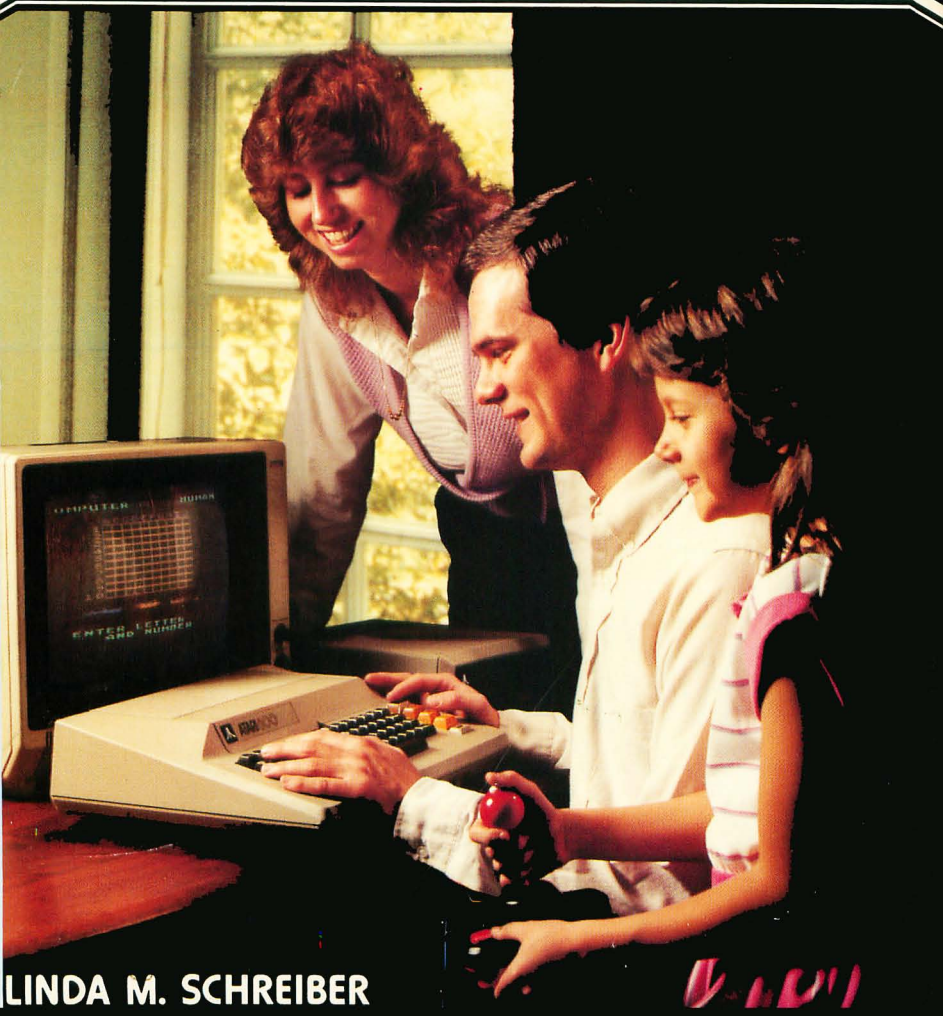
1945

ATARI®

FUN & GAMES

Discover new heights in game-playing excitement on *any* ATARI: 400, 600, 800, and 1200 systems, including XE and XL models!!



LINDA M. SCHREIBER

ATARI®
FUN & GAMES

Discover new heights in game-playing excitement on *any* ATARI: 400, 600, 800, and 1200 systems, including XE and XL models!!

LINDA M. SCHREIBER

TAB TAB BOOKS Inc.
Blue Ridge Summit, PA 17214

To my grandfather, Stephan Rylko, who enjoys a good game and a challenging puzzle.

FIRST EDITION

FIRST PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Schreiber, Linda M.
ATARI fun and games.

1. Computer games. 2. Atari computer—Programming.
I. Title.
GV1469.2.S39 1985 794.8'2 85-22238
ISBN 0-8306-0945-8
ISBN 0-8306-1945-3 (pbk.)

Contents

Introduction

This book was written for individuals, families, and everyone who wants to have fun with the ATARI computer. It is also designed as a hands-on book for the novice as well as the experienced programmer. The assortment of games listed here are both adaptations of traditional board and card games, and some games written specifically for the computer. Most of the games are suitable for children as well as adults.

Chapter 1 contains adaptations of well-known card games. Although most card games are written for two or more players, the computer can be used as a challenging opponent. Children will enjoy these versions of *Fish* and *Old Maid*. *Tarot* and *Twenty-One* were written as party games.

Traditional pencil-and-paper games are easily adapted for the computer; the screen becomes an endless supply of paper and the joysticks or arrow keys are fine electronic pencils. The games in Chapter 2 include some for all skill levels. Some of these games require the player to calculate his moves; others depend more on luck than skill. All the games are based on a type or grid format.

Word games have become very popular in the past few years. An advantage of computer games is that the words used in the program can be changed to meet the player's level. *Jotto* and *Robotman* can be played with youngsters as well as adults. *Fractured Stories* is a favorite for all ages. *Decoder* has two levels of play; use the symbols when the letter encryptions are no longer a challenge.

Logic games challenge the mind. Some of the games in Chapter 4 are based on mathematical formulas; others require concentration in order to win. These games can be played alone or with a friend. The computer serves as the game master, making sure the rules are followed.

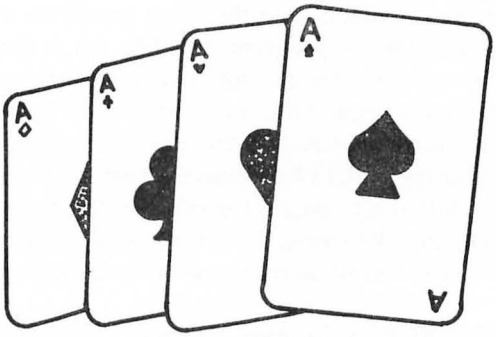
Chapter 5 contains simulations. These games offer many challenges while increasing hand-eye coordination. *Jacks* and *Marbles* are based on children's games. There is a variation of *Darts*, and *Ski* is for those who want to enjoy that sport safely! The version of *Pinball* is designed for young children who do not have the skills for arcade games but want to play anyway.

The techniques used in the programs are described with the listings. Many of the programs use the ATARI's unique features of player/missile graphics, movable character sets, and interrupts. By understanding these features better, you will be able to incorporate them into your own programs. Figures depict the new characters and how they are used in the program. All programs use the ATARI's color and sound capabilities.

The programs are written on an intermediate level, but, although the programs are thoroughly explained, the explanations are written with the assumption that you know the rudiments of programming.

These programs are designed to work on all ATARI computers with 16K of memory. Some of the programs require one joystick; a few require two. The requirements are listed within each program description.

Chapter 1



Card Games

This chapter contains several favorite games to play against the computer. *Go Fish* and *Old Maid* are one-player games kids will enjoy; *Twenty-One* or *Blackjack* can be played by one or two players. The computer will attempt to tell your future in the *Tarot* program.

GO FISH

Objective of the game: To collect more pairs of cards than the computer.

Directions: The computer deals five cards to you and five to itself. You are asked if you want to go first. Enter **Y** if do, otherwise, enter **N**. You are asked to enter your pair. If you do not have a pair, press the Return key. A pair of cards is two cards of the same value and same color, such as a three of clubs plus a three of spades, or a king of hearts plus a king of diamonds.

Next you are asked to enter the card you want. This should be a card that would make a pair with one of the cards in your hand. If you have a seven of spades, ask for a seven of clubs. To enter the card, press the number value (1-10) or face value (J, Q, K, A) of the card, then press the first letter of the suit (C, H, S, D). Press the Return key to record the entry.

If the computer has the card in its hand, it will say so and place the card in your hand. If it does not have the card, you will fish. The first card from the deck will be placed in your hand. If you now have a pair of cards, you can enter them. If you do not have

a pair, just press the Return key and the computer will take its turn. The game continues until all the cards have been played. The player with the most pairs is the winner.

The computer checks to see if you have both cards of any pair you enter. It also checks that the cards constitute a pair. Listing 1-1 is the code for the program, and Fig. 1-1 is the flowchart.

Line 50 sets aside the memory needed for strings and arrays. The C array will contain all 52 cards. These cards will be shuffled and used in the game. CARD$ contains the number or letter that appears on the card. SUIT$ contains the four suit characters. PLAY array and CMP array contain the cards in the hand of the player and the computer. B$ stores the entries —, Y, N, or the cards. The CP array stores the numeric value of the cards entered by the player.

Line 60 PEEKs at location 106 to see how much memory is in the computer you are using. The computer moves the character set into RAM, so 2K must be subtracted from the amount of RAM available. This value is POKEd into location 204, and the first byte of the character set in ROM is stored in location 206. The actual decimal location of the ROM character set is stored in the variable CS. This location will be used to move the four graphics characters that represent the suits into RAM.

Line 70 moves the assembly language subroutine into locations 1536-1555; line 80 contains the decimal code for the routine.

Line 90 changes the screen to graphics 17. This is graphics 1 with no text window. The computer uses the USR command to execute the assembly language subroutine at memory location 1536. When the computer returns to this line, it will place the value of A into location 756. Now the computer is using the character set in RAM. The first byte of this character set is stored in the variable CHARSET.

Line 100 uses two FOR-NEXT loops to move the four characters that represent the suits into RAM. When we use the large text, the computer can only use numbers and uppercase letters, or graphics and lowercase letters. We need to move the four graphics characters into locations in the character set in RAM in order to use them. The computer will replace the percent sign, the ampersand sign, the apostrophe, and the open parenthesis with the graphics characters for the four suits. The computer reads the number from line 110. This is the location of the graphics character in the character set.

