"
2570 PRINT
2580 PRINT"1 - ONE PLAYER GAME"
2590 PRINT"2 - TWO PLAYER GAME"
2600 PRINT"3 - THREE PLAYER GAME"
2610 PRINT"4 - FOUR PLAYER GAME"
2620 PRINT"5 - INSTRUCTIONS FOR PLAYING GAME"
2630 PRINT"6 - CHANGE GAME LIMITS"
2640 PRINT"7 - END PROGRAM"
2650 PRINT
2660 PRINT"PRESS 1 - 7"
2670 GET R$:IF LEN(R$)=0 THEN 2670
2680 IF R$<"1" OR R$>"7" THEN 2670
2690 PRINT R$:PRINT
2700 NP=VAL(R$)
2710 ON NP GOTO 2730,2730,2730,2730,2800,3020,3880
2720 STOP
2730 FOR J=1 TO NP:R$=""
2740 PRINT"NAME OF PLAYER";J;"?"
2750 INPUT R$
2760 IF LEN(R$)=0 THEN 2750
```

```
2770 NA$(J)=LEFT$(R$,8):PRINT
2780 NEXT
2790 RETURN
2800 PRINT
2810 PRINT TT$;SPC(3);"MINI-INSTRUCTIONS"
2820 PRINT
2830 PRINT"THE OBJECT OF THE GAME IS TO REACH"
2840 PRINT WS;"POINTS BY FORMING WORDS."
2850 PRINT"EACH PLAYER FORMS A WORD FROM THE"
2860 PRINT NE;"LETTERS HE HOLDS, PLUS THE LETTERS"
2870 PRINT"IN THE 'MAIN' WORD, PLUS THE";NF
2880 PRINT"'FREE' LETTERS."
2890 PRINT"WORDS MUST BE 3 OR MORE LETTERS LONG"
2900 PRINT"AND MUST USE AT LEAST ONE LETTER FROM"
2910 PRINT"THE MAIN WORD AND AT LEAST ONE FROM"
2920 PRINT"THE PLAYER'S 'HAND'."
2930 PRINT"LETTERS COME FROM THE PLAYER'S HAND"
2940 PRINT"BEFORE THE MAIN WORD.  LETTERS COUNT"
2950 PRINT"1, 3, OR 5 POINTS EACH AS SHOWN BY"
2960 PRINT"THEIR COLORS."
2970 PRINT
2980 PRINT"FULL DETAILS ARE IN THE BOOK"
2990 PRINT"'MIND MOVES' FROM DILITHIUM PRESS."
3000 GOSUB 3840
3010 GOTO 2550
3020 PRINT
3030 PRINT"CURRENT GAME LIMITS ARE:"
3040 PRINT
3050 PRINT"1 --";TL/60;"SECONDS TIME LIMIT"
3060 PRINT TAB(6);"PER TURN."
3070 PRINT
3080 PRINT"2 --";WS;"POINTS WIN THE GAME."
3090 PRINT
3100 PRINT"PRESS 1 OR 2 TO PICK THE ONE TO CHANGE."
3110 PRINT
3120 GET R$:IF LEN(R$)=0 THEN 3120
3130 IF R$="1" THEN 3160
3140 IF R$="2" THEN 3230
3150 GOTO 3120
3160 PRINT"ENTER NEW TIME LIMIT (10 - 600)"
3170 R$="":INPUT R$
3180 IF LEN(R$)=0 THEN 3160
3190 W=INT(VAL(R$))
3200 IF W<10 OR W>600 THEN 3160
3210 TL=W*60
3220 GOTO 3280
3230 PRINT"ENTER POINTS NEEDED TO WIN (20 - 1000)"
3240 R$="":INPUT R$
3250 IF LEN(R$)=0 THEN 3230
3260 W=VAL(R$):IF W<20 OR W>1000 THEN 3230
3270 WS=W
3280 GOTO 2550
3290 T$="":WK$="":MA$="":FR$="":LC$=""
3300 CR$=CHR$(29):CD$=CHR$(17):CL$=CHR$(157)
```

```
3310 PRINT CHR$(147)
3320 PRINT"ONE MOMENT PLEASE..."
3330 HL$=CHR$(18)+CHR$(154)
3340 FOR J=1 TO 40:HL$=HL$+CHR$(32):NEXT
3350 CT$=CHR$(158)
3360 CC$=CHR$(185)
3370 HL$=HL$+CHR$(146)
3380 VP$=LEFT$(HL$,3)+CHR$(146)
3390 B$=CHR$(32):BL$=B$:FOR J=1 TO 5
3400 BL$=BL$+BL$:NEXT
3410 SR$=CR$:FOR J=1 TO 6:SR$=SR$+SR$:NEXT
3420 SR$=LEFT$(SR$,39)
3430 SD$=CD$:FOR J=1 TO 5:SD$=SD$+SD$:NEXT
3440 SD$=LEFT$(SD$,24)
3450 J=FRE(0):PP=1:NL=0
3460 FOR J=1 TO 26:READ W:NL=NL+W
3470 IF W>4 THEN LS(J)=1:GOTO 3500
3480 IF W=1 THEN LS(J)=5:GOTO 3500
3490 LS(J)=3
3500 FOR K=1 TO W:AV(PP)=64+J:PP=PP+1:NEXT
3510 NEXT
3520 DATA 10,2,2,5,10,2,2,5,10,1,2,5,2
3530 DATA 5,10,2,1,5,5,5,5,2,2,1,2,1
3540 RN=1
3550 WS=100
3560 TL=120*60
3570 NF=2:NE=5
3580 DIM UL(NE),UF(NF),UM(12)
3590 PA(1)=0:PD(1)=11
3600 PA(2)=15:PD(2)=11
3610 PA(3)=0:PD(3)=17
3620 PA(4)=15:PD(4)=17
3630 WA=14:WD=8:FA=31:FD=14:MA=30:MD=17
3640 LC(1)=154:LC(3)=5:LC(5)=156
3650 GOSUB 1660
3660 RETURN
3670 PRINT CHR$(147);CHR$(142);CHR$(8);CHR$(158)
3680 POKE 53269,0:POKE 53280,14:POKE 53281,0
3690 TT$="Q U I K W O R D"
3700 PRINT:PRINT
3710 PRINT TAB(12);TT$
3720 PRINT:PRINT
3730 PRINT TAB(4);"COPYRIGHT 1984 DILITHIUM PRESS"
3740 PRINT:PRINT:PRINT:PRINT
3750 PRINT TAB(6);"FROM THE BOOK 'MIND MOVES'"
3760 PRINT:PRINT
3770 PRINT TAB(5);"BY TOM RUGG AND PHIL FELDMAN"
3780 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
3790 PRINT TAB(9);"PRESS A KEY TO BEGIN"
3800 GET R$:IF LEN(R$)=0 THEN 3800
3810 K=RND(-TI)
3820 J=1:W=0:T=0:I=1
3830 RETURN
3840 PRINT
```

```
3850 PRINT TAB(6);"PRESS A KEY TO CONTINUE"
3860 GET R$:IF LEN(R$)=0 THEN 3860
3870 RETURN
3880 POKE 53280,14:POKE 53281,6
3890 PRINT CHR$(147);CHR$(9);CHR$(154)
9990 END

READY.
```

EASY CHANGES

1. To avoid having to use option 6 of the menu to change either the points needed to win or the time limit, you can change lines 3550 and 3560 instead. That way, your new limits will become the standard limits. Suppose you wanted limits of 75 points as the winning score and 90 seconds as the time. Make these changes:

   3550 WS=75
   3560 TL=90*60

2. Change the number of free letters and the number of letters in each player's hand by changing line 3570. You can have up to four free letters and seven letters in each hand, and you must have at least one of each. If you want three free letters and six letters in each hand, change it to:

   3570 NF=3:NE=6

3. Change the number of letters in the main word to begin the game by changing line 210. Change the 3 in the line to any number from 1 to 10. For example, to start with a five-letter word:

   210 FOR J=1 TO 5:GOSUB 1610:MA$=MA$+R$

4. To require the shortest legal words to be at least four letters long, not three, change the 3 in line 1180 to 4. In addition, change the 3 in line 2890 to 4, so the on-screen instructions will be accurate.

5. The colors chosen to represent the one-, three-, and five-point letters have been selected pretty carefully, but you can change them if you wish. Change the three numeric values in line 3640 to values representing the colors you choose, based on the information in Appendix C of the *Commodore 64 Programmer's Reference Guide*. Instead of light blue, white, and purple, you could use something like cyan, yellow, and orange with:

```
3640 LC(1)=159:LC(3)=158:LC(5)=129
```

6. You can change the quantity of each letter in the letter pool by changing lines 3520 and 3530. The first number in line 3520 is the number of As, the second number is the number of Bs, and so on. In this way, line 3520 contains the quantities for A through M, and 3530 has N through Z. If you change any of these, be aware that one-point letters are those with five or more of that letter, five-point letters are those with one letter, and three-point letters are those with two to four of that letter. Also, if you will have more than 104 total letters, you need to increase the 104 in line 160 to be at least as large as the number of letters you will have.

7. The program currently allows for up to 200 words to be stored for use in checking for duplicates when a new word is entered. If you plan to play a very long game (over 500 points or so), you may want to expand this limit before starting the game. Otherwise, when you try to enter the 201st word, the game will be declared a tie. To allow for 500 words, change line 150 to:

```
150 NW=500
```

PROGRAMMER'S NOTEBOOK

To reduce delays caused by the so-called garbage collection process that BASIC goes through when repeated string manipulation of certain types is done, this program uses several numeric arrays to manipulate the ASCII values of the words and letters involved. In particular, the AV array contains the letter pool, and the UL, UM, and UF arrays contain the player's hand, main word, and free letters when a new work word is being checked for legality. After manipulation is complete, the strings are rebuilt from these arrays.

After the letters are shuffled in the AV array, they are drawn sequentially from it as needed during the game. When a letter has been drawn, its ASCII value is replaced with zero to indicate an open space in the letter pool. When letters must be replaced into the pool, they are replaced sequentially into these open spaces. Randomness is maintained by shuffling all letters in the letter pool (including replaced ones) when the next letter to be drawn is less than 12 from the end of the array. After a shuffle, the drawing and replacing begins at the top of the array again.

MAIN ROUTINES

130-170	Displays title screen, initializes variables, displays menu, processes responses, and displays main playing screen.
180-190	Selects and displays free letters and letters of each player's hand.
200-240	Selects and displays starting main word.
250-320	Shows which player is up by displaying his name in reverse letters and a message in the message area.
330-350	Clears the area on the screen for the work word and gets ready for the player's keyboard entry.
360-490	Processes the keyboard entry of player CP's work word. Displays the time remaining.
500-540	Processes [DEL] key to backspace.

550-590	Discards the work word if it is over 10 characters long (after displaying a message).
600-820	Evaluates the work word after [RETURN] is pressed or the time limit expires. Accepts the player's response of A, R, or [f1] after showing if the word was good or not.
830-850	Updates player's score. Saves word for future duplicate checks.
860-880	Returns unused letters to the letter pool from the old main word. Draws any needed letters for free letters and player's hand. Displays them.
890	Forces string garbage collection. Goes back to the next player if the end of a round has not been reached.
900-960	At the end of a round, checks to see if anyone has enough points to win and whether there is a tie. If no win or tie, goes back for a new round.
970-1090	Displays the name of the winner, or declares the game a tie. Waits for a key to be pressed, then starts a new game.
1100-1160	Processes a pass for a player who pressed [f1].
1170-1440	Subroutine to check if the work word is legal. Sets one of the five error messages if not. Builds string T$ to show the colors of the letters used.
1450-1480	Subroutine to check if the work word is a duplicate of a word used earlier in the game.
1490-1540	Subroutine to move the cursor A positions across and D positions down from the upper left corner of the screen.
1550-1570	Subroutine to draw more free letters from the letter pool if necessary.
1580-1600	Subroutine to draw more letters for player CP's hand if necessary.
1610-1650	Subroutine to draw the next available letter from the letter pool and put it into R$. Shuffles the letter pool if near the end.
1660-1710	Subroutine to shuffle the letter pool, displaying a message while the shuffling is taking place.
1720-1740	Subroutine to blank out the message area in the lower right corner.
1750-1790	Subroutine to display the free letters on the screen.
1800-1890	Subroutine to display the name, letters, and score of player CP on the screen.

1900-2080	Subroutine to put the unused letters from the old main word back into the letter pool, and remove the letters from the player's hand and the free letters that were used in making the new main word.
2090	'Subroutine to remove any possible extra keystrokes from the keyboard buffer.
2100-2270	Processing for a challenge. Displays messages, accepts B or G key, and passes or adjusts score.
2280-2540	Subroutine to display the QUIKWORD playing screen.
2550-2790	Subroutine to display the option menu and process the response. Accepts the names of the players.
2800-3010	Displays brief instructions if option 5 is requested.
3020-3280	Changes game limits (option 6).
3290-3660	Subroutine to initialize the main variables at the start of the game and shuffle the letter pool.
3670-3830	Subroutine to display the opening title screen and initialize some variables and system conditions.
3840-3870	Subroutine to display a message and wait for a key to be pressed.
3880-9990	Resets system conditions and ends the program (option 7).

MAIN VARIABLES

SC	Score array containing the scores of the four players.
PA, PD	Arrays containing pointers to the across and down positions of the four players' display areas.
LC	Letter color array containing the ASCII values of the colors of the three different values of letters.
AV	Array of the ASCII values of the 104 letters in the letter pool. A zero value means the letter has been drawn from that position.
LS	Letter scores array containing the point values earned by each of the 26 letters.
CP	Current player number.
NP	Number of players in the game.
A, D	Number of across and down positions to move the cursor from the upper left corner of the screen.
J, K, T, W	Loop and temporary work variables.

TE	Time to end the current player's turn.
TL	Time limit for each player (seconds times 60).
TI	Current time of day on the computer's internal timer (seconds times 60).
MA, MD	Across and down coordinates of the upper left corner of the message area.
WA, WD	Across and down coordinates of the work word area.
FA, FD	Across and down coordinates of the free letters.
SW	Score of the work word.
PS	Previous player's score.
NW	Number of words that can be saved to check for duplication.
PW	Pointer into WD$ to the last word saved.
WS	Number of points needed for a winning score.
RN	Round number.
PP	Pointer into the letter pool array to the next letter to be drawn.
PR	Pointer into the letter pool array to where to put the next letter to be returned.
NL	Number of letters in the letter pool.
FL	Number of letters in WK$ found in the player's letter hand.
FM	Number of letters in WK$ found in the main word.
NE	Number of letters in each player's hand.
NF	Number of free letters.
UL	Array of ASCII values of the letters in the player's hand.
UM	Array of ASCII values of the letters in the main word.
UF	Array of ASCII values of the free letters.
I	Constant 1.
LE$	Array with the letters of the four players.
NA$	Array with the names of the four players.
WD$	Array with all the words used so far in the game.
MA$	Main word.
WK$	Work word.
FR$	Free letters.
R$	Reply of player (key pressed). Also work string.
LC$	Letter color.
CT$	Color of regular text on the screen.
BL$	String of 32 blanks.
B$	Single blank character.

CR$, CD$, CL$	Strings to move the cursor right, down, and left.
CC$	Cursor character.
DM$	Diagnostic message when a player's word is no good.
T$, W$	Temporary work strings.
SR$, SD$	Strings to move the cursor right, down.
TT$	Program title (with alternate blanks).
HL$	Horizontal line string of reverse blanks.
VP$	Vertical point string for use in drawing vertical lines on the playing area.

SUGGESTED PROJECTS

1. Change the priority in which letters are drawn from the player's hand and the old main word to make the work word. Rather than always taking letters first from the player's hand, take an identical letter from the main word if that would make the new word legal.
2. Change the game so it requires at least one letter to be taken from the free letters as well as from the player's hand and the main word.
3. Change the logic that returns letters to the letter pool so that returned letters can immediately be drawn again. In the current form of the game, letters are redrawn only after the letter pool is shuffled.
4. Change the game to increase the points scored if the player completes his word in less than half the time limit.

6.

VERTIGO

VERTIGO is a board game that is as simple to learn as tic-tac-toe, but as challenging to master as checkers or go. The game is a sort of vertical, antigravity tic-tac-toe, in which the object is to get four in a row, not three. Each of the two players in turn releases a helium balloon of his color from the bottom of the playing area. Each balloon rises to the highest available position in its column. The first player to get four of his balloons in a row (vertically, horizontally, or diagonally) wins. You can play against a friend or the computer. If you play against the computer, you can select one of its four skill levels. Needless to say (but we'll say it anyway), as your mind boggles at the complications of the balloon positions, you may be stricken by an attack of . . . VERTIGO!

RULES

The VERTIGO playing area is a nine-by-nine grid. The computer randomly determines which player moves first. In subsequent games, the first move alternates between players. Each player can select any of the nine columns for his move. The player's token (colored helium balloon) is released in the selected column and rises to the highest unoccupied position in that column. Once a column is full with nine balloons, no more moves are possible in that column.

The players alternately make their moves until: one of them gets four in a row — vertically, horizontally, or diagonally; or the board is completely full. If the board is filled but no four-in-a-row has been made, the game is a tie.

VERTIGO is set up as a match consisting of the best-two-out-of-three games, although this can be changed, as explained later.

HOW TO USE IT

The game starts by displaying its title and copyright notice, and waiting for you to press a key to get things going. Once a key is pressed, the program displays a menu of five options. Press a key from 1 to 5 depending on the option you want.

Select option 1 if you want to play against the computer, or option 2 if you and a friend want to play against each other. If you pick option 2, the computer will ask for the names of the two players, each of which can be up to nine characters long. In either case (you vs. computer or you vs. friend), the computer will randomly select which player goes first, then will start the game.

Before we get into the details of how the game is played, let's cover what the other three options do. Use option 3 to change the skill level of the computer prior to choosing option 1 to make it your opponent. If you're playing against a friend, changing the skill level has no effect on the game. (Of course, if you consider the computer your friend, this isn't quite true, but let's get on with it anyway.)

There are four skill levels, imaginatively called 1 to 4. Level 1 makes the computer a pretty simple-minded opponent that often overlooks winning moves by either player. At level 2, the computer does better at blocking you from winning, but still can miss its own winning moves. At level 3, the computer detects winning moves by either side and plays a much stronger game. The computer plays its most advanced game at level 4, of course. You'll probably want to start at one of the lower levels while you're learning the game, then

move to higher levels after you start beating the computer. The skill level is set at level 2 by default if you don't change it via option 3 on the menu.

Menu option 4 displays abbreviated instructions on how to play the game, in case you have forgotten which keys to press in order to select your moves. Option 5 simply ends the game and resets the computer's screen colors and keyboard back to their standard settings.

And now, on to how the game is played. The computer displays the VERTIGO playing area, showing marks at the top and sides to indicate the nine-by-nine grid. When it is your move, your balloon is shown at the bottom of the playing area below the center column. To select the column for your move, use the 1 or 2 key on the keyboard to move the balloon left or right, respectively. The balloon moves one column each time you press a key. When you have the balloon under the column you want, press the [RETURN] key to release it up to its position.

If you are playing against a friend (a human one), his balloon will be displayed next, and will be moved the same way. After each move, the computer checks to see if a four-in-a-row has been made. If so, it indicates who won and updates the game scoreboard at the bottom of the screen. Then it tells you to press a key for a new game or match, depending on how many games have been won. If you would like to end the program rather than play another game, you can press the Q key (for quit). If you'd like to restart the program from the beginning (to take on a new opponent, for example), press the R key (for restart).

When the computer is your opponent, there are only a couple of differences. First, the computer's skill level (1 to 4) is displayed in the upper right corner of the screen. Second, the computer doesn't use the keyboard to make its moves. Instead, it starts by displaying its balloon under the leftmost column, and slowly (or quickly, depending on the skill level and game situation) moves it to the right, column by column, as it evaluates each possible move it might make. Then it releases its balloon under the column it has selected.

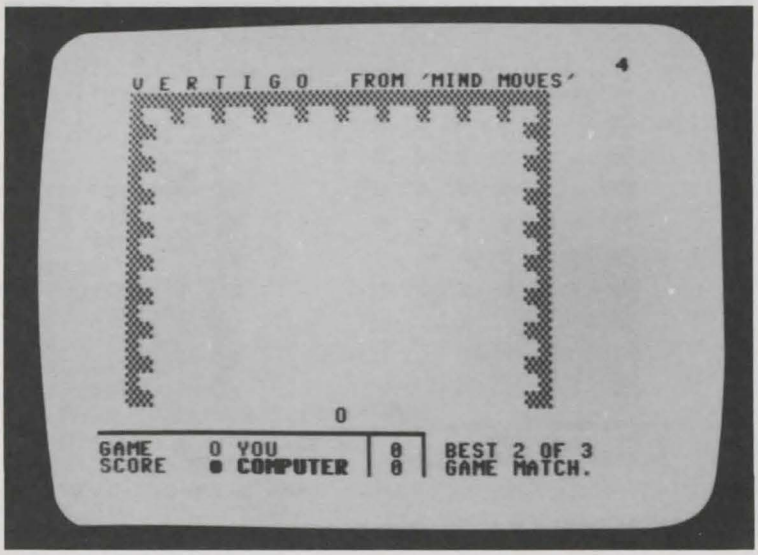
SAMPLE RUN

```
V E R T I G O   OPTION MENU

1 - 1 PLAYER GAME (YOU VS. COMPUTER)
2 - 2 PLAYER GAME
3 - CHANGE COMPUTER SKILL LEVEL
4 - INSTRUCTIONS FOR PLAYING GAME
5 - END PROGRAM

PRESS 1, 2, 3, 4, OR 5
3


THE CURRENT SKILL LEVEL IS 2

LEVEL 1 IS THE WEAKEST OPPONENT,
 AND LEVEL 4 IS THE STRONGEST.

ENTER 1, 2, 3, OR 4
```

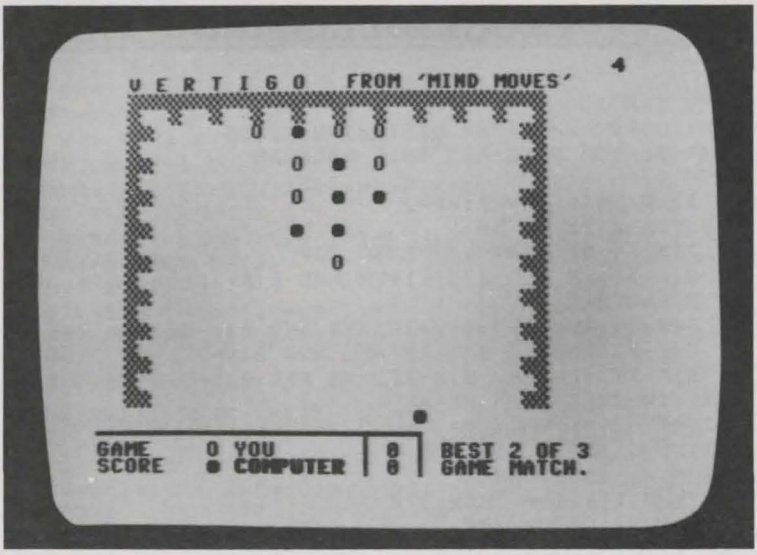
After pressing a key to start the game, the player selects option 3 to alter the computer's skill level.

```
THE CURRENT SKILL LEVEL IS 2

LEVEL 1 IS THE WEAKEST OPPONENT,
 AND LEVEL 4 IS THE STRONGEST.

ENTER 1, 2, 3, OR 4
4

V E R T I G O   OPTION MENU

1 - 1 PLAYER GAME (YOU VS. COMPUTER)
2 - 2 PLAYER GAME
3 - CHANGE COMPUTER SKILL LEVEL
4 - INSTRUCTIONS FOR PLAYING GAME
5 - END PROGRAM

PRESS 1, 2, 3, 4, OR 5
1

BY RANDOM CHOICE, THE FIRST MOVE GOES TO

. . . . YOU
```

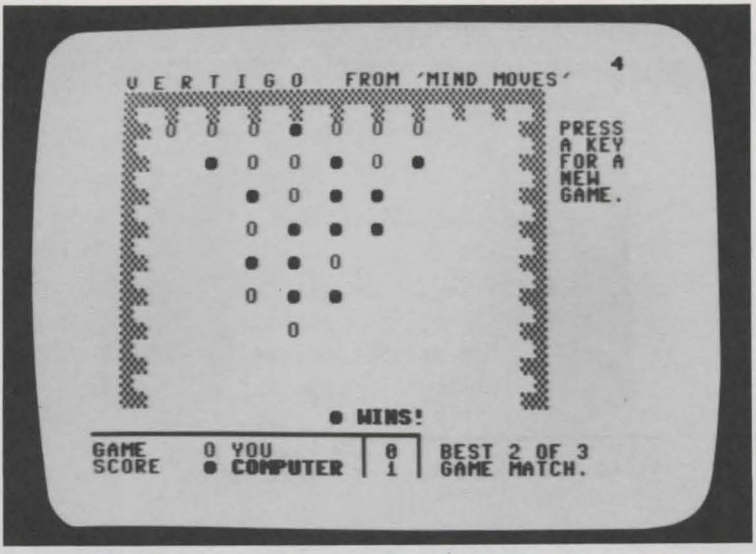
The player displays great confidence by selecting skill level 4, which invokes the computer's best strategy. Next he picks option 1 to start the game, and the computer tells him he gets the first move.



The player's helium balloon is beneath the playing area in the center. By using the 1 or 2 key, he can move the balloon left or right before releasing it to rise upward by pressing the [RETURN] key.



The player and computer have both made several moves, and the computer is in the midst of evaluating where to move on this turn.



After a hard-fought game, the computer finally prevails in the first game of the match by getting four in a row diagonally. The program now waits for the player to press a key to start the next game in the best-two-out-of-three match.

PROGRAM LISTING