The computer uses the second FOR-NEXT loop to move each

Listing 1-1. Go Fish.

```
10 REM GO FISH FOR ONE PLAYER AND THE
COMPUTER
20 REM CHAPTER 1 - CARD GAMES
30 REM BY L.M.SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1984
50 DIM C(52),CARD$(13),SUIT$(4),PLAY(3
0),CMP(30),B$(4)
60 A=PEEK(106)-16:POKE 204,A:CS=PEEK(7
56):POKE 206,CS:CS=CS*256
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256:REM SET UP THE CHARACTER
S
100 FOR CSET=CHARSET+40 TO CHARSET+64
STEP 8:READ V:FOR X=0 TO 7:POKE CSET+X
,PEEK(CS+V*8+X):NEXT X:NEXT CSET
110 DATA 64,80,96,123
115 REM THE CHARACTERS FOR CARD$ AND S
UIT$ ARE INVERSE
120 CARD$="A234567891JQK":SUIT$="%&'("
:REM PUT THE CARDS IN THE STRINGS
130 POSITION 1,12:? #6;"% & shuffling
' (":GOSUB 1050:HS=0:CS=0
140 FOR X=1 TO 10:IF X/2=INT(X/2) THEN
 PLAY(X/2)=C(X):GOTO 160
150 CMP(INT(X/2)+1)=C(X)
160 NEXT X:CD=10:LC=5:CC=5:AF=1:GOSUB
2000
170 CO=3:R=14:C=13:POSITION 0,14:? #6;
"GO FIRST Y/N":GOSUB 1100:IF B$="" THE
N 170
180 IF B$="N" THEN 450
190 GOSUB 300
200 GOSUB 2000:POSITION 0,14:? #6;"WHA
T CARD ":C=12:R=14:CO=1:GOSUB 1100:IF
B$="" THEN 200
210 FOR X=1 TO 13:IF ASC(CARD$(X,X))-1
28=ASC(B$(1,1)) THEN L=LEN(B$):NC=X+(A
SC(B$(L,L))-36)/10:GOTO 230
```

```
220 NEXT X
230 FOR X=1 TO CC:IF CMP(X)<>NC THEN N
EXT X:GOTO 270
240 POSITION 0,16:? #6;"i have it":GOS
UB 3000:CMP(X)=CMP(CC):CC=CC-1:IF CC=0
 THEN GOSUB 600:IF CC=0 THEN 800
250 LC=LC+1:PLAY(LC)=NC:GOSUB 2000
260 GOSUB 300:GOTO 450
270 GOSUB 400:GOSUB 2000:GOTO 260
300 POSITION 0,14:? #6;"YOUR PAIR?
    ";:R=14:C=11:CO=1:GOSUB 1100:IF B$
="" THEN RETURN
310 FOR X=1 TO 13:IF ASC(CARD$(X,X))-1
28=ASC(B$(1,1)) THEN L=LEN(B$):CP(1)=X
+(ASC(B$(L,L))-36)/10
315 NEXT X
320 POSITION 6,15:? #6;"AND?        ":R
=15:C=11:CO=1:GOSUB 1100:IF B$="" THEN
 320
330 FOR X=1 TO 13:IF ASC(CARD$(X,X))-1
28=ASC(B$(1,1)) THEN L=LEN(B$):CP(2)=X
+(ASC(B$(L,L))-36)/10
335 NEXT X
340 IF INT(CP(1))<>INT(CP(2)) OR ABS(C
P(1)-CP(2))<>0.2 THEN 390
350 FOR Q=1 TO LC:IF PLAY(Q)<>CP(1) TH
EN NEXT Q:GOTO 390
360 FOR V=1 TO LC:IF PLAY(V)<>CP(2) TH
EN NEXT V:GOTO 390
370 POSITION 0,17:? #6;"GOOD PAIR":GOS
UB 3000:IF V>Q THEN PLAY(V)=PLAY(LC):L
C=LC-1:PLAY(Q)=PLAY(LC):GOTO 380
375 PLAY(Q)=PLAY(LC):LC=LC-1:PLAY(V)=P
LAY(LC)
380 LC=LC-1:HS=HS+2:IF LC=0 AND CD=52
THEN 800
383 IF LC=0 THEN GOSUB 400
385 RETURN
390 POSITION 0,17:? #6;"NOT A PAIR":GO
SUB 3300:GOSUB 3200:POSITION 0,17:? #6
;"           ":GOTO 300
400 IF CD=52 THEN POSITION 0,17:? #6;"
NO MORE CARDS":GOSUB 3200:RETURN
410 POSITION 0,17:? #6;"FISHING  ":CD=
CD+1:LC=LC+1:PLAY(LC)=C(CD):POSITION 0
,18:? #6;"YOU FISHED A ";
```

```
420 CARD=C(CD):X=6:Y=18:Z=1:GOSUB 1000
:GOSUB 3200:RETURN
450 GOSUB 2000:IF AF>CC THEN AF=1
460 POSITION 0,14:? #6;"i want a  ";:N
C=CMP(AF)-INT(CMP(AF)):IF NC*10/2=INT(
NC*10/2) THEN NC=0.6-NC:GOTO 480
470 NC=0.4-NC
480 CARD=INT(CMP(AF))+NC:X=5:Y=14:GOSU
B 1000:AF=AF+1
490 POSITION 0,15:? #6;"Fish or Have i
t";:CO=4:C=15:R=15:GOSUB 1100:IF B$=""
 THEN 490
500 IF B$="F" THEN GOSUB 600:GOTO 550
510 FOR X=1 TO LC:IF PLAY(X)<>CARD THE
N NEXT X:GOSUB 590:GOTO 550
520 CC=CC+1:CMP(CC)=PLAY(X)
530 PLAY(X)=PLAY(LC):LC=LC-1:IF LC=0 T
HEN IF CD<52 THEN GOSUB 400:GOSUB 2000
:REM DRAW A CARD FROM THE DECK
540 IF LC=0 THEN 800
550 FOR Q=1 TO CC-1:FOR V=Q+1 TO CC:IF
 INT(CMP(Q))=INT(CMP(V)) AND ABS(CMP(Q
)-CMP(V))=0.2 THEN 560
555 NEXT V:NEXT Q:GOTO 580
560 POSITION 0,15:? #6;"
   ":POSITION 0,13:? #6;"i have a pair
 ";:CARD=CMP(Q):X=7:Y=13:Z=1
570 GOSUB 1000:POSITION 0,14:? #6;"
       and ";:Y=14:CARD=CMP(V):GOSUB 1
000:GOSUB 3000:CS=CS+2
575 CMP(V)=CMP(CC):CC=CC-1:CMP(Q)=CMP(
CC):CC=CC-1:IF CC<>0 THEN 200
577 GOSUB 600:IF CC<>0 THEN 200
578 GOTO 800
580 POSITION 0,13:? #6;"no pair     ":G
OSUB 3300:GOSUB 3200:GOTO 200
590 POSITION 0,13:? #6;"i must fish"
600 IF CD=52 THEN POSITION 0,13:? #6;"
NO CARDS LEFT":GOSUB 3200:RETURN
610 CC=CC+1:CD=CD+1:CMP(CC)=C(CD):RETU
RN :REM GET A CARD
790 REM HUMAN IS INVERSE
800 POSITION 0,0:? #6;CHR$(125):POSITI
ON 5,15:? #6;"WINNER IS --":IF HS>CS T
HEN POSITION 6,17:? #6;"HUMAN"
810 IF CS>HS THEN POSITION 4,17:? #6;"
computer"
```

```
815 REM tie IS LOWER CASE INVERSE
820 IF CS=HS THEN POSITION 8,17:? #6;"
tie":REM lower case inverse
830 IF PEEK(53279)<>6 THEN 830
840 RUN
1000 CV=INT(CARD):S=(CARD-CV)*10:IF CV
=10 AND X-Z>1 THEN POSITION (X-Z)*3-1,
Y
1005 IF S=1 OR S=3 THEN 1020
1010 ? #6;CARD$(CV,CV);:IF CV=10 THEN
? #6;"0";:REM ZERO IS INVERSE
1015 ? #6;SUIT$(S,S):RETURN
1020 A=ASC(CARD$(CV,CV)):IF A<190 THEN
 ? #6;CHR$(A-32);:IF CV=10 THEN ? #6;C
HR$(144);
1025 IF A>190 THEN ? #6;CHR$(A+32);
1030 A=ASC(SUIT$(S,S)):? #6;CHR$(A-32)
:RETURN
1050 Q=0:FOR V=0.1 TO 0.4 STEP 0.1:FOR
 X=1 TO 13:C(X+Q)=X+V:NEXT X:Q=Q+13:NE
XT V
1060 FOR X=1 TO 5:FOR Q=52 TO 1 STEP -
1:V=INT(RND(1)*Q):REM PICK A CARD
1070 C(0)=C(V):C(V)=C(Q):C(Q)=C(0):REM
 MOVE THE CARDS
1080 NEXT Q:NEXT X:RETURN :REM DO IT 5
 TIMES
1100 B=1:B$="":OPEN #2,4,0,"K:":GOSUB
3400
1110 GET #2,K:IF K=155 THEN CLOSE #2:R
ETURN
1120 IF K>127 THEN K=K-128:POKE 694,0:
REM RESET INVERSE FLAG
1130 IF K=126 AND CO<3 THEN B$="    ":
B$="":B=1:POSITION C+B,R:? #6;"    ":C
O=1:GOTO 1110
1140 IF K>95 THEN K=K-96:POKE 702,64:R
EM SET FOR UPPERCASE
1150 IF CO=3 THEN IF K=78 OR K=89 THEN
 1200
1160 IF CO=1 THEN IF (K>47 AND K<58) O
R K=65 OR K=74 OR K=75 OR K=81 THEN 12
00
1170 IF CO=2 AND B<4 THEN IF K=67 OR K
=68 OR K=72 OR K=83 THEN 1230
1180 IF CO=4 THEN IF K=70 OR K=72 THEN
 1200
```

```
1190 GOTO 1110
1200 POSITION C+B,R:? #6;CHR$(K):B$(B,
B)=CHR$(K):IF CO>2 THEN CLOSE #2:RETUR
N
1210 B=B+1:IF CO=1 THEN CO=2:IF K=49 T
HEN CO=1
1220 GOTO 1110
1230 IF K=67 THEN K=38
1240 IF K=68 THEN K=39
1250 IF K=72 THEN K=37
1260 IF K=83 THEN K=40
1270 GOTO 1200
2000 POSITION 0,0:? #6;CHR$(125):Y=1:Z
=1:FOR X=1 TO LC:CARD=PLAY(X):POSITION
 (X-Z)*3,Y
2010 GOSUB 1000:IF X/7=INT(X/7) THEN Y
=Y+2:Z=Z+7
2020 NEXT X
2030 POSITION 0,21:? #6;"HUMAN":POSITI
ON 3,22:? #6;HS
2040 POSITION 11,21:? #6;"computer":PO
SITION 15,22:? #6;CS:RETURN
3000 FOR ZZ=1 TO 10:T=INT(RND(1)*150)+
50:SOUND 0,T,10,10:FOR TI=1 TO 10:NEXT
 TI:NEXT ZZ:SOUND 0,0,0,0
3200 FOR TI=1 TO 500:NEXT TI:RETURN
3300 FOR ZZ=1 TO 2:SOUND 0,200,10,10:F
OR TI=1 TO 10:NEXT TI:SOUND 0,0,0,0:FO
R TI=1 TO 5:NEXT TI:NEXT ZZ:RETURN
3400 FOR ZZ=1 TO 2:SOUND 0,20,10,10:FO
R TI=1 TO 10:NEXT TI:SOUND 0,0,0,0:FOR
 TI=1 TO 5:NEXT TI:NEXT ZZ:RETURN
```

of the eight bytes that make up a character from ROM and place it into RAM. The computer finds each byte of the character by multiplying the value of V by eight and adding the value of X and CS. The value of X will increase from zero to seven in the loop. The other variables will remain the same. The value found is placed into the RAM location. Because the value of X increases, each of the eight bytes will be moved in the correct order and stored in the correct location of RAM. The loop continues until all four characters have been moved.

Line 110 contains the ATASCII value of each of the four characters. This is the character's position in the character set.

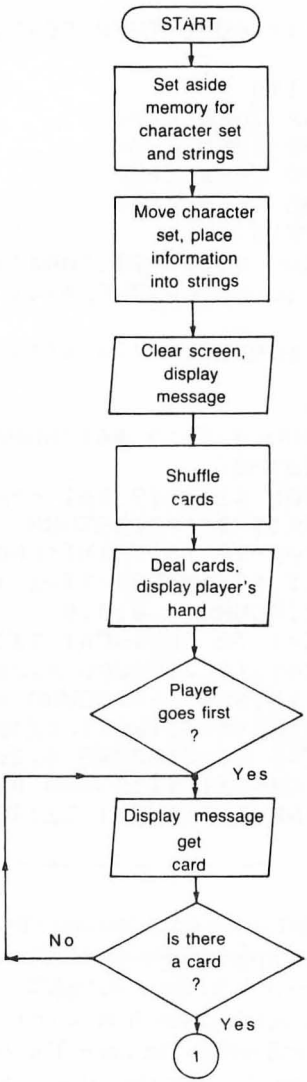
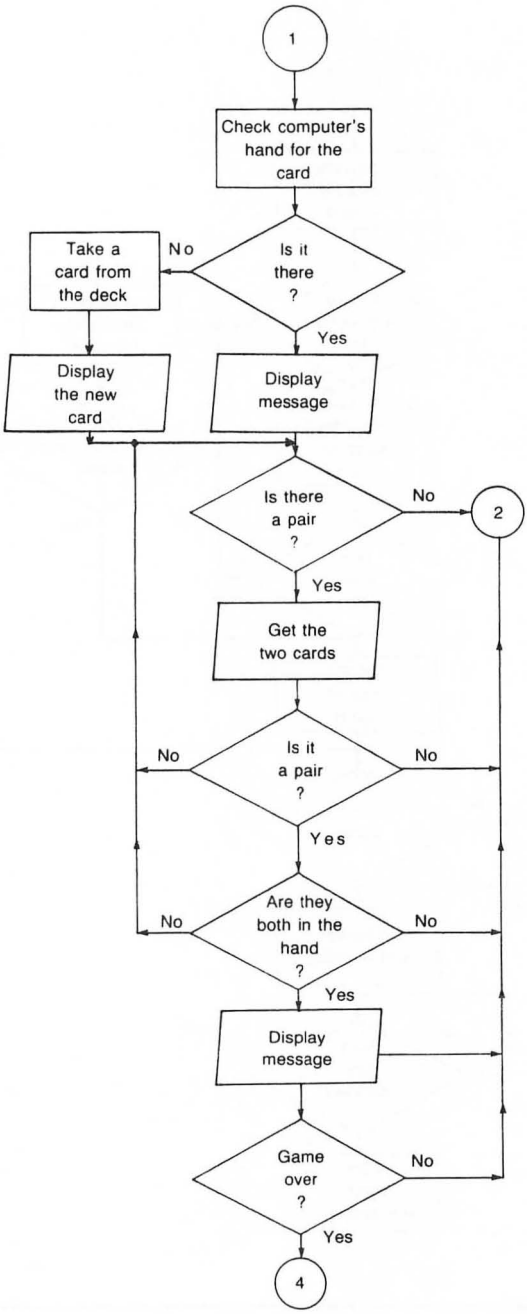


Fig. 1-1. Flowchart for Go Fish. (Continued through page 10.)


1
Check computer's hand for the card
Is it there ?
No
Take a card from the deck
Yes
Display the new card
Display message
Is there a pair ?
No
2
Yes
Get the two cards
Is it a pair ?
No
No
Yes
Are they both in the hand ?
No
No
Yes
Display message
Game over ?
No
Yes
4

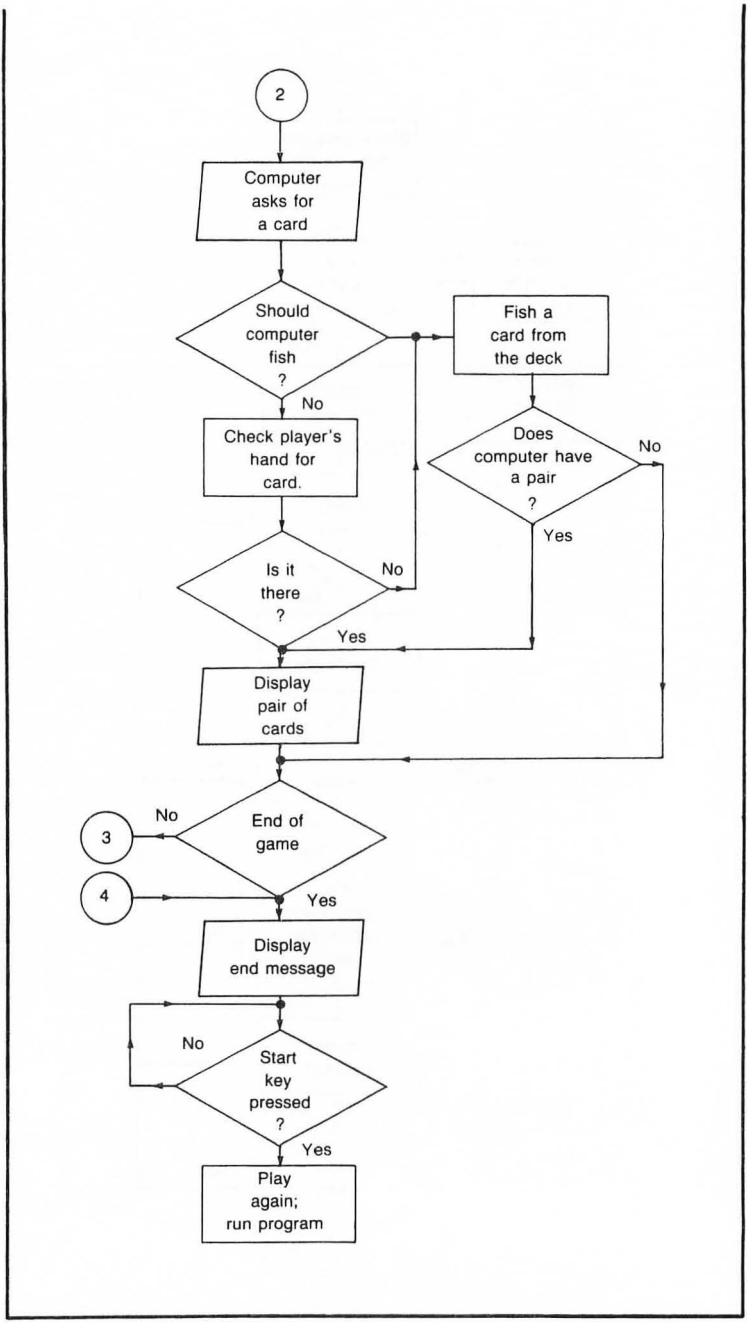


Fig. 1-1. Continued.

Line 120 places the numeric values or letter descriptions of the cards in CARD$. The four characters that have been changed to represent the four suits are placed in SUIT$.

Line 130 prints a message on the screen; be sure to include the four characters. The computer goes to the subroutine at line 1050 to shuffle the cards. The variables HS and CS are set to zero, the score for both players.

Lines 140-160 deal the cards. The first card is dealt to the computer, the second to the player, and so on. The computer divides the variable X by two to find out who gets the card. If the variable is even, the player gets the card; if it is odd, the computer gets it. After 10 cards have been dealt, the variable CD is set to 10. this variable keeps track of how many cards have been dealt. The variables LC and CC are set to 5, the number of cards in each hand. The variable AF is set to 1, and the computer goes to the subroutine in line 2000. This subroutine clears the screen and places the player's cards and the scores on the screen.

Line 170 sets the variable CO to 3. This is a code for the subroutine at line 1100. By using a code variable, the computer will know what letters or numbers are acceptable for the input routine. The variable R is set to 14 and C is set to 13. This is the row and column where the entry will be printed. A question is printed on the screen and the computer uses the subroutine at line 1100 to receive the input. If there is no input, B$ will be empty and the line will repeat.

Line 180 sends the computer to line 450 when the entry is N. The computer goes first.

Line 190 uses the subroutine at line 300 to find out if the player has a pair. When you ask to go first, you will be given an opportunity to play a pair of cards before asking for a card. This occurs only on the first turn. If the computer goes first, it also may play a pair of cards before asking for one.

Line 200 begins the player's turn. The computer uses the subroutine in line 2000 to place the player's cards on the screen and asks for a card to be entered (the card you want from the computer to complete a pair). The row and column variables are set and the code variable is set to 1. The computer uses the subroutine in line 1100 to receive an entry. If no card is entered, the line repeats until a card is asked for.

Lines 210-220 look at the values of the cards in CARD$ and compare them to the value of the card entered. When a match is found, the computer determines the suit of the card the player wants

by subtracting 36 from the ATASCII value of the character. This value is divided by 10 because the suit is the decimal value of the card. The value of the card is stored in the variable NC. The computer is then sent to line 230.

Line 230 looks at all the cards in the computer's hand to see if it can find one that matches the card the player requested. If no match is found, the computer goes to line 270 to fish.

Line 240 tells the player that the computer has the card. The computer uses the subroutine in line 3000 to make a sound, then removes the card from the computer's hand by placing the last card in the computer's hand into this card's position. The variable that keeps track of number of cards in the computer's hand is decreased by one. When this value reaches zero, the computer takes another card from the deck. If there are no more cards, the computer goes to line 800 to end the game.

Line 250 adds 1 to the value of LC. This is the number of cards in the player's hand. The new card is placed into the last position of the player's hand. The computer uses the subroutine at line 2000 to display the entire hand with the new card on the screen.

Line 260 sends the computer to the subroutine at line 300 to get a pair of cards. After the computer returns, the computer is sent to line 450 for its turn.

Line 270 is used when the computer does not have the card the player requested. The computer uses the subroutine at line 400 to take the next card from the deck, and the subroutine from line 2000 to display the new hand on the screen. The computer then goes to line 260 to find out if the player now has a pair of cards.

Lines 300-390 contain the subroutine that fetches the pair of cards from the player. The computer asks for the first card of the pair. The values for the row and column are set, as well as the code for the type of input. The computer uses the subroutine in line 1100 to get the entry. If no card is entered, the computer is sent back to the line it came from; the player has no pair.

Lines 310-315 check the card entered against the card values to convert the entry into a one-digit card value.

Line 320 asks for the second card of the pair. This time the line will repeat if no card is entered, and lines 330-335 again convert the card entered into a one-digit card value.

Line 340 checks the two cards against each other. The number or value of both cards must be the same, and the difference between the two cards must be two-tenths. If either of these conditions are not met, the computer goes to line 390 and tells the player

that this is not a pair.

Lines 350-360 check both cards against every card in the player's hand. The computer knows that a pair was entered; now it must verify that the player has both cards. If either card is not in the hand, the computer goes to line 390 and tells the player this is not a good pair.

Lines 370-385 tell the player the pair was acceptable and proceeds to remove both cards from the player's hand. The last card in the hand must be removed before the first in order for the removal to be accomplished correctly. The number of cards in the player's hand are decreased by one each time a card is removed. The score variable is increased by two, and the computer checks to see if there are cards in the player's hand. If there are none and all 52 cards have been dealt, the computer goes to line 800 to end the game. If cards are left in the deck but the player has no cards, the computer goes to the subroutine in line 400 to get another card. The computer then returns to the main program.

Line 390 tells the player that the two cards entered cannot be used as a pair because they are not a pair, or one or both cards are not in the player's hand. The computer uses the subroutine at line 3300 to make a sound and erases the first card entered. The computer goes back to line 300 for another entry.

Line 400 begins the finishing subroutine. This routine is used when the player runs out of cards. The variable CD is checked; if it is 52, all the cards have been dealt. The computer displays a message, makes a sound, and returns to the main program.

Line 410 tells the player a card is being taken from the deck. The variable CD is increased by one, as is the variable that counts the number of cards in the player's hand; the next card is moved to the player's hand. The computer tells the player what card was placed in his hand.

Line 420 places the decimal value of the card in the variable CARD, and uses the subroutine at line 1000 to print the card on the screen. The computer returns to the main program.

Line 450 begins the computer's turn. The screen is updated and the variable AF is compared to the number of cards in the computer's hand. If the variable is greater than the number of cards, the variable is reset to one. This variable points to the card the computer will try to pair.

Line 460 asks the player for a card. The suit value of the card the computer wants to pair is placed in the variable NC. If the decimal value is even, the computer subtracts this value from six-

tenths. If it is odd, the value is subtracted from four- tenths. This gives the decimal value of the opposite suit.

Line 480 takes the card value and adds it to the decimal value and places it into CARD, so the computer will recognize what card it needs. The subroutine at line 1000 displays this card on the screen. The variable AF is increased by one. On the next turn, the computer will ask for the next card in its hand.

Line 490 asks the player to enter **F** or **H**. F tells the computer to fish, or take the next card from the deck. H means the card is in the player's hand. The variables for the row, column, and code are set. The computer uses the subroutine at line 1100 to get the entry. If no letter was entered, the computer loops at this line.

Line 500 checks the letter in B$. If it is F, the computer goes to the subroutine at line 600 to get a card. The computer then goes to line 550 to see if it has a pair.

Line 510 checks every card in the player's hand for the card the computer requested. The computer uses this line when H is entered. If the card is not in the player's hand, the computer will tell the player it must fish and use the subroutine at line 600 to fish a card from the deck.

Line 520 increases the variable that counts the number of cards in the computer's hand and places the card into the last position of the computer's hand.

Line 530 removes the card from the player's hand by taking the card in the last position and placing it into the position of the card taken by the computer. The variable that counts the number of cards in the player's hand is decreased by one. If the player has no cards and there are cards in the deck, the computer will place the next card into the player's hand and go to line 550 to continue the game.

Line 540 checks the value of LC again. If it is still zero, the computer is sent to line 800 to end the game.

Lines 550-559 take every card in the computer's hand and compare them with every following card to see if the computer has a pair. If there is no pair, the computer goes to line 580 to continue the game.

Line 560 announces to the player that the computer has a pair. The first card value is placed into CARD.

Line 570 uses the subroutine at line 1000 to print the card on the screen. The second card value is placed into CARD and the same routine prints the second card on the screen. The subroutine

at line 3000 makes a sound and the computer's score is increased by two.

Line 575 removes both cards from the computer's hand. If there are cards in the computer's hand, the game continues at line 200 and the player takes a turn.

Line 577 uses the subroutine at line 600 to take the next card. If it takes a card, the computer goes to line 200.

Line 578 sends the computer to line 800 to end the game.

Line 580 tells the player that the computer does not have a pair in its hand. The computer makes a sound, then goes to line 200 for the player's next turn.

Line 590 tells the player that the computer must fish. This line is used when the player responded to the computer's request for a card positively, but the card wasn't in the player's hand.

Line 600 is the fishing routine. If the variable CD is equal to 52, there are no cards left, and the computer returns to the main program.

Line 610 increases the variable CC by one. This variable keeps track of how many cards are in the computer's hand. The variable CD is also increased by one. This is the number of cards that have been removed from the deck. The next card is moved from the deck to the computer's hand and the computer returns to the main program.

Line 800 ends the program. The screen clears and the winning player is named. If the value of HS is greater than CS, the player won and this message is printed on the screen. Line 810 prints that the computer won if its score is higher.

Line 820 states that the game was a tie if both scores are the same.

Line 830 loops until the Start key is pressed.

Line 840 runs the program again. To quit the game, press the System Reset key.

Lines 1000-1030 print the card on the screen. The value of the card is stored as a single number. The suit of the card is a decimal value. The value of the card must be stored in the variable CARD before the computer uses this subroutine. The integer value of the card is stored in the variable CV; the decimal value is stored in the variable S. This value identifies the suit of the card. The variables X and Y are the column and row positions where the card will be printed. The Z value is subtracted from X to move the card one position to the left as long as the variable X is not set to one. To

keep the cards evenly spaced on the screen, the value of X or X – Z is multiplied by three.

Line 1005 sends the computer to line 1020 if the value of S is one or three. The spades are printed in dark blue, the hearts in red.

Line 1010 prints the value of the card on the screen. If it is 10, the zero is also printed.

Line 1015 prints the character for the suit on the screen and returns to the main program.

Line 1020 takes the ASCII value of the card that should be printed. When the card is printed, 32 is subtracted from it so it can be printed in red. If its value is 10 the zero is printed.

Line 1025 adds 32 to the value of A if it is greater than 190. These are the face cards or letter values.

Line 1030 prints the suit character on the screen. This value also has 32 subtracted from it.

Line 1050 places the cards into the C array. Each card is given a value from one to 13 and a suit value from 0.1 to 0.4. The FOR-NEXT loops place these values into the C array. The first loop steps from 0.1 to 0.4. The loop steps by 0.1. The second loop counts from 1-13. The variable Q keeps track of where in the array the cards are to be placed. The values of X and V are added together to get the card value; this card is placed in the array at the location of X plus Q. The first time this loop is executed, the value of Q is zero. Each time the loop is completed, 13 is added to the value of Q. This way, all 52 elements of the array contain a card when the loop is finished.

Lines 1060-1080 shuffle the cards. The cards are shuffled five times. Each time, the computer takes the card from the bottom of the deck and places it into a random position in the deck. The card that was replaced is moved into a temporary storage element. The last card is moved into its position, then this card is moved into the last position. The FOR-NEXT loop counts from 52 to one. The variable Q always points to the last position, so once a card is moved into the last position, the position just before it becomes the last position. This way all the cards in the deck are moved at least once. The loops continue until the deck has been shuffled five times.

Lines 1100-1270 contain the input routine. There are actually three routines within these lines. The position of the letter in the string is pointed to by the variable B. It is set to one for the first letter. B$ is set to an empty string. The keyboard is opened, and the computer uses the subroutine in line 3400 to make a sound. The value of the key that has been pressed is stored in the variable

K. If its value is 155 the Return key has been pressed and the computer returns to the main program.

Line 1120 checks to see if the value of K is greater than 127. If it is, the inverse key has been pressed, the variable K is decreased by 128, and location 694 is cleared for normal video.

Line 1130 checks to see if the delete key has been pressed. If it has, the entire contents of B$ is erased, the variable B is reset to one, and the entry on the screen is erased. The code is reset to one and the computer goes to line 1110 for another input.

Line 1140 checks the value of K to see if it is greater than 95. If it is, the caps key has been pressed and this letter is in lowercase. To reset the letter to uppercase, 96 is subtracted from the value of K and the location 702 is POKEd with 64.

Line 1150 checks the value of CO. This is the code that tells the computer what kind of entry it should look for. If the code is three, the computer should accept only the letters Y and N. If either of these keys were pressed, the computer goes to line 1200.

Line 1160 is used when the CO is 1. The computer will accept any number of the letters A, K, J, or Q. This is the first entry for a card. If any of these keys have been pressed and the computer wants a card value, the computer will go to line 1200.

Line 1170 is used when the value of CO is 2 and there are less than four characters in the string. One of the suit keys (H, C, D, S) must be pressed before the computer an go to line 1230.

Line 1180 will accept the F or H keys if the value of CO is 4.

Line 1190 sends the computer to line 1110. The correct key for the code was not entered.

Line 1200 prints the key that has been pressed on the screen. The character is placed into B$. If the value of CO is greater than two, the computer was looking for a one-keystroke entry. This is used for the yes/no questions and the fish question. The keyboard is closed and the computer returns to the main program.

Line 1210 increases the value of B by one so it can point to the next position in the string. If the value of CO is one, it will be changed to 2. Now the computer will look for the suit key. However, if the value of K is a 1, then the next key should be a 0 and the value of CO is reset to 1.

Line 1220 sends the computer back to line 1110 to get another entry.

Lines 1230-1260 change the value of the key that has been pressed. The first letter of the suit is used for the entry of a particular suit, but these characters have not been changed in the

character set. In order to print the correct suit character on the screen, the value of K must be changed to the correct suit character.

Line 1270 sends the computer to line 1200 to print the suit character.

Lines 2000-2040 update the screen. The screen is cleared, then each card in the player's hand is placed into the variable CARD. The computer calculates the position of the next card on the screen and uses the subroutine in line 1000 to print the card on the screen. Every time seven cards are printed on the screen, the variable Y is increased by 2 to start a new row. The variable Z is increased by 7 for the new offset. The loop continues until all the player's cards are printed on the screen. The scores for the player and the computer are printed near the bottom of the screen. Then the computer returns to the main program. Lines 3000-3400 are various sound routines and timing routines that are used throughout the program.