```
100 REM: VERTIGO
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY TOM RUGG AND PHIL FELDMAN
130 GOTO 430
140 IF J>HP THEN SH=HP:HP=J
150 W=0:FOR K=11 TO HP
160 IF B(K)<1 OR B(K)<>FM THEN 220
170 IF B(K+1)=FM AND B(K+2)=FM AND B(K+3)=FM THEN 240
180 IF K<41 THEN 220
190 IF B(K-10)=FM AND B(K-20)=FM AND B(K-30)=FM THEN 240
200 IF B(K-9)=FM AND B(K-18)=FM AND B(K-27)=FM THEN 240
210 IF B(K-11)=FM AND B(K-22)=FM AND B(K-33)=FM THEN 240
220 NEXT:IF J>SH THEN HP=SH
230 RETURN
240 W=1:GOTO 220
250 FOR MV=1 TO 9:T=0
260 J=MS(MV):IF J=0 THEN 370
270 TA=1:TB=2:GOSUB 390
280 TA=-1:TB=-2:GOSUB 390
290 IF J<33 THEN 330
300 TA=-11:TB=-22:GOSUB 390
```

```
310 TA=-10:TB=-20:GOSUB 390
320 TA=-9:TB=-18:GOSUB 390
330 IF J>77 THEN 360
340 TA=9:TB=18:GOSUB 390
350 TA=11:TB=22:GOSUB 390
360 E(MV)=E(MV)+T
370 NEXT
380 GOTO 750
390 IF B(J+TA)<=0 THEN 420
400 IF B(J+TA)<>B(J+TB) THEN 420
410 T=T+5
420 RETURN
430 GOSUB 2640
440 GOSUB 1990
450 GOSUB 1530
460 CC$=CA$:CP$=PA$
470 IF FM=2 THEN CC$=CB$:CP$=PB$
480 IF NP=1 THEN PRINT HM$;TAB(37);CB$;SK
490 PRINT HM$;BS$;CC$;CP$;
500 IF NP=1 AND FM=2 THEN 520
510 GOTO 920
520 FOR J=1 TO 4
530 PRINT BL$;PB$;:NEXT
540 IF B(15)=0 THEN MV=5:GOTO 900
550 IF B(14)=0 THEN MV=4:GOTO 900
560 FOR MV=1 TO 9:J=90+MV:E(MV)=10
570 IF B(J)<>0 THEN E(MV)=0:MS(MV)=0:GOTO 730
580 IF B(J-10)<>0 THEN 600
590 J=J-10:GOTO 580
600 MS(MV)=J:IF SK<2 THEN 660
610 B(J)=1:FM=1:GOSUB 140:B(J)=0:FM=2
620 IF W=1 THEN E(MV)=77
630 IF SK<3 THEN 660
640 B(J)=2:GOSUB 140:B(J)=0
650 IF W=1 THEN E(MV)=99:GOTO 730
660 T=0:FOR K=-1 TO 1
670 T=T+B(J-10+K)+3*B(J+K)+B(J+10+K)
680 NEXT
690 IF MV=4 OR MV=6 THEN T=T+1
700 IF MV=5 THEN T=T+1
710 E(MV)=E(MV)+T
720 E(MV)=E(MV)+INT(4*RND(1))
730 IF MV<9 THEN PRINT BR$;PB$;
740 NEXT
750 K=0:T=-9:FOR MV=1 TO 9
760 IF E(MV)>T THEN T=E(MV):K=MV
770 NEXT
780 MV=K:J=MS(MV)
790 IF B(J+10)<>0 THEN 870
800 IF T>=77 THEN 870
810 IF SK=4 AND S4=0 THEN S4=1:GOTO 250
820 IF J+10>HP THEN SH=HP:HP=J+10
830 B(J)=2:B(J+10)=1:FM=1:GOSUB 140
840 IF J+10>SH THEN HP=SH
```

```
850 FM=2:B(J)=0:B(J+10)=0
860 IF W=1 THEN E(MV)=1:GOTO 750
870 PRINT CL$;CHR$(32);HM$;BS$;PB$;
880 FOR J=1 TO 4:PRINT BL$;PB$;:NEXT
890 IF MV=1 THEN 1060
900 FOR J=1 TO MV-1:PRINT BR$;PB$;:NEXT
910 GOTO 1060
920 MV=5:GOSUB 1510
930 GET R$:IF LEN(R$)=0 THEN 930
940 IF R$=ML$ THEN 980
950 IF R$=MR$ THEN 1020
960 IF R$=MU$ THEN 1060
970 GOTO 930
980 MV=MV-1
990 IF MV<1 THEN MV=1:GOTO 930
1000 PRINT BL$;CP$;
1010 GOTO 930
1020 MV=MV+1
1030 IF MV>9 THEN MV=9:GOTO 930
1040 PRINT BR$;CP$;
1050 GOTO 930
1060 J=90+MV
1070 IF B(J)<>0 THEN 930
1080 PRINT BU$;CP$;
1090 IF B(J-10)<>0 THEN 1130
1100 J=J-10:PRINT BU$;CP$;
1110 FOR K=1 TO 20:NEXT
1120 GOTO 1080
1130 B(J)=FM
1140 IF J>HP THEN HP=J:SH=HP
1150 S4=0
1160 GOSUB 140:IF W<>0 THEN 1260
1170 T=0:FOR J=91 TO 99
1180 IF B(J)=0 THEN T=1
1190 NEXT:IF T=1 THEN 1220
1200 PRINT HM$;BS$;SC$;"**TIE GAME**"
1210 GOTO 1320
1220 GOSUB 1240
1230 GOTO 460
1240 FM=FM+1:IF FM=3 THEN FM=1
1250 RETURN
1260 PRINT HM$;BS$;CP$;SPC(1);"WINS!"
1270 PRINT SC$
1280 IF FM=1 THEN WA=WA+1
1290 IF FM=2 THEN WB=WB+1
1300 PRINT TAB(20);WA
1310 PRINT TAB(20);WB;
1320 MG$="GAME."
1330 PRINT HM$:PRINT:PRINT:PRINT
1340 IF WA<GW AND WB<GW THEN 1380
1350 MG$="MATCH":WA=0:WB=0
1360 PRINT TAB(34);"MATCH"
1370 PRINT TAB(34);"OVER.":PRINT:PRINT
1380 PRINT TAB(34);"PRESS"
```

```
1390 PRINT TAB(34);"A KEY"
1400 PRINT TAB(34);"FOR A"
1410 PRINT TAB(34);"NEW"
1420 PRINT TAB(34);MG$
1430 GOSUB 1510
1440 GET R$:IF LEN(R$)=0 THEN 1440
1450 IF R$="Q" THEN 2820
1460 IF R$="R" THEN CLR:GOTO 430
1470 PRINT CHR$(147)
1480 GOSUB 2240
1490 GOSUB 1700
1500 GOTO 460
1510 FOR T=1 TO 10:GET R$:NEXT
1520 RETURN
1530 ED$=CHR$(166)
1540 PA$=CHR$(79)
1550 PB$=CHR$(113)
1560 CA$=CHR$(30)
1570 CB$=CHR$(149)
1580 SC$=CHR$(154)
1590 CE$=CHR$(154)
1600 CL$=CHR$(157):CR$=CHR$(29):CU$=CHR$(145):CD$=CHR$(17)
1610 FOR J=1 TO 21:BS$=BS$+CD$:NEXT
1620 FOR J=1 TO 17:BS$=BS$+CR$:NEXT
1630 BL$=CL$+CHR$(32)+CL$+CL$+CL$+CL$
1640 BR$=CL$+CHR$(32)+CR$+CR$
1650 BU$=CL$+CHR$(32)+CU$+CL$
1660 HM$=CHR$(19)
1670 ML$="1":MR$="2":MU$=CHR$(13)
1680 HL$=CHR$(99)
1690 GW=2
1700 PRINT CHR$(147)
1710 PRINT TAB(2);CHR$(5);TT$;SPC(3);"FROM 'MIND MOVES'"
1720 PRINT TAB(2);CE$;
1730 FOR J=1 TO 31
1740 PRINT ED$;:NEXT
1750 PRINT
1760 PRINT TAB(2);
1770 FOR J=1 TO 11:PRINT ED$;CHR$(32);CHR$(32);:NEXT
1780 PRINT
1790 FOR J=1 TO 8
1800 PRINT TAB(2);ED$;ED$;TAB(31);ED$;ED$
1810 PRINT TAB(2);ED$;TAB(32);ED$
1820 NEXT
1830 PRINT TAB(2);ED$;ED$;TAB(31);ED$;ED$
1840 PRINT
1850 PRINT SC$;
1860 FOR J=1 TO 19:PRINT HL$;:NEXT
1870 PRINT CHR$(178);HL$;HL$;HL$;CHR$(174)
1880 PRINT SC$;"GAME";SPC(4);CA$;PA$;SPC(1);NA$;TAB(19);
1890 PRINT SC$;CHR$(98);WA;TAB(23);CHR$(98);
1900 PRINT TAB(25);"BEST";GW;"OF";GW+GW-1
1910 PRINT"SCORE";SPC(3);CB$;PB$;SPC(1);NB$;TAB(19);
1920 PRINT SC$;CHR$(98);WB;TAB(23);CHR$(98);
```

```
1930 PRINT TAB(25);"GAME MATCH.";
1940 FOR J=11 TO 99:B(J)=0:NEXT
1950 FOR J=0 TO 10:B(J)=-1:B(J+100)=-1:NEXT
1960 FOR J=20 TO 90 STEP 10:B(J)=-1:NEXT
1970 HP=11:SH=HP
1980 RETURN
1990 PRINT CHR$(147)
2000 PRINT TT$;SPC(3);"OPTION MENU"
2010 PRINT
2020 PRINT"1 - 1 PLAYER GAME (YOU VS. COMPUTER)"
2030 PRINT"2 - 2 PLAYER GAME"
2040 PRINT"3 - CHANGE COMPUTER SKILL LEVEL"
2050 PRINT"4 - INSTRUCTIONS FOR PLAYING GAME"
2060 PRINT"5 - END PROGRAM"
2070 PRINT
2080 PRINT"PRESS 1, 2, 3, 4, OR 5"
2090 GET R$:IF LEN(R$)=0 THEN 2090
2100 IF R$<"1" OR R$>"5" THEN 2090
2110 PRINT R$
2120 IF R$="5" THEN 2820
2130 NP=VAL(R$)
2140 ON NP GOTO 2150,2150,2390,2480
2150 IF NP=1 THEN 2220
2160 PRINT:NA$="":NB$=""
2170 INPUT"NAME OF FIRST PLAYER";NA$
2180 IF LEN(NA$)=0 THEN 2170
2190 INPUT"SECOND PLAYER";NB$
2200 IF LEN(NB$)=0 THEN 2190
2210 NA$=LEFT$(NA$,9):NB$=LEFT$(NB$,9)
2220 PRINT
2230 IF NP=1 THEN NA$="YOU":NB$="COMPUTER"
2240 IF PF=0 THEN 2310
2250 PF=PF+1:IF PF>2 THEN PF=1
2260 FM=PF
2270 PRINT"THIS TIME THE FIRST MOVE GOES TO":PRINT
2280 IF FM=1 THEN PRINT NA$:GOTO 2300
2290 PRINT NB$
2300 GOTO 2370
2310 PRINT"BY RANDOM CHOICE, THE FIRST MOVE GOES TO"
2320 FOR J=1 TO 4:FOR K=1 TO 400:NEXT
2330 PRINT".";CHR$(32);:NEXT J:FM=1
2340 J=RND(1):IF J>.5 THEN FM=2:PRINT NB$:GOTO 2360
2350 PRINT NA$
2360 PF=FM
2370 FOR J=1 TO 2500:NEXT
2380 RETURN
2390 PRINT:PRINT:PRINT
2400 PRINT"THE CURRENT SKILL LEVEL IS";SK
2410 PRINT:PRINT"LEVEL 1 IS THE WEAKEST OPPONENT,"
2420 PRINT" AND LEVEL 4 IS THE STRONGEST."
2430 PRINT:PRINT"ENTER 1, 2, 3, OR 4"
2440 GET R$:IF LEN(R$)=0 THEN 2440
2450 IF R$<"1" OR R$>"4" THEN 2440
2460 SK=VAL(R$)
```