OLD MAID

Objective of the game: To get rid of all the cards and not be left holding the last card.

Directions: There are 49 cards in the deck; all the cards are dealt. You have 24, and the computer has 25. You are asked if you would like to go first. If you do, enter **Y**; otherwise enter **N**. You are asked if you have a pair. If you do, enter the two cards. A pair is two cards of the same value and the same color, either a heart and a diamond or a club and a spade. If you don't have a pair, press the Return key. You will be told how many cards the computer has. Pick one of these cards and it will be placed in your hand. The game alternates between you and computer until someone runs out of cards. The first player to run out of cards wins. Figure 1-2 is the flowchart for this and Listing 1-2 contains the code.

Line 50 sets aside memory for strings and arrays. The C array contains the cards. CARD$ contains the value of the cards and SUIT$ the characters that represent the four suits. The arrays PLAY and CMP will contain the cards in the player's and computer's hands. B$ will contain the information entered and CP is the temporary storage area for the entered cards.

Line 60 finds out how much memory is in the computer by PEEKing at location 106. The character set will be moved to the area just before the screen, 2K from the end of memory. This value and the beginning byte of the ROM-based character set are POKEd into locations 204 and 206. The decimal address for the first byte

of the ROM-based character set is stored in the variable CS.

Line 70 reads the code for the assembly language subroutine and places it in memory locations 1536-1555. This assembly language subroutine will move the character set located in ROM into RAM.

Line 80 contains the decimal values of the assembly language subroutine that moves the character set.

Line 90 changes the screen to graphics 17, the large colored letters with no text window. The computer uses the USR command to execute the assembly language subroutine that begins in memory location 1536. When the computer returns to this line, it will change the character set to the one in RAM by POKEing location 756 with the value of A. The first byte of this character set is stored in the variable CHARSET.

Line 100 moves the graphic characters that depict the four suits from ROM into RAM. Since the computer can only display numbers and uppercase letters, or graphic characters and lowercase letters, it is necessary to move the suit characters into the locations normally occupied by the percent sign, the ampersand, the apostrophe, and the open parenthesis. To move these characters, the computer reads the location of the character in the character set. To figure out where the first byte is located, the computer multiplies this number by 8, then adds the first byte of the character set and the value of X. X will increment through this loop so that all eight bytes of the character can be transferred. This loop continues until all four characters are moved.

Line 110 contains the position of the four characters in the character set.

Line 120 places the values of the cards in CARD$ and the four suits in SUIT$.

Line 130 places a message on the screen so you will know the computer is working and not caught in an endless loop. The computer uses the subroutine at line 1050 to place the cards in the C array and to shuffle them. When the computer returns to this line, the variable CD is set to one. This is the variable that points to the next card to be dealt. The screen is also cleared.

Lines 140-160 deal the cards into two hands: the player's and the computer's . The number of cards the player and the computer were dealt are placed into variables LC and CC. The player is dealt 24 cards and the computer 25. The subroutine at line 2000 places the player's cards on the screen.

Line 170 asks the player if he wants to go first. The variables

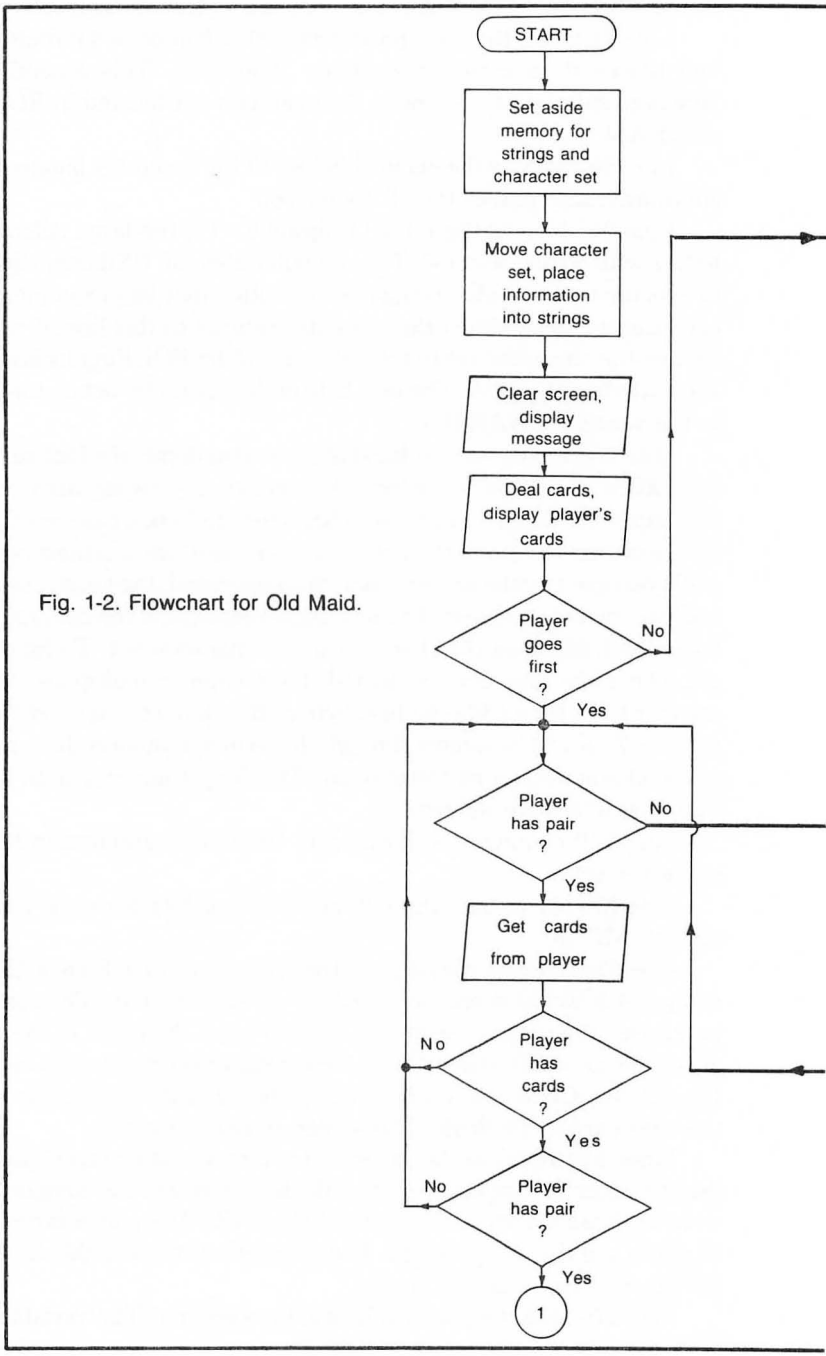
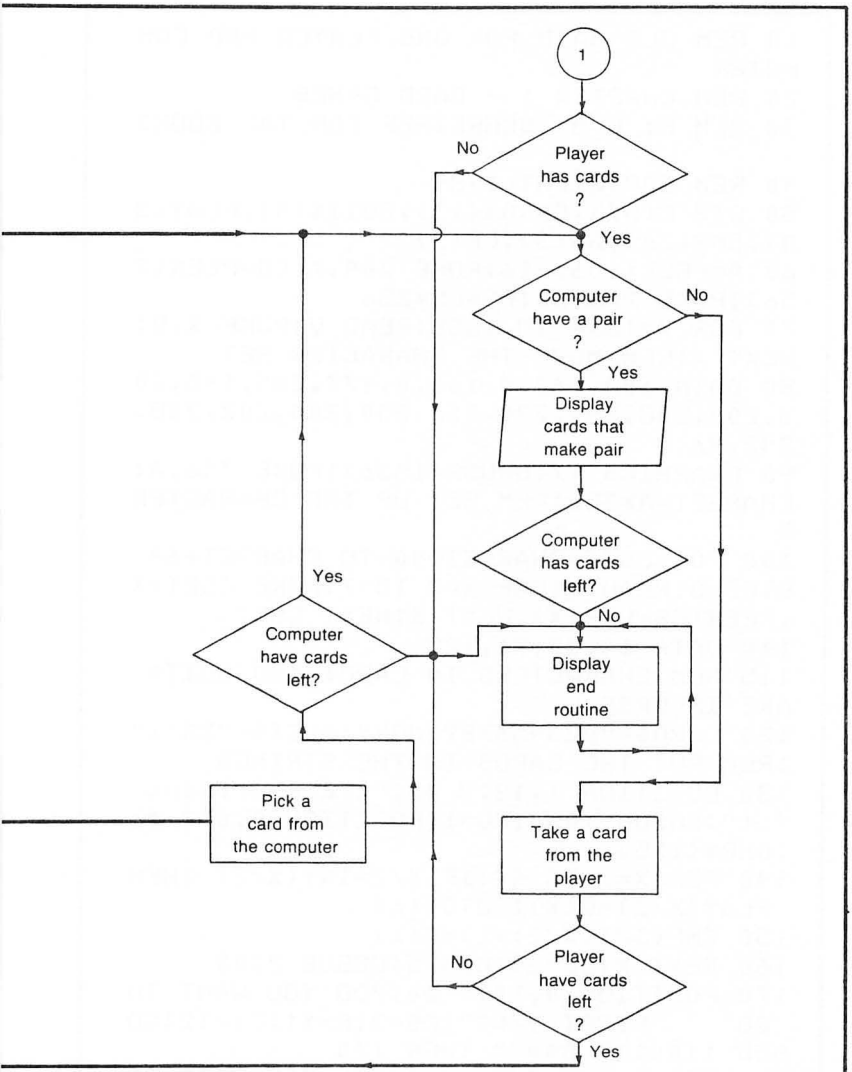


Fig. 1-2. Flowchart for Old Maid.


1
Player has cards ?
No
Yes
Computer have a pair ?
No
Yes
Display cards that make pair
Computer has cards left?
No
Yes
Computer have cards left?
Display end routine
Pick a card from the computer
Take a card from the player
Player have cards left ?
No
Yes

Listing 1-2. Old Maid.

```
10 REM OLD MAID FOR ONE PLAYER AND COM
PUTER
20 REM CHAPTER 1 - CARD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS

40 REM COPYRIGHT 1984
50 DIM C(52),CARD$(13),SUIT$(4),PLAY(3
0),CMP(26),B$(3),CP(2)
60 A=PEEK(106)-16:POKE 204,A:CS=PEEK(7
56):POKE 206,CS:CS=CS*256
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256:REM SET UP THE CHARACTER
S
100 FOR CSET=CHARSET+40 TO CHARSET+64
STEP 8:READ V:FOR X=0 TO 7:POKE CSET+X
,PEEK(CS+V*8+X):NEXT X:NEXT CSET
110 DATA 64,80,96,123
115 REM CHARACTERS IN CARD$ AND SUIT$
ARE INVERSE
120 CARD$="A234567891JQK":SUIT$="%&'("
:REM PUT THE CARDS IN THE STRINGS
130 POSITION 1,12:? #6;"% & shuffling
' (":GOSUB 1050:CD=1:POSITION 0,0:? #6
;CHR$(125)
140 FOR X=1 TO 49:IF X/2=INT(X/2) THEN
 PLAY(X/2)=C(X):GOTO 160
150 CMP(INT(X/2)+1)=C(X)
160 NEXT X:LC=24:CC=25:GOSUB 2000
170 POSITION 0,10:? #6;"DO YOU WANT TO
 GO      FIRST Y/N?":CS=3:R=11:C1=12:GO
SUB 1100:IF B$="" THEN 170
180 IF B$="N" THEN 400:REM COMPUTER GO
ES FIRST
190 POSITION 0,10:? #6;"
                                ":CP(1)=0:CP
(2)=0:REM 40 BLANKS FOR 2 ROWS
200 POSITION 0,10:? #6;"FIRST CARD IS
-":CS=1:R=10:C1=14:GOSUB 1100:IF B$=""
 THEN 370
210 FOR X=1 TO 13:IF ASC(CARD$(X,X))-1
```

```
28=ASC(B$(1,1)) THEN L=LEN(B$):CP(1)=X
+(ASC(B$(L,L))-36)/10:GOTO 230
220 NEXT X
230 POSITION 4,11:? #6;"NEXT CARD -":C
S=1:R=11:C1=14:GOSUB 1100:IF B$="" THE
N 230
240 FOR X=1 TO 13:IF ASC(CARD$(X,X))-1
28=ASC(B$(1,1)) THEN L=LEN(B$):CP(2)=X
+(ASC(B$(L,L))-36)/10:GOTO 260
250 NEXT X
260 IF INT(CP(1))<>INT(CP(2)) OR ABS(C
P(1)-CP(2))<>0.2 THEN 340
270 FOR X=1 TO LC:IF CP(1)<>PLAY(X) TH
EN NEXT X:CARD=CP(1):GOTO 320
280 PLAY(X)=0:GOSUB 3020:FOR X=1 TO LC
:IF CP(2)<>PLAY(X) THEN NEXT X:CARD=CP
(2):GOTO 320
290 PLAY(X)=0:GOSUB 550
300 IF LC>0 THEN GOSUB 2000:GOTO 400:R
EM COMPUTERS TURN
310 ? #6;CHR$(125):GOTO 600:REM END OF
 GAME
320 GOSUB 3000:POSITION 0,14:? #6;"NOT
 FOUND ";:Z=1:X=5:Y=14:GOSUB 3010:GOSU
B 1000
330 GOSUB 4000:POSITION 0,14:? #6;"
              ":TRAP 40000:GOTO 190
340 GOSUB 3000:POSITION 0,14:? #6;"NOT
 A MATCH":GOTO 330
370 IF LC=30 THEN 200:REM MUST HAVE A
PAIR
380 POSITION 0,14:? #6;"i have ";CC;"
cards":? #6;"PICK ONE":CS=4:R=15:C1=10
:GOSUB 1100:IF B$="" THEN 380
385 B=VAL(B$):IF B>CC THEN 380
390 LC=LC+1:PLAY(LC)=CMP(B):CMP(B)=CMP
(CC):CMP(CC)=0:CC=CC-1:IF CC=0 THEN 60
0
395 GOSUB 2000
400 FOR Q=1 TO CC-1:FOR V=Q+1 TO CC:IF
 INT(CMP(Q))<>INT(CMP(V)) THEN NEXT V:
GOTO 470:REM CARDS DON'T MATCH
410 IF ABS(CMP(Q)-CMP(V))<>0.2 THEN NE
XT V:GOTO 470:REM SUITS DON'T MATCH
420 POSITION 0,14:? #6;"I HAVE A MATCH
 ";:X=6:Z=1:Y=14:CARD=CMP(Q):GOSUB 100
```

```
0:POSITION 11,15:? #6;"AND ";
430 X=6:Y=15:CARD=CMP(V):GOSUB 1000:GO
SUB 3020:GOSUB 3100:CMP(Q)=0:CMP(V)=0:
IF CC=2 THEN CC=0:GOTO 600
440 FOR X=1 TO CC-1:IF CMP(X)=0 THEN F
OR Y=X+1 TO CC:CMP(Y-1)=CMP(Y):NEXT Y:
CC=CC-1:GOTO 440
450 NEXT X:IF CMP(X)=0 THEN CC=CC-1
460 GOSUB 2000:GOTO 190
470 NEXT Q
500 GOSUB 3010:POSITION 0,14:? #6;"i m
ust pick from you.":C1=INT(RND(1)*LC)+
1
510 POSITION 0,15:? #6;"i picked a ";:
Z=1:X=4:Y=15:CARD=PLAY(C1):GOSUB 1000:
GOSUB 3100
520 CC=CC+1:CMP(CC)=CARD:PLAY(C1)=PLAY
(LC):LC=LC-1:IF LC=0 THEN 600
530 GOTO 460
550 IF LC=2 AND PLAY(1)=0 AND PLAY(2)=
0 THEN LC=0:RETURN
560 FOR X=1 TO LC-1:IF PLAY(X)=0 THEN
FOR Y=X+1 TO LC:PLAY(Y-1)=PLAY(Y):NEXT
 Y:LC=LC-1:GOTO 560
570 NEXT X:IF PLAY(X)=0 THEN LC=LC-1
580 RETURN
600 POSITION 0,0:? #6;CHR$(125):POSITI
ON 1,10:? #6;"THE WINNER IS ---"
610 IF CC=0 THEN ? #6;"    the compute
r":GOTO 630
620 ? #6;"      THE HUMAN"
630 GOSUB 4000:RUN
1000 CV=INT(CARD):S=(CARD-CV)*10:IF CV
=10 AND X-Z>1 THEN POSITION (X-Z)*3-1,
Y
1005 IF S=1 OR S=3 THEN 1020
1010 ? #6;CARD$(CV,CV);:IF CV=10 THEN
? #6;"0";:REM ZERO IS INVERSE
1015 ? #6;SUIT$(S,S):RETURN
1020 A=ASC(CARD$(CV,CV)):IF A<190 THEN
 ? #6;CHR$(A-32);:IF CV=10 THEN ? #6;C
HR$(144);
1025 IF A>190 THEN ? #6;CHR$(A+32);
1030 A=ASC(SUIT$(S,S)):? #6;CHR$(A-32)
:RETURN
1050 Q=0:FOR V=0.1 TO 0.4 STEP 0.1:FOR
```

```
 X=1 TO 13:C(X+Q)=X+V:NEXT X:Q=Q+13:NE
XT V
1060 C(26)=C(50):REM GET RID OF ONE QU
EEN & TWO KINGS
1070 FOR X=1 TO 5:FOR Q=49 TO 1 STEP -
1:V=INT(RND(1)*Q):REM PICK A CARD
1080 C(0)=C(V):C(V)=C(Q):C(Q)=C(0):REM
 MOVE THE CARDS
1090 NEXT Q:NEXT X:RETURN :REM DO IT 5
 TIMES
1100 B=1:B$="":OPEN #2,4,0,"K:"
1110 GET #2,K:IF K=155 THEN 1220
1120 IF K>127 THEN K=K-128:POKE 694,0:
REM RESET INVERSE FLAG
1130 IF K=126 AND B>1 THEN B=1:B$="
":B$="":POSITION C1+B,R:? #6;"   ":CS=
1:GOTO 1110
1140 IF K>95 THEN K=K-96:POKE 702,64:R
EM SET FOR UPPERCASE
1150 IF CS=1 AND B<4 THEN IF (K>47 AND
 K<58) OR K=65 OR K=74 OR K=75 OR K=81
 THEN 1200
1160 IF CS=2 AND B<4 THEN IF K=67 OR K
=68 OR K=72 OR K=83 THEN 1230
1170 IF CS=3 AND B<4 THEN IF K=89 OR K
=78 THEN CS=0:GOTO 1200
1180 IF CS=4 AND B<3 THEN IF (K>47 AND
 K<58) THEN 1200
1190 GOTO 1110
1200 B$(B,B)=CHR$(K):POSITION C1+B,R:?
 #6;CHR$(K):B=B+1:IF CS=1 THEN CS=2:IF
 B=2 AND K=49 THEN CS=1
1210 IF CS<>0 THEN 1110
1220 CLOSE #2:RETURN
1230 IF K=67 THEN K=38
1240 IF K=68 THEN K=39
1250 IF K=72 THEN K=37
1260 IF K=83 THEN K=40
1270 GOTO 1200
2000 POSITION 0,0:? #6;CHR$(125):Y=1:Z
=1:FOR X=1 TO LC:CARD=PLAY(X):POSITION
 (X-Z)*3,Y
2010 GOSUB 1000:IF X/7=INT(X/7) THEN Y
=Y+2:Z=Z+7
2020 NEXT X
2030 POSITION 0,20:? #6;"HUMAN":POSITI
```

```
ON 2,21:? #6;LC:POSITION 10,20:? #6:"c
omputer":POSITION 14,21:? #6;CC
2040 RETURN
3000 SOUND 0,50,10,10:FOR TI=1 TO 50:N
EXT TI:SOUND 0,0,0,0:RETURN
3010 SOUND 0,200,10,10:FOR TI=1 TO 10:
NEXT TI:SOUND 0,0,0,0:RETURN
3020 FOR ZZ=10 TO 200 STEP 10:SOUND 0,
ZZ,10,10:FOR TI=1 TO 10:NEXT TI:NEXT Z
Z:SOUND 0,0,0,0:RETURN
3100 FOR TI=1 TO 500:NEXT TI:RETURN
4000 IF PEEK(53279)<>6 THEN 4000
4010 RETURN
```

CS, R, and C1 are set. The variable CS indicates what type of input is expected. The variables R and C1 are the row and column for the input. The computer uses the subroutine in line 1100 to get the entry. If the Return key is pressed and nothing was entered, B$ will be empty and null. The computer loops at this line until N or Y is entered.

Line 180 checks to see if N was entered. If so, the computer is directed to line 400. The computer will play first.

Line 190 clears the question from the screen and clears both elements of the array.

Line 200 prints a message of the screen, then sets the variables for the input routine, the row and column. The computer uses the input subroutine at line 1100 to get the first card of the pair. If nothing is entered, B$ will be empty and the computer is sent to line 370 to pick a card.

Lines 210-220 convert the card entered into its decimal equivalent. Each card is stored as a whole number. The suit is the decimal added to the whole number.

Line 230 gets the second card of the pair. The variables are set before the computer uses the subroutine at line 1100. The computer loops at this line until a card is entered.

Lines 240-250 convert the second card into its decimal equivalent.

Line 260 checks the whole number of the card to see if both cards have the same value. The suit is also checked to see if they are both red or black. The value for the same color suits are two-tenths apart, so if the difference between the suit values is two-tenths, both suits are the same color. If either of the value or suit conditions are not met, the computer is sent to line 340 to print

the message on the screen, then ask for the pair again.

Line 270 checks the cards in the player's hand to make sure the player has the first card entered. If the card is not in the player's hand, the computer goes to line 320 and tells the player the card was not found. The computer asks for the pair again.

Line 280 removes the card from the player's hand, makes a sound, then looks for the second card. If it is not in the player's hand, the computer goes to line 320 to tell the player the card could not be found.

Line 290 removes the card from the player's hand, then uses the subroutine in line 550 to move the cards in the player's hand up one position.

Line 300 checks to see if the player has any cards left. If there are cards in the player's hand, the computer uses the subroutine in line 2000 to print the current hand on the screen. The computer is sent to line 400 for the computer's turn.

Line 310 clears the screen and sends the computer to line 600 to end the game.

Line 320 uses the subroutine in line 3000 to make a sound, then prints a message on the screen. This tells the player the card cannot be found.

Line 330 erases the message on the screen and sends the computer to line 190 for another pair.

Line 340 makes a sound, then tells the player that the cards entered were not a match. The computer is sent to line 330 to erase the message and go back to the beginning of the input routine.

Line 370 checks to see how many cards are in the player's hand. If there are 30, the computer goes back to line 200. With that many cards, there has to be a pair.

Line 380 tells the player how many cards are in the computer's hand and instructs the player to pick one. The computer uses the input subroutine in line 1100. The computer loops at this line until a number is entered.

Line 385 finds the value of B$. If the value is more than the number of cards in the computer's hand, the computer goes back to line 380 for another entry.

Line 390 adds 1 to the number of cards in the player's hand. The card that was chosen is removed from the computers hand and placed into the player's hand. The last card in the computer's hand is moved into the position of the card that was taken, and the last card is erased. The variable that counts how many cards are in the computer's hand is decreased by 1. If this variable is 0, the com-

puter goes to line 600 to end the game.

Line 395 sends the computer to the subroutine at line 2000 that places the cards on the screen.

Line 400 begins the computer's turn. The computer begins with the first card in the computer's hand and checks every card after it to see if it matches. If a match cannot be made with the first card, the computer goes to line 470 to continue the loop. The loop continues until every card in the computer's hand has been compared with the others for a possible match.

Line 410 compares the suit values of the two cards. If the suit colors of the two cards are not the same, the computer will continue the loop.

Line 420 is executed when the computer finds two cards of the same value and same suit color. The message is printed on the screen. The first card's value is placed into CARD and the computer uses the subroutine at line 1000 to print this card on the screen.

Line 430 moves the second card value into the variable CARD and the subroutine at line 1000 is used to print this card on the screen. The subroutine at line 3020 makes a sound and the subroutine at line 3100 pauses the program. The two cards are removed from the computer's hand by placing a zero into their position in the array. The computer checks to see if these were the last two cards in the computer's hand. If they were, the variable CC is set to 0 and the computer is sent to line 600 to end the game.

Line 440 begins with the first card of the computer's hand and looks for the elements that contain a zero. This is where the cards were removed. When the computer finds such an element, it moves the rest of the cards in the hand up one position, and decreases the value of CC by 1. The loop continues until all the cards have been moved up.

Line 450 continues the loop to find the elements of the array that contain a zero. The last element of the array is checked to see if it contains a zero. If it does, the variable CC is decreased by 1.

Line 460 uses the subroutine at line 2000 to update the screen. Then the computer is sent to line 190 for the player's turn.

Line 470 continues the loop that looks for a matching pair of cards.

Line 500 begins the routine that picks a card from the player. The computer uses this routine when it can not find a pair of cards in its hand. The computer uses the subroutine at line 3010 to make a sound, then it chooses a random number based on the number

of cards in the player's hand. Remember, the variable LC keeps track of how many cards are in the player's hand.

Line 510 tells the player which card was picked. The variables Z, X, and 15 are set for the correct position on the screen. The value of the card that was picked is moved into CARD and the computer uses the subroutine at line 1000 to print the card on the screen. The subroutine at line 3100 pauses the program for a few seconds.

Line 520 adds 1 to the value of CC because the computer is adding a card to its hand. The card the computer picked is placed into the last element of the array. The last card in the player's hand is moved to the position of the card that was just removed. The variable LC is decreased by 1, then checked to see if it is now 0. If it is, the computer is sent to line 600 to end the game.

Line 530 sends the computer to line 460. This updates the screen, then continues the game with the player's turn.

Line 550 begins the routine that moves the cards up one position in the player's hand. If the variable LC is 2, the last two cards in the player's hand were just played so there are no cards to move up. The variable LC is reset to 0 and the computer returns to the main program.

Line 560 begins the FOR-NEXT loop that moves the cards up. The computer compares the value of the element of the array to zero. If it is zero, the card was played and the computer begins another loop to move up the rest of the cards in the hand. The variable LC is decreased by 1 and the loop begins again.

Line 570 continues the first loop that looks for the element with zero. After this loop is completed, the computer checks the last element of the array to see if it is zero. If it is, the computer subtracts 1 from the value of LC.

Line 580 returns the computer to the main program.

Line 600 begins the ending routine. The screen is cleared and the winning message is printed on the screen.

Line 610 checks the variable CC to see if it is zero. If it is, the computer is the winner and this message is printed on the screen.

Line 620 prints the message that the human is the winner of the game.

Line 630 sends the computer to the subroutine in line 4000. When the computer returns to this line, the program is run again. To stop this program completely, press the System Reset key.

Line 1000 prints the card on the screen. The value of the card must be in the variable CARD before the computer uses this subroutine. The integer or whole number indicates the value of the

card. This value is placed into the variable CV. The value of the suit is placed into the variable S. This value is obtained by multiplying the difference between the variable CARD and the variable CV by 10. The suit value is stored as the decimal in the variable CARD. If the value of the card is 10, the position of the card on the screen is recalculated.

Line 1005 checks the value of the variable S. The hearts and diamonds are printed in red and the spades and clubs are printed in dark blue.

Line 1010 prints the card on the screen. The value of the card is taken from CARD$. If the value of the card is 10, the zero is printed on the screen.

Line 1015 prints the suit character on the screen and returns to the main program.

Line 1020 takes the ASCII value of the card from CARD$. If the value is less than 190 the computer subtracts 32 from the value and prints it on the screen. This is the number in the alternate color. If the value of CV is 10, the zero is printed on the screen.

Line 1025 adds 32 to the value of A if the value of A is greater than 190. This prints the letter on the screen in the other color.

Line 1030 prints the suit character on the screen in the alternate color.