```
2470 PRINT R$:PRINT:PRINT:GOTO 2000
2480 PRINT:PRINT
2490 PRINT TT$;SPC(2);"MINI-INSTRUCTIONS":PRINT
2500 PRINT"THE OBJECT OF THE GAME IS TO GET FOUR"
2510 PRINT"BALLOONS IN A ROW -- HORIZONTALLY,"
2520 PRINT"VERTICALLY, OR DIAGONALLY.  THE TWO"
2530 PRINT"PLAYERS ALTERNATE RELEASING BALLOONS TO"
2540 PRINT"RISE TO THE HIGHEST AVAILABLE POSITION"
2550 PRINT"IN THE SELECTED COLUMN.  MOVE YOUR"
2560 PRINT"BALLOON LEFT OR RIGHT WITH THE KEYS"
2570 PRINT"1 (LEFT) AND 2 (RIGHT).  RELEASE THE"
2580 PRINT"BALLOON WITH THE RETURN KEY.":PRINT
2590 PRINT"FULL DETAILS ARE IN THE BOOK"
2600 PRINT"'MIND MOVES' FROM DILITHIUM PRESS."
2610 PRINT:PRINT"PRESS A KEY TO CONTINUE."
2620 GET R$:IF LEN(R$)=0 THEN 2620
2630 GOTO 1990
2640 PRINT CHR$(147);CHR$(142);CHR$(8)
2650 POKE 53269,0:POKE 53280,7:POKE 53281,7
2660 TT$="V E R T I G O"
2670 PRINT:PRINT
2680 PRINT TAB(12);TT$
2690 PRINT:PRINT
2700 PRINT TAB(4);"COPYRIGHT 1984 DILITHIUM PRESS"
2710 PRINT:PRINT:PRINT:PRINT
2720 PRINT TAB(6);"FROM THE BOOK 'MIND MOVES'"
2730 PRINT:PRINT
2740 PRINT TAB(5);"BY TOM RUGG AND PHIL FELDMAN"
2750 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
2760 PRINT TAB(9);"PRESS A KEY TO BEGIN"
2770 GET R$:IF LEN(R$)=0 THEN 2770
2780 K=RND(-TI)
2790 J=1:FM=0:MV=0:W=0:T=0:SK=2:I=1
2800 DIM B(110),E(9),MS(9)
2810 RETURN
2820 POKE 53280,14:POKE 53281,6
2830 PRINT CHR$(147);CHR$(9)
9990 END

READY.
```

EASY CHANGES

1. You can change the colors and characters used for the balloons, playing area edge, and scoreboard by changing the values in parentheses in lines 1530 through 1590. See the Main Variables section for an explanation of what each variable is used for, and see Appendix C of the

Commodore 64 Programmer's Reference Guide for a list of which numbers stand for which colors and characters. For example, to alter the colors of the two players' balloons from green and brown to white and black, make these changes:

```
1560 CA$=CHR$(5)
1570 CB$=CHR$(144)
```

2. If you would rather play a match of the best-four-out-of-seven games instead of the best-two-out-of-three, change line 1690:

```
1690 GW=4
```

3. When you play against the computer, a certain amount of randomness is used at all skill levels to prevent the computer from always making exactly the same moves in the same situations. Increase the randomness by increasing the 4 in line 720. Try a value of 6 or 8. Decrease the randomness by changing the 4 to 2 or 3. Eliminate it by changing the 4 to 1.

4. The computer will always move to the center column or the column just to the left of center if it has not been moved to yet. Eliminate this by deleting lines 540 and 550.

5. Change the keys that cause the players' balloons to be moved left, right, and up by changing line 1670. If, for example, you would like the keys L, R, and U to cause movement left, right, and up, make this change:

```
1670 ML$="L":MR$="R":MU$="U"
```

6. Eliminate the display of the computer's skill level by deleting line 480.

PROGRAMMER'S NOTEBOOK

VERTIGO is programmed using some techniques common to many board games that involve moving tokens and trying to optimize position. The board (playing area) is represented by the B array, which has 110 elements. Elements 11 through 99 (excluding multiples of 10) represent legal positions on the board, while zero through 10 and 100 through 110 are used to indicate the immediate out-of-bounds. Each element contains a value to indicate the state of that position on the board. Zero means the position is a legal board position that is currently not occupied by either player. Values of one and two indicate occupation by players one and two. A value of minus one means the position is not a legal one (i.e., it is beyond the edge of the playing area).

Element 11 is the upper left corner of the playing area, 19 is the upper right, 91 is the lower left, and 99 is the lower right. This numbering scheme results in the first digit indicating the row number and the second digit indicating the column number of the position on the board.

The program uses character type graphics, rather than sprites, and controls the positioning of the balloons by printing strings of cursor movement characters. It moves a balloon by printing a blank character on it to erase it, printing a string of cursor movement characters in the appropriate direction, and printing another balloon.

When the computer is the opponent, its move is determined based on the skill level in use. Each possible move (columns one through nine) is evaluated in turn, and a numeric value indicating the strength of the move is calculated. These nine values are saved in the E array. The move with the highest value is selected unless making that move would give the opponent a victory if the opponent moved in the same column afterwards. In that case, the move with the next highest value is used.

At skill level 1, the evaluation value for a move is based entirely on an analysis of the proximity of the move to pre-

vious moves, plus a slight favoritism for columns 4, 5, and 6, plus a small random factor to prevent the computer from always making the same move in the same situation. The computer doesn't look to see if its move will win or block a victory, but only to see if there are a lot of earlier moves in adjacent positions.

For skill level 2, the computer in addition checks each move to see if it can make a move that blocks a victory by its opponent. If so, that move is given a high evaluation number (77).

Skill level 3 adds a check to see if each move would result in a victory by the computer. If so, each such move is given an even higher evaluation number (99).

At skill level 4, all of the above evaluating is done, but additional factors are added based on whether each move will either block an opponent's two-in-a-row, or extend the computer's own two-in-a-row to three.

MAIN ROUTINES

140-240	Subroutine to determine if a move to B(J) would result in a victory. If so, W is set to one.
250-420	Subroutine to do the added analysis of skill level four.
430-510	Mainline logic to call routines that display the title screen, initialize variables, display the option menu, and go to the proper routines for computer vs. human or human vs. human.
520-910	Determines the computer's move.
520-530	Moves computer's balloon from the position below column five to below column one.
540-550	Immediately selects column five or four if the top row is unoccupied.
560-740	Scans the moves in columns one through nine, determining the evaluation value for each.
750-780	Selects the highest evaluation value and corresponding move number.
790-860	Selects MV as the move if the evaluation is high enough. Otherwise, checks to see if the subsequent

	move by opponent in the same column would win. If so, discards move and picks the next-best one.
870-910	Moves computer's balloon to the position beneath the column selected.
920-970	Processes human player's movement of balloon left, right, and up, based on keys pressed.
980-1010	Moves human player's balloon left one column.
1020-1050	Moves human player's balloon right one column.
1060-1150	Moves either player's balloon up to the highest available position in the column. Updates board.
1160-1250	Checks if anyone won the game or if a tie occurred. Goes back to get the other player's move if not.
1260-1420	Shows who won, updates scoreboard, sees if the match has ended.
1430-1500	Waits for a key to be pressed for next game/match. Starts new game.
1510-1520	Subroutine to empty the keyboard buffer.
1530-1980	Subroutine to initialize variables, display playing screen, initialize board.
1990-2630	Subroutine to display the option menu and process responses, determine who goes first, etc.
2640-2810	Subroutine to display title screen, initialize some variables.
2820-9990	Resets screen colors and ends program.

MAIN VARIABLES

B	VERTIGO board (playing area) array.
E	Evaluation value array.
MS	Array of move subscripts. Contains subscripts of the nine possible moves.
MV	Move currently being evaluated (column number).
FM	Player who gets first move; also current mover.
J, K	Work and subscript variables.
T, W	Temporary work variables.
SK	Skill level.
I	Constant one.
NP	Number of human players in the game (1 or 2).
PF	Player who went first in the game.

GW	Number of games needed to win the match.
TA, TB	Temporary work variables for skill level 4 analysis.
HP	Highest position occupied in B array so far.
SH	Work variable to save HP.
TT$	Program title (with alternate blanks).
R$	Reply from keyboard.
NA$, NB$	Names of the two players.
ED$	Graphics character for edge of playing area.
PA$, PB$	Graphics balloons used for the two players.
CA$, CB$	Colors of the two players.
SC$	Scoreboard color.
CE$	Color for the playing area edge character.
CL$, CR$, CU$, CD$	Cursor movement strings for moving left, right, up, and down.
BS$	String to move the cursor to the balloon start position.
BL$, BR$, BU$	Strings to move a balloon left, right, and up.
HM$	String to move the cursor to the home (upper left corner) position.
ML$, MR$, MU$	Keys pressed by the player to move left, right, up.
HL$	Horizontal line graphics character.
CC$, CP$	Current player's color and graphics character.
MG$	String indicating whether it was a match or a game that just ended.

SUGGESTED PROJECTS

1. Alter the computer's playing strategy, or add higher skill levels.
2. Switch the game to a different-sized playing area, either larger or smaller than the nine-by-nine area used in this version.

APPENDIX A
Software Backup and Loading Instructions

WHAT YOU NEED TO RUN THE PROGRAMS

1. Commodore 64 computer with a single disk drive. These instructions assume you have the Commodore VIC-1541 disk drive. If you have another type, the procedure to follow may differ.
2. The dilithium Mind Moves diskette that is the companion to the book.
3. A blank diskette that has been formatted with the NEW command in accordance with the instructions in the *VIC-1541 Single Drive Floppy Disk User's Manual.* Here is a command to do this. (Don't confuse the letter O with the number 0.)

```
OPEN 15,8,15,"NEW0:MM,01"
```

PRELIMINARY STEPS

Make a working (backup) copy of the dilithium disk.

1. Turn on the Commodore 64 computer, the disk drive, and the TV set or monitor.
2. Put the dilithium disk into the disk drive and close the disk drive door.

3. Enter this command:

   LOAD"AEROJAM",8

4. The Commodore 64 will display SEARCHING FOR AEROJAM on the screen, then LOADING, then (after 20-60 seconds) READY.
5. Remove the dilithium disk from the disk drive, and insert your formatted blank diskette. This will become the working copy of your diskette.
6. Enter this command:

   SAVE"AEROJAM",8

7. The computer will display SAVING AEROJAM on the screen, then (after 20-60 seconds) READY.
8. Repeat steps 2 through 7 for each of the other programs on the dilithium disk. The programs are:

AEROJAM	QUIKWORD
ESCAPADE	VERTIGO
HOTSHOT	QMAKER
QUIKWIT	SEQCOPY

9. You now have a disk with a copy of all the Mind Moves *programs* (six game programs plus two utility programs). You now must copy the four QUIKWIT *data files* onto this disk. This requires the use of a special utility program such as the one we have provided on your disk. This program, called SEQCOPY, copies a sequential data file from one diskette to another. (Sequential data files are the type used by QUIKWIT.)

   SEQCOPY is easy to use. It begins by asking you the name of the source file. This is the file you wish to have copied. Then it asks the name of the target file. This is

the name you wish to call the data file on the new disk. Typically the two names are identical.

Four sequential data files accompany the QUIKWIT program, and you need to run SEQCOPY four times to copy all of them. Their names are witspo, witgen, witsci, and witart. When entering these names for SEQCOPY, keep the titles all in lowercase and don't include any punctuation marks.

After you provide the names, SEQCOPY tells you to put in place the source disk (the one with the original data file) then to hit a keyboard key to signal you're ready. Next, the program tells you it's reading the file. When it's done, it indicates how many data records (or lines) it read. This should be 701 records for the QUIKWIT files as provided with the software (one line of the category name plus 100 complete questions each consisting of two lines for the question proper and five for the answer list.)

Next, SEQCOPY requests that you put the target disk in the disk drive then hit any key. The program writes the new copy and indicates when it's finished. Voila!

10. Once the eight programs and the four data files are copied onto your working disk, put the original dilithium disk in a safe place. Use only your working copy. That way you can easily recreate another working copy in case the first working copy is damaged, lost, or (eventually) worn out. Label your working copy as follows:

    MIND MOVES COPY.
    COPYRIGHT 1984 DILITHIUM PRESS.

    Note that these programs and data files are copyrighted, and it is illegal for you to make copies to give or sell to others, or to allow others to make copies of your disks.

USING THE PROGRAMS

1. Turn on the Commodore 64 computer, the disk drive, and the TV set or monitor.
2. Select the program you would like to run. Read about the program in *Mind Moves* so you will know how it works. Then, load the program from disk into the Commodore 64's memory with the command:

   LOAD"program",8

   Substitute the chapter name for the word *program* in the LOAD command. For example, if you decide to try VERTIGO, the command is:

   LOAD"VERTIGO",8

   After the program is loaded and you see the READY message, enter the command RUN and press [RETURN] to begin the game.
3. After you end a game and see the READY message on the screen, go back to step 2 to try the next program.

TROUBLESHOOTING

1. Details about what to do when things don't work right are shown in the How To Use This Book section at the front of the book. Read the sections of the manuals mentioned there to be sure you are operating the Commodore 64 properly, and read the book chapter for the program you are using to be sure you fully understand how the program is supposed to work.
2. If all else fails and you have carefully read all these instructions, refer to the Errata Offer section in the back of the book.

APPENDIX B
QUIKWIT Cassette Version

The QUIKWIT chapter presents a disk-based version of the game. If your system is cassette-based and you are typing in the programs yourself, this appendix shows you how to modify the program to use cassette data files. We must caution you, however, that cassette data files are slow and relatively unreliable. If you have a choice, by all means use a disk. Some other differences between the diskette and cassette versions are explained later in this appendix.

The cassette-based version of QUIKWIT is similar to the disk-based version. If you're typing in the program, use the listing from the QUIKWIT chapter, except use the following statements as presented here to replace and add to those in the QUIKWIT chapter. Most of these statements replace statements of the same line numbers as those in the QUIKWIT chapter listing. A few, however, are new line numbers completely.

```
 250 fz=1
 350 dc(1)=0:dc(2)=1:dc(3)=0:dc(4)=0
 590 add=704:page=add/64
2980 k=0
2990 k=k+1:if dc(k)=0 and k<=nf then 2990
3000 for q=1 to 24
3010 input#1,a$(1,q):input#1,a$(2,q):
     for z=1 to 200:next
3020 for j=1 to 5:input#1,b$(j,q):next
3025 for z=1 to 200:next
3050 goto 3400
```

```
3240 print bk$;"Initializing cassette files."
3250 cd=7:gosub 3180
3260 rem
3280 k=0
3290 k=k+1:if dc(k)<>0 then 3340
3300 if k<nf then 3290
3310 cd=15:cr=0:gosub 3180
3320 print bk$;"Fatal error -- no files requested."
3330 gosub 2530:cd=21:cr=0:gosub 3180:end
3340 open 1,1,0,f$(k):q=dc(k)
3350 if q<0 then q=int(fz*rnd(1))+1
3360 input#1,c$(k):if q=1 then qn(k)=0:goto 3430
3370 qn(k)=q-1:p=7*qn(k):for j=1 to p
3380 input#1,r$:if st<>0 then 3400
3390 next:goto 3430
3400 close 1:cd=15:cr=0:gosub 3180
3410 print bk$;"Fatal error, end of tape reached.":end
3420 rem
3430 print"Hit a keyboard key."
3432 get r$:if r$="" then 3432
3434 return
3680 print"No.";tab(5);"Name"
3690 for k=1 to nf:print k;f$(k):next:return
3870 next:if q>=1 then 4060
```

Don't forget to have your computer in lowercase text mode before you type in these changes. To do this, type in the following statement:

```
PRINT CHR$(14) [RETURN]
```

Of course, to use the cassette version of QUIKWIT, you must have the data files available on cassette. See Appendix C for an explanation of how to create them and back them up. You also can use instructions in Appendix C to create your own additional QUIKWIT question files.

There are some differences between the way the

QUIKWIT program operates in the cassette mode and the way it operates in the diskette mode. Only one file can be active at a time in the cassette mode, and you cannot read past the end of the file, or an error results. This requires you to be aware of where you are in each file.

Each time the program reads data from the cassette, a block of 24 questions is read. This means each data file must contain at least 24 questions, and you must start each block read at a question number at least 24 questions before the end of the file. Thus, be careful when using main menu option 3 that you don't specify too large a starting number. The random start request in option 3 confines the starting number to a value no greater than that set by the variable fz in line 250. If your data file contains more than 24 questions, change the value of fz in line 250 to the number of questions in the file minus 23. This ensures you can't read past the end of the tape on the first block read when the random start is selected. Also, if you request that more than one file be turned on, only the first will actually be used. If you don't use option 3 at all, the file request will automatically be for the data file witgen (General Information) beginning at question one.

Before you select main menu option 1 or 2 to start the game, try to have the cassette tape positioned properly. You can always have the tape at the beginning, because the cassette version of QUIKWIT will search the tape for your requested file. However, if your desired file is in the middle of the tape, this search process will be tedious. If you know approximately where the file starts, position the tape somewhat in front of it. This will greatly cut down the search time.

After you select option 1 or 2, the program will request that you press the play button on the recorder. When you do so, the display screen will blank out while the cassette tape is searched for your requested file. When the file is found and initialized, the program will ask that you press a keyboard key to continue. Then the first block of 24 questions will be read from the cassette tape and QUIKWIT will be off and running.

APPENDIX C
QUIKWIT Data Files

QUIKWIT DATA FILE CREATION

QUIKWIT game program questions are stored in data files. If you've purchased our software package, these files are contained on your diskette. If not, you have to create these files. You use the program in this appendix to do just that. Even if you have our software, you may want to use this program and create some of your own question files to spring on unsuspecting friends.

The program presented here and included on dilithium's ready-to-run software is called QMAKER (for Question Maker). It prompts you to enter questions and creates files for use with QUIKWIT on either disk or cassette.

QUIKWIT assumes that four data files are present. These files are called witspo, witgen, witsci, and witart. They contain questions in the categories of sports, general information, science, and the arts, respectively. Should you desire some other number of data files (more than four if you add some of your own, fewer than four if you don't type them all in) or wish to rename some of the files, you must make some changes to QUIKWIT.

Please refer to the program listing of QUIKWIT. Here's how to tell it what files are available. First create all the data files you desire using QMAKER. Then set the variable nc in line 270 of QUIKWIT to the total number of data files available. The string array f$ contains the names of each of these data files. They are enumerated in lines 300-330 of

QUIKWIT. Change these names or add or subtract some as appropriate. Once this is done, QUIKWIT should be ready to run.

Now let's discuss how to use the QMAKER program. When you run it, the first thing it does is request the name of the data file you wish to create. Enter this name followed by [RETURN]. Make sure you have your diskette or cassette tape properly positioned, as data will be written to it shortly. The program next asks you whether a disk or cassette will be used. Type D or C (followed by [RETURN]) as appropriate.

The screen display shifts to uppercase/lowercase mode. Enter the category name for the file of questions you're about to enter. The display now changes to its basic data entry screen. There are four parts to this screen. The upper box shows the name of the file being created and the current question number being entered. A question box and answer box hold the main text of each question and its list of answers. And the bottom box indicates what action is desired next.

A check mark prompts you for each input line one at a time. A diagonal line symbol acts as a cursor to show where the next character you type will appear. Type in your data using uppercase and lowercase characters. The cursor will not let you go past the end of any expected line. Hit [RETURN] to end each line of input. If an input line is to be blank, just hit [RETURN] immediately. Should you make a typo on a line, use the [DEL] key to back up the cursor and retype your data.

The quote mark cannot be displayed with this program. When you use the quote key ("), an up arrow appears where the quote mark should go. This is only for the purpose of QMAKER. The data file will be written properly, and the quote mark will appear as expected when QUIKWIT is run.

Each question is input in the two lines of the question box. Be sure to space your questions between the two lines as desired. If the second line is not needed, simply hit [RETURN] when prompted for it.

You must now type in the answer possiblities. You can use up to five but must, of course, have at least two. Each possible answer goes on a separate line — the numerals 1 to 5 are already there for you. *Place the correct answer first, in the space for answer 1.* QUIKWIT will shuffle your answers when it presents each question.

After each question is entered, along with its answers, QMAKER wants to know if it was typed OK. You have three possible replies. Type y to indicate yes, that you're ready to type in the next question. Type n to indicate no, that you would like to retype it. (Use this option if you find a mistake.) Type f to indicate that question is okay and that the file is now finished. This will write the last piece of data on the disk or cassette, close the file, and exit QMAKER.

If you're creating files on disk, there is no restriction on the minimum number of questions for each file. If you're using a cassette, however, each file must contain at least 24 questions.

The listing of QMAKER is printed in lowercase text mode. Get your computer in this mode before typing in the program. To do this, type in the following statement:

PRINT CHR$(14) [RETURN]

Following the listing of QMAKER are some sample questions to get you going on each of the four data files. Space limits us to 25 questions for each file. Many more questions come with the software package.

QMAKER Program Listing

```
100 rem: qmaker
110 rem: copyright 1984 dilithium press
120 rem: by phil feldman and tom rugg
130 poke 53269,0:poke 53280,1:poke 53281,13
140 print chr$(28);chr$(147)
150 print chr$(8);chr$(142)
160 print "quikwit data file creator"
170 print:input"name of file to create";f$
```

```
180 if len(f$)=0 then 170
190 close 4:close 15:print:print
200 print:print"make sure disk or cassette is ready."
210 print:input"what medium (d=disk or c=cassette)";d$
220 if d$="d" then 250
230 if d$<>"c" then 210
240 open 4,1,2,f$:goto 280
250 open 15,8,15:open 4,8,4,"0:"+f$+",s,w"
260 input#15,t$,r$
270 if t$<>"00" then print:print"error - ";r$:end
280 print chr$(147);chr$(14)
290 print"Now in lower case mode"
300 print:print"Input the question category for this"
310 print"file.  Examples might be:":print
320 print"   General Information"
330 print"   Sports"
340 print"   Science and Nature"
350 print"   Arts and Entertainment"
360 print:input"Category";c$
370 if len(c$)=0 then 360
380 print#4,c$
390 dim a$(2),b$(5)
400 n=0
410 z$=chr$(129)+chr$(18)
420 for j=1 to 40:z$=z$+chr$(32):next
430 y$="":for j=1 to 40:y$=y$+chr$(29):next
440 print chr$(147);chr$(14)
450 poke 53280,8:poke 53281,15
460 n=n+1:gosub 720
470 m=38
480 for q=1 to 2:print chr$(19)
490 for p=1 to 7+q:print chr$(17);:next
500 print chr$(18);chr$(144);chr$(186);chr$(146);
510 gosub 930:a$(q)=t$:print:next
520 m=31
530 for q=1 to 5:print chr$(19)
540 for p=1 to 11+q:print chr$(17);:next
550 print chr$(18);chr$(186);chr$(146);
560 for p=1 to 5:print chr$(29);:next
570 gosub 930:b$(q)=t$:print:next
580 for j=1 to 4:print chr$(17);:next
590 print"  Is question OK (y, n, or f)?";
600 get r$:if len(r$)=0 then 600
610 if r$="y" or r$="f" then 650
620 if r$="n" then 640
630 goto 600
640 n=n-1:goto 460
650 print#4,a$(1)
660 print#4,a$(2)
670 for j=1 to 5:print#4,b$(j):next
```

```
680 j=fre(0)
690 if r$="y" then 460
700 close4:close15
710 end
720 print chr$(147)
730 print chr$(28);" QMAKER -- creating file:";f$
740 print
750 print" Question #";n:print
760 print" Use [RETURN] to end each input line"
770 print:print z$;
780 print left$(z$,9);"question box";left$(z$,23);
790 print left$(z$,3);left$(y$,38);chr$(32);
800 print left$(z$,3);left$(y$,38);chr$(32);
810 print z$;left$(z$,09);"answer box";left$(z$,25);
820 for j=1 to 5
830 print left$(z$,3);chr$(144);chr$(146);j;
840 print "-";chr$(32);left$(y$,31);left$(z$,5);
850 next
860 print z$
870 print chr$(145);chr$(31);"  Next action"
880 print"    y - yes, proceed to next question"
890 print"    n - no, redo this one"
900 print"    f - yes, but file is now finished"
910 print chr$(19);
920 return
930 t$="":fg=0
940 print chr$(169);chr$(157);
950 get r$:if len(r$)=0 then 940
960 if len(t$)<m then 990
970 if asc(r$)=13 then 1120
980 goto 940
990 if asc(r$)<>20 and asc(r$)<>148 then 1030
1000 if len(t$)=0 then 940
1010 print chr$(32);chr$(157);chr$(157);
1020 t$=left$(t$,len(t$)-1):goto 940
1030 if asc(r$)=17 or asc(r$)=29 or asc(r$)=145 then 940
1040 if asc(r$)=157 or asc(r$)=147 or asc(r$)=19 then 940
1050 if r$="," then print",";:t$=t$+chr$(100):fg=1:goto 940
1060 if r$=":" then print":";:t$=t$+chr$(101):fg=1:goto 940
1070 if r$=";" then print";";:t$=t$+chr$(102):fg=1:goto 940
1080 if asc(r$)=34 then print chr$(94);:t$=t$+chr$(103)
1090 if asc(r$)=34 then fg=1:goto 940
1100 if asc(r$)=13 then 1120
1110 t$=t$+r$:print r$;:goto 940
1120 if len(t$)=0 then t$=chr$(104)
1130 print chr$(32)
1140 if fg=1 then t$=chr$(105)+t$
1150 return