Line 1050 begins the subroutine that places the cards into the C array and shuffles the cards. The first FOR-NEXT loop counts from 0.1 to 0.4 by 0.1. These are the four suits for the cards. The second loop counts from 1 to 13. Each time this loop is completed, the variable Q is increased by 13. This variable points to the position in the array where the next card will be placed. Its value is added to the value of X. This way a card is placed in each element of the array.

Line 1060 moves the last Jack into the position held by the second King. Two Kings and one Queen will not be used in this program.

Line 1070 begins two FOR-NEXT loops. The first loop is repeated five times. This is the number of times the deck of cards will be shuffled. The second loop begins with the last card used and continues backwards to the first card. The computer chooses a random card to move in the deck.

Line 1080 places this card in a temporary storage area. The last card in the deck is moved into the position of the card that was just picked. The card in the temporary area is moved to the last position.

Line 1090 continues the loop. Since the number of cards the computer can choose from is smaller by one each time this loop is executed, all the cards can be moved; once the card is placed into the last position, however, it can never be touched again. The second loop continues until the cards have been shuffled five times. The computer then returns to the main part of the program.

Line 1100 begins the input subroutine. The variable B is set to 1. This variable points to the position the character will occupy in the string. The string is cleared and the keyboard is opened.

Line 1110 waits until a key is pressed. The value of that key is placed into the variable K. If its value is 155, the Return key was pressed and the computer is sent to line 1220.

Line 1120 checks the value of K. If it is greater than 127, the inverse key was pressed. To get the normal value of the key, subtract 128 from the value of K. POKE location 694 with a zero to reset the flag for normal input.

Line 1130 checks to see if the value of K is 126. This is the delete key. If it is 126 and the value of B is greater than 1, then the variable B is reset to 1, and the contents of B$ are erased. B$ is set to a null string (""), and the letters or numbers on the screen are erased. The code is reset to 1, and the computer is sent to line 1110 for another entry.

Line 1140 checks to see if the value of K is greater than 95. If it is, the entry was a lowercase. The value of K is decreased by 96 and the location 702 is POKEd with 64. This resets the keyboard for uppercase.

Line 1150 checks the variable CS to see if it is 1. If it is, and the value of B is less than 4, the computer will check the value of K to see if a number key or the letter Q, K, J, or A was entered. The computer is sent to line 1200 if one of these keys was pressed.

Line 1160 is used by the computer if the value of CS is 2 and the value of B is less than four. This line checks to see if one of the letters designating a suit was pressed. The four letters that the computer will accept for suits are H, S, C, and D, for hearts, spades, clubs, and diamonds. When one of these keys are pressed, the computer is directed to line 1230.

Line 1170 is used when the value of CS is set to 3. The computer will only accept the letters N or Y on this line.

Line 1180 is used when the variable CS is set to 4. This line checks for a number key.

Line 1190 sends the computer back to line 1110. The key that was pressed could not be used by the computer.

Line 1200 places the character for the pressed key into B$. This character is then printed on the screen and the value of B is increased by 1. If the value of CS is 1, it is reset to 2 to allow the suit of the card to be entered. However, if the key pressed for the card value was numeral 1, the variable CS is reset to 1 because 0 must be entered after the 1.

Line 1210 sends the computer back to line 1110 if the value of CS is not zero. It is reset to 0 after a Y or N key was entered so the computer does not have to wait for the Return key to be pressed.

Line 1220 closes the keyboard and sends the computer back to the main program.

Lines 1230-1270 change the value of K from the letter key that was pressed to the value of the character of the suit. Once the value has been changed, the computer is sent to line 1200 to print the character on the screen.

Lines 2000-2020 restore the information on the screen. The cards that are in the player's hand are printed on the screen.

Line 2030 prints how many cards are left in the player's and the computer's hands.

Line 2040 sends the computer back to the main program.

Lines 3000-3020 are the sound subroutines used in the program. There are three different types of sounds used in this program.

Line 3100 is a timing loop. It is used to slow the program down.

Lines 4000-4010 are used at the end of the program. The computer loops at line 4000 until the Start key is pressed. Once this key is pressed, the computer returns to the main program.

TAROT CARDS

Objective of the game: To be able to forecast the future.

Directions: This program can be used as a parlor game. The computer asks for the sex and marital status of the player then shuffles the cards and deals them on the screen. The position of the cards determines the future of the player. None of the predictions are very specific; use your own imagination to fill in the details. The flowchart for this program is Fig. 1-3, and Listing 1-3 is the code.

Line 50 sets aside the memory for the arrays and variables. The C array holds the deck of cards; CARD$ and SUIT$ contain the values of the cards and the suit characters. The PLAY array holds the cards in the order they are displayed on the screen. WHO$

holds the information needed to print messages on the screen.

Line 60 determines how much memory is in the computer by PEEKing at location 106. This value is decreased by 16 so it points to memory just above the screen area. This is where the character set will be moved. The value is POKEd into location 204. The first byte of the ROM-based character set is POKEd into location 206. This information is used by the assembly language subroutine that moves the character set from ROM into RAM. The decimal value of the first byte of the character set is ROM is stored in variable CS.

Line 70 reads the assembly language subroutine from line 80 and POKEs it into memory locations 1536-1555.

Line 80 contains the decimal values for the assembly language subroutine that moves the character set from ROM into RAM.

Line 90 changes the screen to graphics 17; which is graphics 1 with no text window. The computer uses the USR command to execute the assembly language that begins in memory location 1536. When the computer returns to this line, it changes the character set to the one in RAM by POKing location 756 with the value of A. The first byte of this character set is stored in the variable CHARSET.

Line 100 begins the FOR-NEXT loop that changes some of the characters in the character set. The computer reads a value from line 110. This is the location of the character that will be moved from the ROM character set into the RAM character set. The second FOR-NEXT loop takes each of the eight bytes and moves them into RAM. These characters are the four graphic characters that represent the four suits. The computer cannot display both numbers and graphic characters in the large color letter mode, so we replace some of the characters that will not be used with these four characters.

Line 110 contains the location of the four characters.

Line 120 sets CARD$ to the values of the cards, SUIT$ to the four characters that represent the suits, and WHO$ to specific phrases that will be used in the program.

Line 130 prints a message on the screen while the computer uses the subroutine at line 1050 to place the cards into the C array and shuffle the cards. When the computer returns to this line, the screen is cleared.

Line 140 sets the variable CS to 1. This tells the computer which letters to accept in the input routine. The computer prints a message on the screen and uses the subroutine in line 1100 to get an answer.

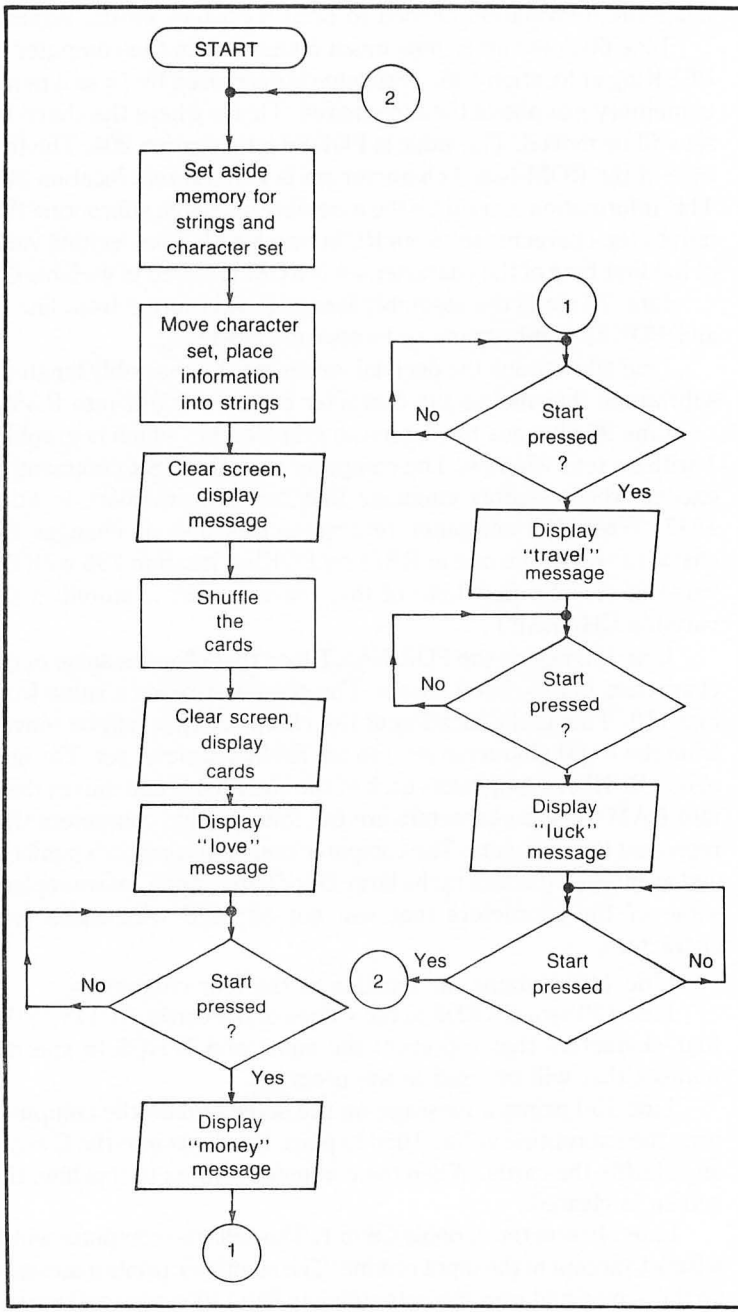


Fig. 1-3. Flowchart for Tarot.

Listing 1-3. Tarot.

```
10 REM TAROT CARDS
20 REM CHAPTER 1 - CARD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1984
50 DIM C(52),CARD$(6),SUIT$(4),PLAY(7,
5),WHO$(72)
60 A=PEEK(106)-16:POKE 204,A:CS=PEEK(7
56):POKE 206,CS:CS=CS*256
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256:REM SET UP THE CHARACTER
S
100 FOR CSET=CHARSET+40 TO CHARSET+64
STEP 8:READ V:FOR X=0 TO 7:POKE CSET+X
,PEEK(CS+V*8+X):NEXT X:NEXT CSET
110 DATA 64,80,96,123
115 REM CHARACTERS IN CARD$ AND SUIT$
ARE INVERSE
120 CARD$="A91JQK":SUIT$="%&'(":WHO$="
MANY PEOPLE YOUR FAMILY YOUNG WOMAN YO
UNG MAN   YOUR WIFE   AN ASSOCIATE"
130 POSITION 1,12:? #6;"% & the tarot
' (":GOSUB 1050:CD=1:POSITION 0,0:? #6
;CHR$(125)
140 CS=1:POSITION 0,3:? #6;"Male or Fe
male?":GOSUB 1100:IF K=77 THEN SEX=0:G
OTO 160
150 SEX=1:WHO$(49,72)="GOOD FRIEND YOU
R HUSBAND"
160 CS=2:POSITION 0,5:? #6;"Married or
 Single?":GOSUB 1100:IF K=77 THEN STAT
=0:GOTO 180
170 STAT=1:WHO$(49,72)="GOOD FRIEND AN
 ASSOCIATE"
180 FOR X=0 TO 7:FOR Y=0 TO 5:PLAY(X,Y
)=0:NEXT Y:NEXT X
190 POSITION 0,0:? #6;CHR$(125):Q=0:Z=
1:Y=1:FOR X=1 TO 24:CARD=C(X):POSITION
 (X-Z)*3,Y:GOSUB 1000
200 PLAY(X-Z+1,INT(Y/2)+1)=CARD
```

```
210 IF X/6=INT(X/6) THEN Y=Y+2:Z=Z+6
215 REM love IS LOWER CASE INVERSE
220 NEXT X:NR=11:POSITION 0,NR:? #6;"l
ove":CARD=5.1:NR=NR+1:IF  NOT SEX THEN
 CARD=6.1
230 FOR Y=1 TO 4:FOR X=1 TO 6:IF PLAY(
X,Y)=CARD THEN 250
240 NEXT X:NEXT Y
250 MC=X:MR=Y:Y=MR-1:L=0:MON=0:TRA=0:L
UCK=0
260 IF (MC<3 AND (MR=1 OR MR=4)) OR (M
C>4 AND (MR=1 OR MR=4)) THEN POSITION
0,NR:? #6;"SOMETIMES FEEL ALONE"
270 FOR X=MC-1 TO MC+1:IF PLAY(X,Y)=0
OR (X=MC AND Y=MR) THEN NEXT X:GOTO 30
0
280 NC=PLAY(X,Y)-INT(PLAY(X,Y)):IF NC<
>0.1 THEN NEXT X:GOTO 300
285 SOUND 0,100,10,10:FOR Z=1 TO 10:NE
XT Z:SOUND 0,0,0,0
290 NC=INT(PLAY(X,Y)):POSITION 0,NR:?
#6;"CLOSE - ";:? #6;WHO$((NC-1)*12+1,(
NC-1)*12+12):NR=NR+1:L=1:NEXT X
300 Y=Y+1:IF Y<>MR+2 THEN 270
310 IF L=0 THEN POSITION 0,NR:? #6;"LO
VE ESCAPES YOU":NR=NR+1
320 GOSUB 4000:NR=11:Y=MR-1:POSITION 0
,NR:? #6;"money/gifts":NR=NR+1
330 FOR X=MC-1 TO MC+1:IF PLAY(X,Y)=0
OR (MC=X AND MR=Y) THEN NEXT X:GOTO 36
0
340 NC=PLAY(X,Y)-INT(PLAY(X,Y)):IF NC<
>0.3 THEN NEXT X:GOTO 360
345 SOUND 0,100,10,10:FOR Z=1 TO 10:NE
XT Z:SOUND 0,0,0,0
350 NC=INT(PLAY(X,Y)):POSITION 0,NR:?
#6;"FROM ";:? #6;WHO$((NC-1)*12+1,(NC-
1)*12+12):NR=NR+1:MON=1:NEXT X
360 Y=Y+1:IF Y<>MR+2 THEN 330
370 IF MON=0 THEN POSITION 0,NR:? #6;"
MANY LOSSES"
375 REM TRAVEL IS INVERSE
380 GOSUB 4000:NR=11:Y=MR-1:POSITION 0
,NR:? #6;"TRAVEL":NR=NR+1
390 FOR X=MC-1 TO MC+1:IF PLAY(X,Y)=0
OR (MC=X AND MR=Y) THEN NEXT X:GOTO 42
0
```

```
400 NC=PLAY(X,Y)-INT(PLAY(X,Y)):IF NC<
>0.2 THEN NEXT X:GOTO 420
405 SOUND 0,100,10,10:FOR Z=1 TO 10:NE
XT Z:SOUND 0,0,0,0
410 NC=INT(PLAY(X,Y)):POSITION 0,NR:?
#6;"WITH ";:? #6;WHO$((NC-1)*12+1,(NC-
1)*12+12):NR=NR+1:TRA=1:NEXT X
420 Y=Y+1:IF Y<>MR+2 THEN 390
430 IF TRA=0 THEN POSITION 0,NR:? #6;"
NO TRAVELS"
440 GOSUB 4000:NR=11:Y=MR-1:POSITION 0
,NR:? #6;"LUCK":NR=NR+1
450 FOR X=MC-1 TO MC+1:IF PLAY(X,Y)=0
OR (MC=X AND MR=Y) THEN NEXT X:GOTO 48
0
460 NC=PLAY(X,Y)-INT(PLAY(X,Y)):IF NC<
>0.4 THEN NEXT X:GOTO 480
465 SOUND 0,100,10,10:FOR Z=1 TO 10:NE
XT Z:SOUND 0,0,0,0
470 NC=INT(PLAY(X,Y)):POSITION 0,NR:?
#6;"STRESS ";:? #6;WHO$((NC-1)*12+1,(N
C-1)*12+12):NR=NR+1:LUCK=1:NEXT X
480 Y=Y+1:IF Y<>MR+2 THEN 450
490 IF LUCK=0 THEN POSITION 0,NR:? #6;
"NO TRAGEDIES"
500 GOSUB 4000:RUN
1000 CV=INT(CARD):S=(CARD-CV)*10:IF CV
=10 AND X-Z>1 THEN POSITION (X-Z)*3-1,
Y
1005 IF S=1 OR S=3 THEN 1020
1010 ? #6;CARD$(CV,CV);:IF CV=3 THEN ?
 #6;"0";:REM ZERO IS INVERSE
1015 ? #6;SUIT$(S,S):RETURN
1020 A=ASC(CARD$(CV,CV)):IF A<190 THEN
 ? #6;CHR$(A-32);:IF CV=3 THEN ? #6;CH
R$(144);
1025 IF A>190 THEN ? #6;CHR$(A+32);
1030 A=ASC(SUIT$(S,S)):? #6;CHR$(A-32)
:RETURN
1050 Q=0:FOR V=0.1 TO 0.4 STEP 0.1:FOR
 X=1 TO 6:C(X+Q)=X+V:NEXT X:Q=Q+6:NEXT
 V
1060 FOR X=1 TO 5:FOR Q=24 TO 1 STEP -
1:V=INT(RND(1)*Q):REM PICK A CARD
1070 C(0)=C(V):C(V)=C(Q):C(Q)=C(0):REM
 MOVE THE CARDS
```

```
1080 NEXT Q:NEXT X:RETURN :REM DO IT 5
 TIMES
1100 OPEN #2,4,0,"K:"
1110 GET #2,K:IF K=155 THEN 1110
1120 IF K>127 THEN K=K-128:POKE 694,0
1130 IF K>95 THEN K=K-96:POKE 702,64
1140 IF CS=1 THEN IF K=77 OR K=70 THEN
 1170
1150 IF CS=2 THEN IF K=77 OR K=83 THEN
 1170
1160 GOTO 1110
1170 CLOSE #2:RETURN
4000 IF PEEK(53279)<>6 THEN 4000
4010 FOR X=11 TO NR:POSITION 0,X:? #6;
"                    ":NEXT X
4020 RETURN
```