ready.
```

QUIKWIT Questions

File:WITSPO
Category:Sports and Games

Question # 1
What is the highest anyone ever pole vaulted with a bamboo pole?
1 - Between 15-17 feet
2 - Between 11-13 feet
3 - Between 13-15 feet
4 - Less than 11 feet
5 - Over 17 feet

Question # 2
Duke Kahanamoku was a pioneer in what sport?
1 - Surfing
2 - Soccer
3 - Football
4 - Wrestling
5 - Diving

Question # 3
What basketball player was known as "Hondo"?
1 - John Havlicek
2 - Hal Greer
3 - George Mikan
4 - Austin Carr
5 - Bobby Jones

Question # 4
Who pitched 12 perfect innings and lost the game?
1 - Harvey Haddix
2 - Catfish Hunter
3 - Jim Bunning
4 - Don Larsen
5 - Don Drysdale

Question # 5
How many warm-up pitches is a relief pitcher allowed upon entering a game?
1 - 8
2 - 6
3 - 10
4 - 12
5 - one minute's worth

Question # 6
How much money is given to a player at the start of Monopoly?
1 - $1500
2 - $1142
3 - $2000
4 - $1750
5 - $1350

Question # 7
What position did Sammy Baugh play?

1 - Quarterback
2 - Forward
3 - Third Baseman
4 - Goalie
5 - Pitcher

Question # 8
What is Red Auerbach's real first name?
1 - Arnold
2 - Harvey
3 - John
4 - Robert
5 - Stanley

Question # 9
Who stole home more than any other major leaguer?
1 - Ty Cobb
2 - Lou Brock
3 - Maury Wills
4 - Luis Aparicio
5 - Owen Wilson

Question # 10
The heaviest fresh water white catfish ever caught with rod & reel weighed...
1 - 17 lb. 7 oz.
2 - 36 lb.
3 - 122 lb. 9 oz.
4 - 81 lb. 14 oz.
5 - 58 lb. 9 oz.

Question # 11
Which chess piece cannot move backwards?
1 - Pawn
2 - Rook
3 - King
4 - Knight
5 - Bishop

Question # 12
The Vare Trophy is awarded to.....

1 - Female golfers
2 - Highest average bowlers
3 - Hockey goalies
4 - Football defensemen
5 - College football coaches

Question # 13
Who was head coach at UCLA during their basketball empire?
1 - Wooden
2 - Rupp
3 - Driesell
4 - Meyer
5 - Knight

Question # 14
Which baseball team won a record 111 games in one season in 1954?
1 - Indians
2 - Yankees
3 - Tigers
4 - Red Sox
5 - White Sox

Question # 15
Who has the most career base on balls?

1 - Babe Ruth
2 - Lou Gehrig
3 - Ty Cobb
4 - Hank Aaron
5 - Stan Musial

Question # 16
Which major tournament has Arnold Palmer never won?
1 - PGA
2 - US Open
3 - Masters
4 - British Open
5 - Has won them all

Question # 17
In what city did the Lakers play

```
before Los Angeles
 1 - Minneapolis
 2 - Cincinnati
 3 - Philadelphia
 4 - St. Louis
 5 - Hartford

Question # 18
Who has won the most NBA most valuable
player awards?
 1 - Abdul-Jabbar
 2 - Russell
 3 - Chamberlain
 4 - Malone
 5 - Erving

Question # 19
Which sport's rules are based on
Marquess of Queensbury rules?
 1 - Boxing
 2 - Polo
 3 - Cricket
 4 - Horse Racing
 5 - Baseball

Question # 20
Which sports announcer was NOT a
professional athlete?
 1 - Howard Cosell
 2 - Jim Palmer
 3 - Lynn Swann
 4 - Frank Gifford
 5 - Don Meredith

Question # 21
What pitcher has the most career wins?

 1 - Cy Young
 2 - Walter Johnson
 3 - Grover Alexander
 4 - Christy Mathewson
 5 - Steve Carlton

Question # 22
Who was the first woman named Sports
Illustrated's Sportsman of the Year?

 1 - Billie Jean King
 2 - Chris Evert
 3 - Peggy Fleming
 4 - Wilma Rudolph
 5 - Nancy Lopez

Question # 23
Which driver has won the most
Indianapolis 500's?
 1 - A.J. Foyt
 2 - Parnelli Jones
 3 - Bill Vukovich
 4 - Johnny Rutherford
 5 - Mario Andretti

Question # 24
Which term is NOT common to both
basketball and football?
 1 - Offsides
 2 - Pass
 3 - Field Goal
 4 - Out of bounds
 5 - Turnover

Question # 25
How many laps are there in the Indy
500?
 1 - 200
 2 - 500
 3 - 1
 4 - 100
 5 - 501

File:WITGEN
Category:General Information

Question # 1
Fala was a.....

 1 - Dog
 2 - Goddess
 3 - Drink
 4 - Dance craze
 5 - Song

Question # 2
Night blindness is caused by a
deficiency of which vitamin?
 1 - A
 2 - B
 3 - C
 4 - D
 5 - None of the other choices

Question # 3
Which ship rammed and sank a British
cruiser killing 338 passengers?
 1 - Queen Mary
 2 - Andrea Doria
 3 - Titanic
 4 - Love Boat
 5 - Princess Victoria

Question # 4
There are more Russians than
Americans.
 1 - True
 2 - False
 3 -
 4 -
 5 -

Question # 5
What did the Marlboro Man have
tattooed on his hand?
 1 - Eagle
 2 - Snake
 3 - Flag
 4 - "Mom"
 5 - Ship

Question # 6
Which car was NOT produced by a U.S.
manufacturer?
 1 - Dasher
 2 - Skylark
 3 - Cutlass
 4 - Impala
 5 - Polara

Question # 7
How many years is fourscore and seven?

 1 - 87
 2 - 7
 3 - 47
 4 - 407
 5 - 40

Question # 8
Which country has the SECOND most
television sets in the world?
 1 - USSR
 2 - USA
 3 - Japan
 4 - India
 5 - Canada

Question # 9
Where did the zeppelin Hindenburg
burn?
 1 - New Jersey
```

2 - Germany
3 - Atlantic Ocean
4 - Scotland
5 - California

Question # 10
For what charge was Al Capone finally imprisoned?
1 - Tax Evasion
2 - Murder
3 - Loitering
4 - Treason
5 - Grand larceny

Question # 11
From 1820-1980, which country produced the most immigrants to the U.S?
1 - Germany
2 - Great Britain
3 - Italy
4 - Ireland
5 - U.S.S.R.

Question # 12
What best describes Bebe Rebozo's relationship to Richard Nixon?
1 - Pal
2 - Press officer
3 - Lawyer
4 - Cabinet member
5 - Relative

Question # 13
The Pony Express ran between California and what state?
1 - Missouri
2 - Kansas
3 - Colorado
4 - Oklahoma
5 - Texas

Question # 14
"Corpus delicti" might get you.....

1 - Convicted
2 - Initiated
3 - Lost
4 - Nauseous
5 - Itching

Question # 15
In what field might you use a P.E. ratio?
1 - Stock market
2 - Geometry
3 - Physics
4 - Interior decorating
5 - Basketball

Question # 16
A lavalier is an.....

1 - Ornament
2 - Island
3 - Inconvenience
4 - Automobile
5 - Enemy

Question # 17
Richard Burton refused to eat cooked peas.
1 - False
2 - True
3 -
4 -
5 -

Question # 18
What do Camp Fire Girls sell?
1 - Mints
2 - Cookies
3 - Seeds
4 - Kindling
5 - Magazines

Question # 19
What was the first name of the red-baiting McCarthy?
1 - Joseph
2 - Eugene
3 - Len
4 - Sam
5 - Ian

Question # 20
Which country is NOT landlocked?

1 - Belgium
2 - Afghanistan
3 - Switzerland
4 - Nepal
5 - Luxembourg

Question # 21
Who said, "Give me liberty or give me death?"
1 - Patrick Henry
2 - Benedict Arnold
3 - Nathan Hale
4 - Paul Revere
5 - Paul Revere and the Raiders

Question # 22
Dan Cooper was a.....

1 - Hijacker
2 - Pianist
3 - Lawyer
4 - Civil Rights leader
5 - Supreme Court justice

Question # 23
Which letter is NOT on the telephone dial?
1 - Z
2 - V
3 - J
4 - P
5 - K

Question # 24
In which of these activities do the most Americans participate?
1 - Swimming
2 - Jogging
3 - Tennis
4 - Bowling
5 - Golfing

Question # 25
Who was NOT assasinated?

1 - Batista
2 - Huey Long
3 - Trotsky
4 - Gandhi
5 - None of the other choices

```
File:WITSCI
Category:Science and Nature


Question # 1
Which is the LEAST common blood type
in the U.S.?
 1 - AB
 2 - A
 3 - B
 4 - O positive
 5 - O negative

Question # 2
Which of these men was NOT a Greek
philosopher?
 1 - Aristophanes
 2 - Democritus
 3 - Eratosthenes
 4 - Thales
 5 - Archimedes

Question # 3
Monsoons are always.....

 1 - Heavy winds
 2 - Rainy storms
 3 - All of the other answers
 4 - Humid squalls
 5 - Floods

Question # 4
Euclid's greatest contribution was
to.....
 1 - Mathematics
 2 - Physics
 3 - Chemistry
 4 - Astronomy
 5 - Biology

Question # 5
What is the area of a rectangle three
feet by four feet?
 1 - 12 square feet
 2 - 7 square feet
 3 - 14 square feet
 4 - 36 square feet
 5 - 28 square feet

Question # 6
What is stagflation?

 1 - Hi unemployment-Low inflation
 2 - Low unemployment-Low inflation
 3 - Hi unemployment-Hi inflation
 4 - Low unemployment-Hi inflation
 5 - Bachelor Party

Question # 7
Which is the "Sunshine Vitamin"?

 1 - D
 2 - C
 3 - B1
 4 - A
 5 - E

Question # 8
What is the approximate duration of a
solar day?
 1 - 24 hours
 2 - 12 hours
 3 - 7 days
 4 - 60 minutes
 5 - 30.3 days

Question # 9
How many millimeters in a meter?

 1 - 1000
 2 - 10
 3 - .10
 4 - .100
 5 - 100

Question # 10
An icosahedron has how many sides?

 1 - 20
 2 - 12
 3 - 9
 4 - 100
 5 - 1000

Question # 11
The fallopian tube was first
described by.....
 1 - Fallopius
 2 - Vesalius
 3 - Marcus Welby
 4 - Hippocrates
 5 - Gray

Question # 12
Where are you farthest from the
Earth's center?
 1 - Equator
 2 - North Pole
 3 - South Pole
 4 - Tropic of Cancer
 5 - Tropic of Capricorn

Question # 13
Which animal lives longest?

 1 - Man
 2 - Killer whale
 3 - Turtle
 4 - Asiatic elephant
 5 - Boa constrictor

Question # 14
Braille was invented by a Frenchman
named Louis Braille.
 1 - True
 2 - False
 3 -
 4 -
 5 -

Question # 15
The large intestine is also known as
the.....
 1 - Colon
 2 - Caecum
 3 - Ileum
 4 - Coccyx
 5 - Appendix

Question # 16
Which of these men first believed
that the world was round?
 1 - Pythagoras
 2 - Christopher Columbus
 3 - Archimedes
 4 - Plato
 5 - Aristotle

Question # 17
Which mosquitos bite?

 1 - Females only
```

```
 2 - Males only
 3 - Both males and females
 4 - Neither males nor females
 5 -

Question # 18
Which of these animals can be
venomous?
 1 - All of the other choices
 2 - Ants
 3 - Water moccasins
 4 - Octopi
 5 - Scorpions

Question # 19
Could a Siamese cat see in a pitch
black cave?
 1 - No
 2 - Yes
 3 -
 4 -
 5 -

Question # 20
When did Gutenberg invent the printing
press?
 1 - 1450
 2 - 1300
 3 - 1625
 4 - 1779
 5 - 1824

Question # 21
"D.D.S." is.....

 1 - A dental degree
 2 - An insecticide
 3 - A feminine hygiene spray
 4 - An element in gunpowder
 5 - An enzyme

Question # 22
What does an altimeter measure?

 1 - Height
 2 - Viscosity
 3 - Weight
 4 - Depth
 5 - Moisture

Question # 23
Which state is in both the Western
and Eastern hemispheres?
 1 - Alaska
 2 - Hawaii
 3 - Maine
 4 - Florida
 5 - None of the other choices

Question # 24
A tsuranagakobitozami is a.....

 1 - Shark
 2 - Ion
 3 - Semiconductor
 4 - Measuring device
 5 - Chicken Egg

Question # 25
The decimal equivalent of 2/11 is.....

 1 - .1818
 2 - .2222
 3 - .1616
 4 - .2727
 5 - .1111
```

```
File:WITART
Category:Arts and Entertainment

Question # 1
Most theatrical movies are screened
at.....
 1 - 35 millimeters
 2 - 16 millimeters
 3 - 70 millimeters
 4 - None of the other choices
 5 - 8 millimeters

Question # 2
Barry Sadler sang a #1 song about.....

 1 - Green berets
 2 - Pink houses
 3 - Red balloons
 4 - Blue eyes
 5 - White rooms

Question # 3
On what TV show did Kookie appear?

 1 - 77 Sunset Strip
 2 - Route 66
 3 - Three's Company
 4 - It Takes Two
 5 - Surfside Six

Question # 4
Which of the following was NOT a
gossip columnist?
 1 - Bob Woodward
 2 - Louella Parsons
 3 - Hedda Hopper
 4 - Army Archerd
 5 - Rona Barrett

Question # 5
Who said, "Not so fast, Louis"?

 1 - Humphrey Bogart
 2 - Dean Martin
 3 - Keely Smith
 4 - Mae West
 5 - Roger Moore

Question # 6
Which was NOT an Elvis Presley film
role?
 1 - Sandy Tuttle
 2 - Deke Rivers
 3 - Dr. John Carpenter
 4 - Toby Kwimper
 5 - Joe Lightcloud

Question # 7
"The Girl From Ipanema" is an example
of what type of music?
 1 - Bossa Nova
 2 - Reggae
 3 - Heavy Metal
 4 - Calypso
 5 - Gospel

Question # 8
Elvis Presley never won a Grammy.

 1 - False
 2 - True
 3 -
 4 -
 5 -

Question # 9
Richard Penniman is also known as....
```

1 - Little Richard
2 - Richard Dawson
3 - Richard II
4 - Capt. America
5 - Dick Buttons

Question # 10
Elmer is the original name of.....
1 - Howdy Doody
2 - Bozo the Clown
3 - Charlie Brown (in Peanuts)
4 - Blondie
5 - Goofy

Question # 11
Which rock act is NOT American?

1 - Def Leppard
2 - Journey
3 - Styx
4 - R.E.M.
5 - Quiet Riot

Question # 12
Greystoke was the real last name of...

1 - Tarzan
2 - Green Lantern
3 - Batman
4 - Flash Gordon
5 - Spiderman

Question # 13
Mycroft has a brother named.....

1 - Sherlock Holmes
2 - Tom Swift
3 - James Bond
4 - Mickey Mouse
5 - Indiana Jones

Question # 14
What kind of physical activity did Van Halen encourage in their 1984 hit?
1 - Jumping
2 - Throwing
3 - Running
4 - Surfing
5 - None of the other choices

Question # 15
Which of the following NEVER played a gangster in a movie?
1 - Al Capone
2 - George Raft
3 - Jimmy Cagney
4 - Al Pacino
5 - Humphrey Bogart

Question # 16
Who was Bob Hope's partner in "The Road" movies?
1 - Bing Crosby
2 - Dean Martin
3 - Frank Sinatra
4 - Sammy Davis Jr.
5 - Joe E. Louis

Question # 17
How many horns did the purple people eater have?
1 - 1
2 - 0
3 - 2
4 - 3
5 - 4

Question # 18
What was the sequel to "True Grit?"

1 - Rooster Cogburn
2 - Hondo
3 - Stagecoach
4 - McQ
5 - Conqueror

Question # 19
Who played John Kennedy in "P.T. 109"?
1 - Cliff Robertson
2 - Martin Sheen
3 - Robert Mitchum
4 - Tony Roberts
5 - Dale Robertson

Question # 20
Dustin Hoffman refuses to wear a wristwatch, even in movie roles.
1 - False
2 - True
3 -
4 -
5 -

Question # 21
What was T.S. Eliot's first name?

1 - Thomas
2 - Theodore
3 - Taylor
4 - Stearns
5 - Thackeray

Question # 22
What comic strip takes place in the city of Sebastapol?
1 - Peanuts
2 - Nancy
3 - Bloom County
4 - Family Circus
5 - Blondie

Question # 23
Who played the title role in Mel Brooks' "Young Frankenstein"?
1 - Gene Wilder
2 - Gene Hackman
3 - Peter Boyle
4 - Boris Karloff
5 - Richard Pryor

Question # 24
What group sang "Lola," "Apeman," and "Come Dancing"?
1 - Kinks
2 - Genesis
3 - Huey Lewis and the News
4 - Martha and the Vandellas
5 - Four Lads

Question # 25
In "Days of Wine and Roses," Jack Lemmon was.....
1 - A drunk
2 - In drag
3 - A cop
4 - A tourist
5 - A gourmet

QUIKWIT DATA FILE BACKUP

If you've copied our QUIKWIT questions as data files, or if you've used the QMAKER program to create additional QUIKWIT data files of your own, you'll probably want to create backup copies of those files you've worked so hard on. It's always a good idea to back up all your files on separate diskettes or tapes. You can do this easily for any program files that you have created. To copy sequential data files like the ones used in QUIKWIT, however, you have to use a special utility program such as the one we provide. The program, called SEQCOPY, copies a sequential data file from one diskette or cassette to another. The program is provided on diskette to those of you who've purchased our Mind Moves software package. This program's use is referred to in Appendix A, the Software Backup and Loading Instructions. The program is listed below, and its use by those of you who don't have the dilithium software is explained. Don't forget to save the SEQCOPY program once you've typed it in. The program as listed is for use with a disk drive system. The few changes you'll have to make from the program as listed in order to use it with a cassette system are explained following the Cassette-Based System subheading.

The program is listed in lowercase text mode. Get your computer into this mode before typing in the program. To do this, type in this statement:

```
PRINT CHR$(14) [RETURN]
```

SEQCOPY is easy to use. It begins by asking you the name of the source file. This is the file you wish to have copied. Then it asks the name of the target file. This is the name you wish to call the data file on the new diskette or cassette. Typically the two names are identical.

QUIKWIT has four sequential data files provided with the software, and you need to run SEQCOPY four times to copy

all of them. Their names are witspo, witgen, witsci, and witart, as we explained near the beginning of this appendix. When entering these names for SEQCOPY, keep the titles all in lowercase and don't include any punctuation marks.

After you provide the names, SEQCOPY will tell you to put in place the source diskette or cassette (the one with the original data file) then to hit a keyboard key to signal you're ready. Next, the program tells you it's reading the file. When it's done, it indicates how many data records (or lines) it has read.

Next, SEQCOPY requests that you put the target disk or cassette in the drive then hit any key. The program writes the new copy and indicates when it's finished.

Cassette-Based System

For a cassette system, modify the listed disk version of the SEQCOPY program from this appendix with the following changes:

```
190 print"Hit any key when source tape is in place"
210 print:print"Reading tape"
220 n=0:open 5,1,0,n$
270 print:print"Hit any key when target tape is in
    place"
300 open 2,1,2,t$
```

After you load SEQCOPY into your computer, get the two tapes ready. One tape has the data files you want to back up. This is the source tape. The second is a new tape that will receive the copied data files. This is the target tape.

When you mount the source tape in the cassette drive, be sure to position it somewhere before the actual file. If you're not sure where the file is, you can rewind the tape and the computer will search for the file. When mounting the target tape, make a note of the tape counter. Then you'll know exactly where the file is when you use the backup tape later.

SEQCOPY Program Listing

```
100 rem: seqcopy
110 rem: copyright 1984 dilithium press
120 rem: by phil feldman and tom rugg
130 print chr$(14);chr$(147)
140 close 1:close 2:close 15
150 dim a$(2000)
160 input"name of source file";n$
170 input"name of target file";t$
180 print
190 print"Hit any key when source disk is in place"
200 get r$:if r$="" then 200
210 print:print"Reading disk"
220 n=0:open 15,8,15:open 5,8,5,"0:"+n$+",s,r"
230 n=n+1
240 input#5,a$(n)
250 if st=0 then 230
260 close 5:print n;"records read"
270 print:print"Hit any key when target disk is in place"
280 get r$:if len(r$)=0 then 280
290 print:print"Writing copy":print
300 open 2,8,2,"0:"+t$+",s,w"
310 for q=1 to n
320 print#2,a$(q)
330 next
340 close 2:close15
350 print"Finished"

ready.
```

APPENDIX D
CHECKSUM Program to Check Typing Accuracy

This appendix explains the use of the CHECKSUM program, which should be very useful to you in finding errors in your typing of the programs. It uses a pretty simple technique known as a *checksum* (or hash total or check total, among other names). You need to have a disk on your system to use this program. We won't go into a lengthy explanation of checksums, but the concept is simple. Suppose someone writes down a column of numbers and their sum. (Let's say 10 numbers, each of which is six digits long.) Suppose your job (don't ask us why) is to key these 10 numbers into an adding machine and have the machine come up with the same sum. You do this, being moderately careful to hit all the keys correctly. When the sum comes out, you discover either that it is the same (in which case you are reasonably confident that you hit all the right keys) or that it is different (in which case you are quite confident that you miskeyed one or more numbers).

It's possible, of course, that the same sum could be created in spite of certain types of keying errors. Not too likely, depending on your keying ability, but possible. But it's almost certain that different sums mean your 10 numbers weren't the same as the original 10. We say *almost* because of the chance that the sum of the original 10 numbers was wrong.

In any event, this approach is the same one taken with checksums for the programs in this book. The program that we wrote for this appendix looks through each set of 10 program lines of the program file you have saved on disk,

adding up a checksum using the numerical representation of the BASIC program. It isn't important exactly what the numbers mean. What is important is that you can run this same program to check that the checksum for each of *your* 10 typed lines matches with the checksum *we* found in our original programs. If you find a mismatch, you have focused on a group of 10 program lines to search for typing errors.

HOW TO USE THE CHECKSUM PROGRAM

1. Enter the command NEW to erase any program from the computer's memory, and carefully type in the CHECKSUM program shown in this appendix. Save it on disk.
2. Type NEW again. Now type in one of the programs shown in the six chapters of this book and save it on disk, as explained in the How To Use This Book section at the front of the book.
3. Load the CHECKSUM program from disk into the computer's memory with this command:

   ```
   LOAD"CHECKSUM",8
   ```

4. Now type RUN and press [RETURN] to run the CHECKSUM program. When it asks for the name of the file to check, respond with the name of the program file you saved (e.g., VERTIGO3). A good idea is to start by checking the CHECKSUM program itself, although it may not run properly if you made severe typing errors. If it *does* run properly, you'll know you typed it right.
5. Compare the checksum numbers shown on the screen (for *your* typing efforts) with the checksum numbers for the same program that are shown in this appendix. Note the numbers of any groups of line numbers that have a checksum figure in the appendix different from the one shown on the screen. It will take anywhere from one to

five minutes to display all the checksum numbers for a program, depending on how long it is.

6. If the line number ranges themselves don't match, you have either omitted one or more lines when typing the program, or you have typed extra lines. For example, if the book shows a checksum for lines 100 to 190, but your screen shows the lines 100 to 200, you left out a line somewhere in the 100 to 190 range. The CHECKSUM program simply starts with the first line number it finds (which should be 100 for all the programs in this book) and calculates a checksum number for each group of 10 lines.

7. Once you have noted all the ranges with checksum differences, LOAD the program being checked and use the LIST command to list each of those line ranges on the screen (or your printer). For example, use the command:

   ```
   LIST 100-190
   ```

   Carefully check those 10 lines against the same 10 lines printed in the book, looking for typing errors in your version. Correct the error(s) and go on to the next group of lines with a checksum difference. When done with all of them, SAVE the program on disk with a new name, go back to step 3 and run CHECKSUM again to test your newly saved program file.

8. When the checksums all match, there's a very good chance you have caught all your important typing errors. Certain types of typing differences will not be caught, however.

   First, we intentionally did not include blank spaces in the checksum total. This is because blanks are optional in most parts of programs, and we included them in ours only for easier readability. If you prefer not to type them, fine. Omitting blank spaces or typing extra ones will not change the checksum numbers. However, this

means it's up to you to be sure you type the right number of spaces when it *does* matter — between quotation marks and in DATA statements.

Second, certain typing errors will not be caught by this checksum method. If a line says "A=B" in it, and you type "B=A" instead, it will not be caught because the sums will be the same. Of course, if you type "A=C" or "A+B" or "A/B" or many other things, they *will* be caught.

PROGRAM LISTING

```
100 REM: CHECKSUM
110 REM: COPYRIGHT 1984 DILITHIUM PRESS
120 REM: BY TOM RUGG AND PHIL FELDMAN
130 NL=10:PRINT CHR$(142);CHR$(147);TAB(15);"CHECKSUM"
140 CLOSE 5:CLOSE 15
150 PRINT:PRINT"NAME OF FILE TO CHECK (OR 'QUIT')?"
160 N$="":INPUT N$:IF N$="QUIT" THEN END
170 IF LEN(N$)=0 THEN 150
180 PRINT
190 PRINT"PRESS A KEY WHEN DISK IS IN PLACE"
200 GET A$:IF LEN(A$)=0 THEN 200
210 PRINT:PRINT"READING DISK":PRINT
220 OPEN 15,8,15:OPEN 5,8,5,"0:"+N$+",P,R"
230 INPUT#15,A$,B$,C$,D$
240 IF VAL(A$)=0 THEN 310
250 IF VAL(A$)<>62 THEN 280
260 PRINT"ERROR - NO SUCH FILE NAME ON DISK"
270 PRINT:GOTO 140
280 PRINT"**** DISK ERROR ****":PRINT
290 PRINT"DOS ERROR MESSAGE NO. ";A$
300 PRINT B$:GOTO 490
310 GET#5,A$,B$
320 GET#5,A$,B$
330 IF LEN(A$)=0 AND LEN(B$)=0 THEN 460
340 GET#5,A$,B$:A$=A$+CHR$(0):B$=B$+CHR$(0)
350 CL=ASC(A$)+256*ASC(B$):IF SL=0 THEN SL=CL
360 GET#5,A$:IF LEN(A$)=0 THEN 400
370 A=ASC(A$):IF A=32 THEN 390
380 CS=CS+A
390 GOTO 360
400 LC=LC+1:IF LC>=NL THEN 420
410 GOTO 320
420 IF LS=0 THEN GOSUB 510
430 GOSUB 500:LS=LS+1
440 IF LS>=20 THEN GOSUB 540
```