The computer needs to know the sex of the player; if M was entered, the variable SEX is set to 0 and the computer is sent to line 160.

Line 150 sets the variable SEX to 1 and changes some of the phrases in WHO$.

Line 160 sets the CS variable to 2. The computer will now look for an M or S in the input routine. The second question is printed on the screen and the computer uses the subroutine in line 1100 to get the response. If M was entered, the STAT variable is set to 0 and the computer is sent to line 180.

Line 170 sets the STAT variable to 1 and changes some of the phrases in WHO$.

Line 180 clears the PLAY array. The computer does not clear string or variable arrays when the program is run.

Line 190 clears the screen, sets the Q variable to 0, Z to 1 and Y to 1. The Z variable is used to place the card in the correct column on the screen. The Y variable is the row value. The FOR-NEXT loop places the cards in the PLAY array and on the screen. The card to be printed on the screen is placed in the CARD variable. The position is calculated and the computer is sent to the subroutine in line 1000 to print the card on the screen.

Line 200 places the card into the PLAY array.

Line 210 checks the value of X to see if the computer is at the end of the line. If it is, the Z variable is increased by 6.

Line 220 continues the FOR-NEXT loop. The NR variable is set to 11. This is where the message will be printed. The first topic

in which the future is forecast is LOVE. The value of CARD is set to the queen of hearts, but, if the value of SEX is 0, it is changed to the king of hearts.

Line 230 looks for the king or queen of hearts in the array.

Line 240 continues the loops.

Line 250 stores the values for variables X and Y in variables MC and MR. This is the position of the heart card that we will be comparing with other cards on the screen. The Y variable is set to the position of MR – 1. The value of L, MON, TRA, and LUCK are set to 0. These four variables represent four areas—luck, money, travel, and love—for which the computer will tell the future.

Line 260 checks to see where the card is located. If it is along one of the edges of the screen, a message will appear on the screen.

Line 270 checks the row above the card for a card. If there are no cards in this row, the computer is sent to line 300.

Line 280 checks the card to see if it is a heart. If it is not, the loop continues.

Line 285 plays a short tone.

Line 290 prints a message on the screen based on the value of the card next to the heart. This indicates the person to whom the player is emotionally close.

Line 300 adds one to the value of Y; this is the row under the row just searched. The loop continues until the row above, the row below, and the row where the heart is located is searched.

Line 310 checks if any hearts surround the king or queen of hearts. If none do, the L variable will remain 0 and a different message is printed on the screen.

Line 320 uses the subroutine in line 4000 to keep the message on the screen. When a player wants to advance to the next message, press the Start key. The variable NR and Y are reset, and a new message appears on the screen.

Lines 330-370 use the same routines to find the diamonds that surround the heart. Diamonds usually depict money, gifts, or good fortune. If any diamonds surround the heart, the computer will make a sound and the appropriate messages will be printed on the screen. If there are no diamonds, a different message is printed on the screen.

Line 380 uses the subroutine in line 4000 and waits for the Start key to be pressed. The variables are reset and a new message is placed on the screen.

Lines 390-430 search for a club. If a club is near the heart, the computer makes a sound and a message is printed on the screen.

If no clubs are near, there is no travel in the player's future.

Line 440 waits for the Start key to be pressed. Then the computer will tell the player what kind of luck to expect.

Lines 450-490 look for spades. Spades are the misfortune cards; the fewer spades around the heart the better. If any exist, the computer prints a message on the screen. If there are none, the computer prints a better outlook on the future.

Line 500 is the end of the program. The computer uses the subroutine in line 4000 to wait for the Start key. When it returns to this line, the program is run again. To end the program, press the System Reset key.

Lines 1000-1030 print the cards on the screen. The values of the printed cards must be stored in the CARD variable before this subroutine can be used. The value of each card is stored in the variable CV, and the suit value in the S variable. The suit of the card is checked; if the card is a heart or a diamond the routine in line 1020 is used to print the card on the screen. Hearts and diamonds are printed in red and spades and clubs in dark blue. If the value of the card is 10, the zero is printed after the one. Once the card is printed on the screen in the correct color, the computer returns to the main program.

Lines 1050-1080 place the cards in the array and shuffle them. The suit of the card is determined by the decimal value added to the card value. The two FOR-NEXT loops place the deck of cards into the array. The cards are shuffled by removing one card from the deck and placing it into a temporary element of the array. The card in the last element of the array is moved to the position of the card that was removed. The card in temporary storage is moved to the last element of the array. The loop counts backwards; once a card is placed in the last element of the array, the pointer is moved to the element just before it. The entire array is shuffled five times.

BLACKJACK

Objective of the game: To try to collect 21, or as many points as possible without going over 21, while collecting more points than the dealer.

Directions: This program is written for one or two players. The program begins by placing one player on the screen. Press the Select key to change the number of players to two. The program will alternate between one and two players. Press the Start key when you are ready to begin.

The screen clears after a few seconds and you are asked to bid. Bids must be between $1 and $500. Next the cards are dealt on

the screen. The dealer will have one card face down and the other showing. Each player is dealt two cards, both showing.

If the dealer has an ace showing, you can take insurance. If the dealer has blackjack you will lose your bet if you do not have insurance and also have a blackjack.

On your turn you can press the B key if you have blackjack, the D key to double your bet on your first turn, the H key to get another card, or the S key to stand. After each player plays his hand, the dealer plays his. The dealer is required to stand on a "hard" 17 (i.e., one not containing a one-or-eleven ace) but draw to a "soft" 16.

A player who holds a blackjack is paid at odds of three to two unless the dealer also holds a blackjack. A winner is paid out even money, and a player who ties the dealer has his wager returned.

After each hand the program asks if you want another hand. If so, another hand will be dealt. If not, the game ends. The game also ends when the players are out of money or a player's winnings exceed $1000. Figure 1-4 is the flowchart for this program, and Listing 1-4 contains the code.

Line 50 sets aside the memory needed for the arrays and strings. The C array holds the deck of cards, CARD$ and SUIT$ hold the values of the cards and the characters for the suits. The PLAY array holds the cards the players acquire, and the B array keeps track of he money the players have, the bet, and insurance. B$ is used to store an input.

Line 60 finds out how much memory the computer has and subtracts 2K from this amount. The character set will be moved into RAM just in front of the screen memory. The addresses of the new character set and the old character set are stored in locations 204 and 206. These two values will be used by the assembly language subroutine that moves the character set from ROM into RAM. The first byte of the ROM-based character set is stored in variable CS.

Line 70 reads the code for the assembly language subroutine and places it in memory locations 1536-1555; line 80 contains the decimal codes for the routine.

Line 90 changes the screen to graphics 17, which is graphics 1 with no text window. The computer uses the USR command to execute the assembly language subroutine that begins in memory location 1536. This is the assembly language subroutine that moves the character set from ROM into RAM. When the computer returns to this line, it POKEs the address of the RAM-based character set into location 756. If your screen contains garbage, or the characters

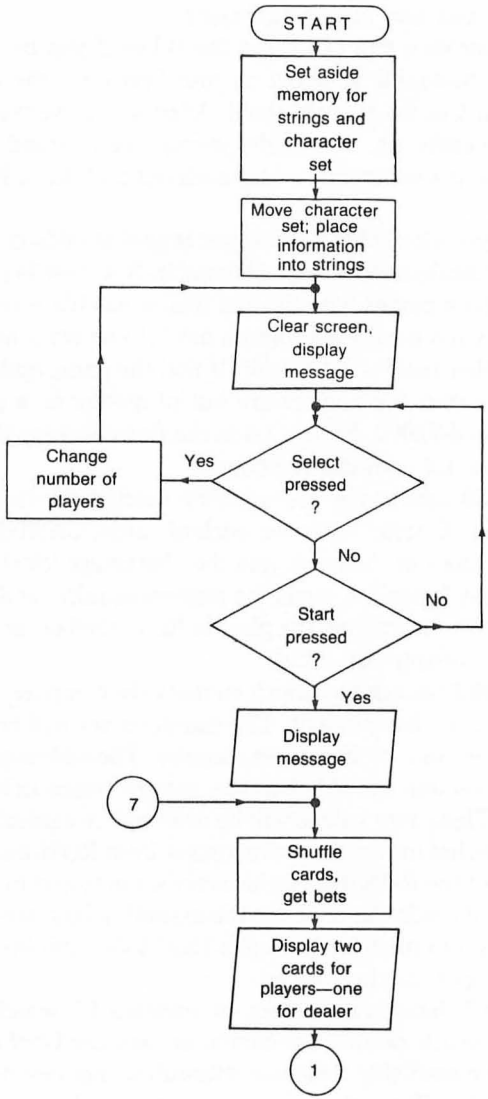
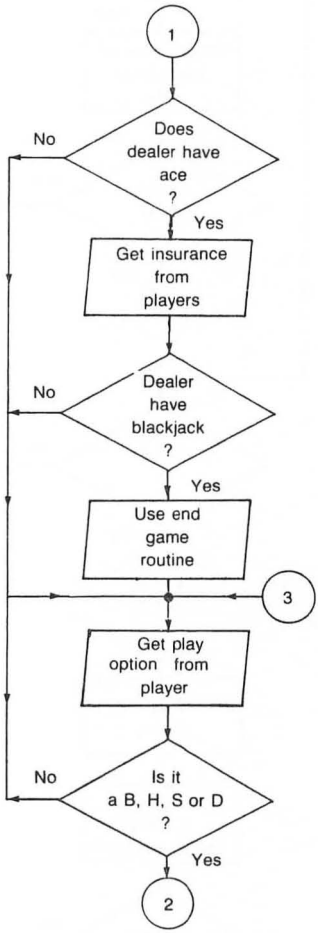
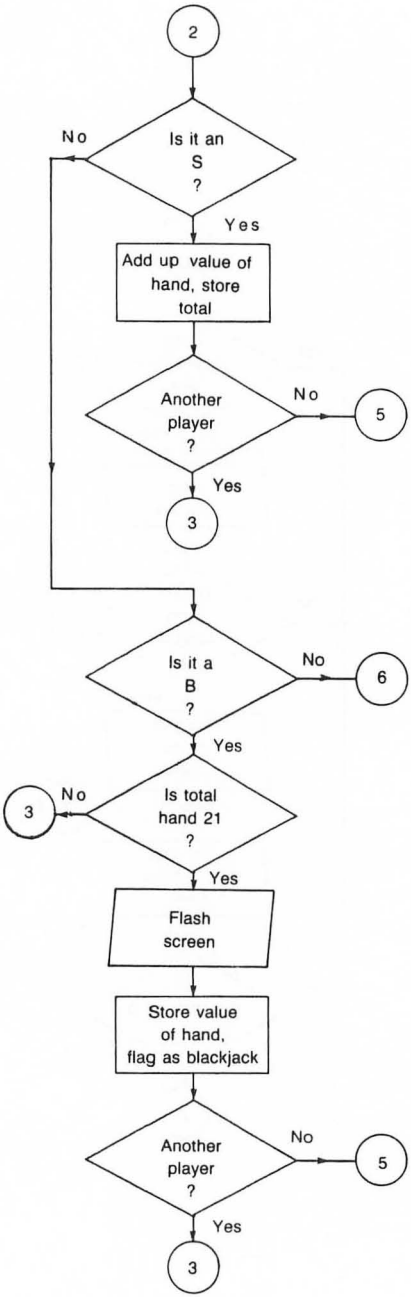
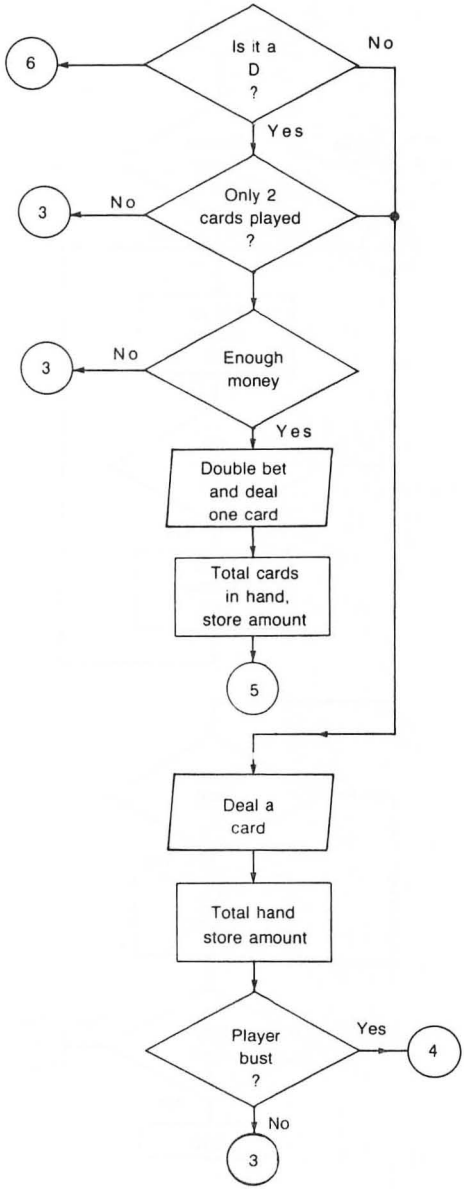