```
450 CS=0:LC=0:SL=0:GOTO 320
460 IF CS=0 THEN 480
470 GOSUB 500
480 PRINT"GRAND TOTAL =";GT
490 CLOSE 5:CLOSE15:PRINT:END
500 GT=GT+CS:PRINT SL;"TO";CL;TAB(15);CS:RETURN
510 PRINT:PRINT"FILE NAME=";N$:PRINT
520 PRINT"LINE NUMBERS";TAB(15);"CHECKSUM"
530 RETURN
540 PRINT"PRESS A KEY TO GO ON";
550 FOR J=1 TO 10:GET A$:NEXT
560 GET A$:IF LEN(A$)=0 THEN 560
570 LS=0:PRINT:RETURN

READY.
```

CHECKSUM DATA FOR THE PROGRAMS

FILE NAME=AEROJAM

LINE NUMBERS	CHECKSUM
100 TO 190	10781
200 TO 290	21193
300 TO 390	25341
400 TO 490	23256
500 TO 590	16678
600 TO 690	12053
700 TO 790	17172
800 TO 890	21742
900 TO 990	17198
1000 TO 1090	21873
1100 TO 1190	18890
1200 TO 1290	19730
1300 TO 1390	20705
1400 TO 1490	10735
1500 TO 1590	13222
1600 TO 1690	13331
1700 TO 1790	14111
1800 TO 1890	14270
1900 TO 1990	16157
2000 TO 2090	13938
2100 TO 2190	13140
2200 TO 2290	13610
2300 TO 2390	13782
2400 TO 2490	13630
2500 TO 2590	12366
2600 TO 2690	12646
2700 TO 2790	18236
2800 TO 2890	13455
2900 TO 2990	15903
3000 TO 3090	13046
3100 TO 3190	13075
3200 TO 3290	17795
3300 TO 3390	21745
3400 TO 3490	17292
3500 TO 3590	16325
9990 TO 9990	128

GRAND TOTAL = 568550

FILE NAME=ESCAPADE

LINE NUMBERS	CHECKSUM
100 TO 190	11241
200 TO 290	9667
300 TO 390	13622
400 TO 490	11797
500 TO 590	17817
600 TO 690	11497
700 TO 790	12376
800 TO 890	12425
900 TO 990	15645
1000 TO 1090	13723
1100 TO 1190	12757
1200 TO 1290	16129
1300 TO 1390	17209
1400 TO 1490	15486
1500 TO 1590	11801
1600 TO 1690	8870
1700 TO 1790	15064
1800 TO 1890	16696
1900 TO 1990	13867

```
 2000 TO 2090    19453
 2100 TO 2190    21543
 2200 TO 2290    24858
 2300 TO 2390    23139
 2400 TO 2490    26613
 2500 TO 2590    25527
 2600 TO 2690    23759
 2700 TO 2790    15742
 2800 TO 2890    18192
 2900 TO 2990    11664
 3000 TO 3090    11597
 3100 TO 9990     4327
GRAND TOTAL = 484103


FILE NAME=HOTSHOT

LINE NUMBERS    CHECKSUM
 100 TO 190      19242
 200 TO 290      29857
 300 TO 390      20009
 400 TO 490      20997
 500 TO 590      26861
 600 TO 690      27752
 700 TO 790      19516
 800 TO 890      22256
 900 TO 990      18693
 1000 TO 1090    15239
 1100 TO 1190    17024
 1200 TO 1290    19731
 1300 TO 1390    22079
 1400 TO 1490    13834
 1500 TO 1590    16807
 1600 TO 1690    15480
 1700 TO 1790    12147
 1800 TO 1890    15916
 1900 TO 1990    16490
 2000 TO 2090    20029
 2100 TO 2190    17783
 2200 TO 2290    15147
 2300 TO 2390    27324
 2400 TO 2490    24097
 2500 TO 2590    18705
 2600 TO 2690    18228
 2700 TO 2790    16531
 2800 TO 2890    17000
 2900 TO 2990     8936
 3000 TO 3090    14975
 3100 TO 3190    13837
 3200 TO 3290    16096
 3300 TO 3390    15109
 3400 TO 3490    14655
 3500 TO 3590    11999
 9990 TO 9990      128
GRAND TOTAL = 640509


FILE NAME=QUIKWIT

LINE NUMBERS    CHECKSUM
 100 TO 190      13857
 200 TO 290      12105
 300 TO 390      15533
 400 TO 490      14941
 500 TO 590      12463
 600 TO 690      19269
 700 TO 790      12395
 800 TO 890      13931
 900 TO 990      18978
 1000 TO 1090    13843
 1100 TO 1190    13181
 1200 TO 1290    12757
 1300 TO 1390    16948
 1400 TO 1490    14794
 1500 TO 1590    15788
 1600 TO 1690    15787
 1700 TO 1790    17642
 1800 TO 1890    16498
 1900 TO 1990    17039
 2000 TO 2090    20783
 2100 TO 2190    16412
 2200 TO 2290    17294
 2300 TO 2390    14658
 2400 TO 2490    14359
 2500 TO 2590    15067
 2600 TO 2690    14165
 2700 TO 2790    17055
 2800 TO 2890    15476
 2900 TO 2990    15997
 3000 TO 3090    15409
 3100 TO 3190    13970
 3200 TO 3290    12943
 3300 TO 3390    18682
 3400 TO 3490    18612
 3500 TO 3590    17897
 3600 TO 3690    14932
 3700 TO 3790    19016
 3800 TO 3890    22124
 3900 TO 3990    13326
 4000 TO 4090    15148
 4100 TO 4190    14758
 4200 TO 9990     5419
GRAND TOTAL = 651251
```

```
FILE NAME=QUIKWORD

LINE NUMBERS    CHECKSUM
 100 TO 190      14052
 200 TO 290      12633
 300 TO 390      13081
 400 TO 490      12685
 500 TO 590      12229
 600 TO 690      11716
 700 TO 790      14359
 800 TO 890      13004
 900 TO 990      12833
 1000 TO 1090    8772
 1100 TO 1190    16207
 1200 TO 1290    17560
 1300 TO 1390    15913
 1400 TO 1490    11063
 1500 TO 1590    10017
 1600 TO 1690    14800
 1700 TO 1790    10602
 1800 TO 1890    12259
 1900 TO 1990    12561
 2000 TO 2090    10758
 2100 TO 2190    14420
 2200 TO 2290    12873
 2300 TO 2390    7577
 2400 TO 2490    9571
 2500 TO 2590    10660
 2600 TO 2690    12081
 2700 TO 2790    8500
 2800 TO 2890    17287
 2900 TO 2990    18918
 3000 TO 3090    8414
 3100 TO 3190    10828
 3200 TO 3290    11943
 3300 TO 3390    14028
 3400 TO 3490    14813
 3500 TO 3590    10480
 3600 TO 3690    13314
 3700 TO 3790    12480
 3800 TO 3890    10229
 9990 TO 9990    128
GRAND TOTAL = 475648
```

```
FILE NAME=VERTIGO

LINE NUMBERS    CHECKSUM
 100 TO 190      16941
 200 TO 290      14740
 300 TO 390      10983
 400 TO 490      9513
 500 TO 590      13129
 600 TO 690      14836
 700 TO 790      11765
 800 TO 890      15749
 900 TO 990      9805
 1000 TO 1090    7845
 1100 TO 1190    9620
 1200 TO 1290    9267
 1300 TO 1390    10645
 1400 TO 1490    7691
 1500 TO 1590    7079
 1600 TO 1690    16750
 1700 TO 1790    8760
 1800 TO 1890    12862
 1900 TO 1990    15311
 2000 TO 2090    13456
 2100 TO 2190    11102
 2200 TO 2290    13064
 2300 TO 2390    11077
 2400 TO 2490    15337
 2500 TO 2590    23917
 2600 TO 2690    11739
 2700 TO 2790    16073
 2800 TO 9990    3395
GRAND TOTAL = 342451
```

```
FILE NAME=QMAKER

LINE NUMBERS    CHECKSUM
 100 TO 190      14252
 200 TO 290      15675
 300 TO 390      14720
 400 TO 490      12289
 500 TO 590      14920
 600 TO 690      8864
 700 TO 790      13846
 800 TO 890      14681
 900 TO 990      11258
 1000 TO 1090    23330
 1100 TO 1150    6646
GRAND TOTAL = 150481
```

```
FILE NAME=SEQCOPY

LINE NUMBERS    CHECKSUM
 100 TO 190      13251
 200 TO 290      14209
 300 TO 350      4190
GRAND TOTAL = 31650
```

```
FILE NAME=CHECKSUM

LINE NUMBERS    CHECKSUM
 100 TO 190       15719
 200 TO 290       13805
 300 TO 390       12752
 400 TO 490       10288
 500 TO 570       11008
GRAND TOTAL = 63572
```

BIBLIOGRAPHY

Behrendt, Bill L. *Music and Sound for the Commodore 64.* New York, Micro Text Publications, 1983.

Commodore Business Machines. *Commodore 64 Programmer's Reference Guide.* Wayne, Pennsylvania, Commodore Business Machines, 1983.

Commodore Business Machines. *Commodore 64 User's Guide.* Wayne, Pennsylvania, Commodore Business Machines, 1983.

DaCosta, Frank. *Writing BASIC Adventure Programs for the TRS-80.* Blue Ridge Summit, Pennsylvania, Tab Books, 1982.

Daly, Maurice. *Daly's Billiard Book.* New York, Dover Publications, 1971.

Rugg, Tom; Feldman, Phil; Western Systems Group. *More Than 32 BASIC Programs for the Commodore 64 Computer.* Beaverton, Oregon, dilithium Press, 1983.

Willis, Jerry, and Willis, Deborah. *How To Use the Commodore 64.* Beaverton, Oregon, dilithium Press, 1983.

ERRATA OFFER

No matter how thoroughly we test our programs, and no matter how carefully our editors check things out, there is always the possibility of errors creeping into the final product. Our experience has shown that the overwhelming majority of complaints from readers about so-called errors in our programs are not program errors at all. They are typing errors by the reader. So *please*, before you send that irate letter claiming that one of the programs won't work (and making unkind remarks about our ancestries), take the time to read the How To Use This Book section in the front of the book. The probability is *extremely* high that you have made one or more typing errors.

But, as we said, errors in the programs and the text of the book can happen. If you are firmly convinced that you have made no typing errors in a program, but you still can't get it to work, take these steps:

1. Borrow a printer, if necessary, and make a printed listing of the program as you typed it. This will be easier for you to recheck (one more time), and it allows us to verify your typing accuracy.
2. Write down a detailed description of the problem. What's the first thing that goes wrong? What is the *exact* error message, if any? Exactly what did you respond to each option and question, and what was the result you saw on the screen? If possible, we would like to see what the current values are for the main variables

used in the line with the error. If variables J and K are used in the line, enter the command PRINT J,K right after the error occurs to see their current values.

3. Send the listing and the description, along with $1 and a self-addressed stamped envelope, to the address below. We hate to ask for money for this, but it has worked out to be the best compromise that gives readers with problems somewhere to go for help, while reducing the volume of mail that otherwise would come from people who haven't checked their typing. We'll send you a list of errata (corrections to errors), if any, and may be able to help you with your problem even if it is not a result of an error in the book.

Errata — C-64 Mind Moves
Software Support
dilithium Press
8285 S.W. Nimbus, Suite 151
Beaverton, Oregon 97005-6401

MIND MOVES

SIX GAMES OF SKILL AND STRATEGY TO CHALLENGE YOUR MIND!

Looking for strategy games that provide both challenge and entertainment on your Commodore 64?

Here are six intriguing games you can play with a friend, your computer, or all by yourself! A potpourri of intelligent games for the entire family, Mind Moves includes:

- **AEROJAM:** A resource management game. Try to control the arrival and departure of strange spacecraft in an unfamiliar world.
- **ESCAPADE:** An adventure game. Can you explore the mysteries and survive the dangers, and still manage to remove the treasures?
- **HOTSHOT:** A cushion billiards simulation. Line up your shots and try to outscore your opponents.
- **QUICKWIT:** A trivia/quiz game. Answer multiple choice questions faster than your opponent.
- **QUICKWORD:** A fascinating twist to the classic game of anagrams. Unscramble your opponent's word list faster than they can unscramble yours.
- **VERTIGO:** A dizzying board game. The first player to line up the helium balloons wins.

This book is full of clearly explained instructions and examples to help you begin playing your mind moving games right away!

You can either buy the book/software package and have a disk with the programs ready to run on your Commodore 64, or you can type in the programs yourself for disk or cassette.

0 45078 00995 56054

Programs run on a Commodore 64 computer with 64K and a color or black-and-white TV or monitor.

dilithium Press

SOFTWARE