Fig. 1-4. Flowchart for Blackjack. (Continued through page 46.)


1
Does dealer have ace ?
No
Yes
Get insurance from players
Dealer have blackjack ?
No
Yes
Use end game routine
3
Get play option from player
Is it a B, H, S or D ?
No
Yes
2


2
Is it an S ?
No
Yes
Add up value of hand, store total
Another player ?
No
5
Yes
3
Is it a B ?
No
6
Yes
3
No
Is total hand 21 ?
Yes
Flash screen
Store value of hand, flag as blackjack
Another player ?
No
5
Yes
3


Is it a D ?
No
6
Yes
Only 2 cards played ?
No
3
Enough money
No
3
Yes
Double bet and deal one card
Total cards in hand, store amount
5
Deal a card
Total hand store amount
Player bust ?
Yes
4
No
3

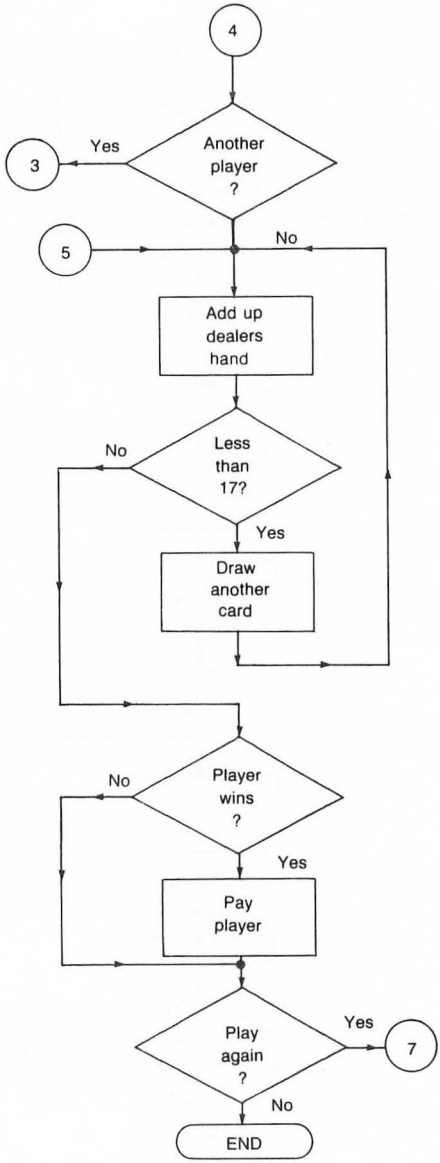


Fig. 1-4 continued.

Listing 1-4. Blackjack.

```
10 REM 21 - BLACKJACK
20 REM CHAPTER 1 - CARD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS

40 REM COPYRIGHT 1984
50 DIM C(52),CARD$(13),SUIT$(4),PLAY(2
,13),B(2,3),B$(4)
60 A=PEEK(106)-16:POKE 204,A:CS=PEEK(7
56):POKE 206,CS:CS=CS*256
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256:REM SET UP THE CHARACTER
S
100 FOR CSET=CHARSET+40 TO CHARSET+64
STEP 8:READ V:FOR X=0 TO 7:POKE CSET+X
,PEEK(CS+V*8+X):NEXT X:NEXT CSET
110 DATA 64,80,96,123
115 REM CHARACTERS IN CARD$ AND SUIT$
ARE INVERSE
120 CARD$="A234567891JQK":SUIT$="%&'("
:REM PUT THE CARDS IN THE STRINGS
125 REM number of players IS LOWER CAS
E INVERSE
130 SOUND 0,100,10,10:? #6;CHR$(125):P
L=1:POSITION 1,5:? #6;"number of playe
rs":SOUND 0,0,0,0
140 POSITION 9,10:IF PEEK(53279)=5 THE
N PL=PL+1:IF PL=3 THEN PL=1:REM GET TH
E NUMBER OF PLAYERS
150 IF PEEK(53279)=6 THEN 170
160 ? #6;PL:FOR X=1 TO 100:NEXT X:GOTO
 140:REM TIMING LOOP FOR SWITCHES
170 FOR X=1 TO 2:B(X,1)=500:B(X,2)=0:B
(X,3)=0:NEXT X:IF PL=1 THEN B(2,1)=0
180 POSITION 1,12:? #6;"% & shuffling
' (":GOSUB 1050:CD=0
190 FOR X=0 TO 2:FOR V=0 TO 13:PLAY(X,
V)=0:NEXT V:NEXT X
195 REM 21 IS INVERSE
200 POSITION 0,0:? #6;CHR$(125):POSITI
```

```
ON 9,0:? #6;"21":POSITION 0,2:? #6;"de
aler"
210 FOR X=1 TO PL:POSITION X*10-10,6:?
 #6;"PLAYER #";X:B$=STR$(B(X,1)):POSIT
ION X*10-9,23:? #6;"$    "
220 POSITION X*10-5-LEN(B$),23:? #6;B$
:NEXT X:FOR X=1 TO PL:IF B(X,1)=0 THEN
 280:REM SETUP-SKIP A PLAYER WITH NO M
ONEY
230 R=4:SOUND 0,100,10,10:POSITION 0,R
:? #6;"BID-PLAYER #";X;" $     ":CO=15:
CS=1:SOUND 0,0,0,0:GOSUB 1100
240 B$=B$(1,CO-15):B=VAL(B$):IF B<1 OR
 B>500 THEN GOTO 230
250 IF B>B(X,1) THEN 230
260 B(X,2)=B:B(X,1)=B(X,1)-B:POSITION
X*10-9,22:? #6;"$    ":POSITION X*10-5-
LEN(B$),22:? #6;B
270 B$=STR$(B(X,1)):POSITION X*10-9,23
:? #6;"$    ":POSITION X*10-5-LEN(B$),2
3:? #6;B$
280 NEXT X
290 P1=1:R=7:POSITION 0,4:? #6;"
              ":REM REMOVE THE BID PROM
PT
300 FOR X=1 TO PL:IF B(X,2)=0 THEN PLA
Y(X,0)=22:GOTO 320
310 CD=CD+1:POSITION X*10-10,R:CARD=C(
CD):GOSUB 1000:PLAY(X,P1)=CARD
320 NEXT X:R=R+1:R2=R:R3=R
330 CD=CD+1:CARD=C(CD):PLAY(0,P1)=CARD
:P1=P1+1:IF P1=2 THEN POSITION 0,3:GOS
UB 1000:GOTO 300
335 REM ASTERISK IS INVERSE
340 POSITION 4,3:? #6;"*":CD=CD+1:IF I
NT(PLAY(0,1))<>1 THEN 410
350 FOR X=1 TO PL:IF B(X,2)=0 THEN 400

355 REM QUESTION MARK IS INVERSE
360 SOUND 0,100,10,10:POSITION 0,0:? #
6;"                    ";:POSITION INT(
X*10-10),9:? #6;"?"
370 POSITION 0,0:? #6;"INSURANCE Y/N?"
:SOUND 0,0,0,0:CS=2:CO=17:R=0:GOSUB 11
00:IF B$="N" THEN 400
380 B(X,3)=INT(B(X,2)/2):B$=STR$(B(X,3
```

```
));POSITION X*10-9,21:? #6;"$":POSITIO
N X*10-5-LEN(B$),21:? #6;B$
390 B(X,1)=B(X,1)-B(X,3):B$=STR$(B(X,1
)):POSITION X*10-9,23:? #6;"$    ":POSI
TION X*10-5-LEN(B$),23:? #6;B$
400 POSITION INT(X*10-10),9:? #6;" ":N
EXT X
410 IF INT(PLAY(0,1))<10 AND INT(PLAY(
0,1))<>1 THEN 500
420 X=0:GOSUB 1210:REM SEE IF DEALER H
AS BLACKJACK
430 POSITION 0,0:? #6;"
    ":IF TC<>21 THEN FOR X=1 TO PL:B(X
,3)=0:NEXT X:GOTO 500
435 REM black-jack IS LOWER CASE INVER
SE
440 SOUND 0,50,10,10:POSITION 5,0:? #6
;"black-jack":CARD=PLAY(0,2):POSITION
4,3:GOSUB 1000:SOUND 0,0,0,0
450 FOR X=1 TO PL:GOSUB 1210:IF TC=-1
THEN B(X,2)=2*B(X,2):IF B(X,3)=0 THEN
B(X,2)=0
460 IF TC<>-1 THEN B(X,2)=0:B(X,3)=0
470 B(X,1)=B(X,1)+B(X,2)+B(X,3):B(X,3)
=0:B(X,2)=0:POSITION X*10-9,21:? #6;"
   ":POSITION X*10-9,22:? #6;"     "
480 B$=STR$(B(X,1)):POSITION X*10-9,23
:? #6;"$    ":POSITION X*10-5-LEN(B$),2
3:? #6;B$:REM SHOW MONEY LEFT
490 NEXT X:FOR V=1 TO 200:NEXT V:GOTO
880:REM ASK FOR ANOTHER GAME
500 R1=R:D=0:FOR X=1 TO PL:R=R1:IF B(X
,2)=0 THEN 620
510 SOUND 0,100,10,10:POSITION 0,0:? #
6;"                    ":POSITION 0,0:?
 #6;"Hit or Stand?":SOUND 0,0,0,0
515 REM QUESTION MARK IS INVERSE
520 R=R2:POSITION X*10-9,R:? #6;"?":CS
=0:CO=13:R=0:GOSUB 1100:R=R2:REM FIND
OUT WHAT THE PLAYER WANTS TO DO
530 IF B$="S" THEN GOSUB 1210:GOTO 600
:REM NEXT PLAYER
540 IF B$="B" THEN GOSUB 1210:IF TC=-1
 THEN GOSUB 630:PLAY(X,0)=TC:GOTO 620:
REM THERE IS A BLACKJACK
550 IF B$="B" AND TC<>-1 THEN 510
```

```
560 IF B$="D" THEN GOSUB 640:D=1:REM S
EE IF BET CAN BE DOUBLED
570 CARD=C(CD):CD=CD+1:PLAY(X,P1)=CARD
:POSITION X*10-10,R:GOSUB 1000:R=R+1:P
1=P1+1:R2=R:GOSUB 1210
575 IF D=1 THEN 600:REM ONLY ONE CARD
ON A DOUBLE DOWN
580 IF TC>21 AND (T1=0 OR T1>21) THEN
600
590 GOTO 510
600 IF (TC<22 AND TC>T1) OR TC=-1 THEN
 PLAY(X,0)=TC:GOTO 620
610 PLAY(X,0)=T1:IF T1=0 OR T1>21 THEN
 PLAY(X,0)=TC
620 POSITION X*10-9,R:? #6;" ":P1=3:R=
R3:R2=R:D=0:NEXT X:GOTO 700
630 FOR Q=0 TO 15:SETCOLOR 4,Q,0:SOUND
 0,Q*10,10,10:NEXT Q:SETCOLOR 4,0,0:SO
UND 0,0,0,0:RETURN
640 IF B(X,1)<B(X,2) THEN POP :GOTO 51
0
650 IF P1>3 THEN POP :GOTO 510
660 B(X,1)=B(X,1)-B(X,2):B(X,2)=2*B(X,
2):B$=STR$(B(X,1)):POSITION X*10-9,23:
? #6;"$    "
670 POSITION X*10-5-LEN(B$),23:? #6;B$
:B$=STR$(B(X,2)):POSITION X*10-9,22:?
#6;"$    "
680 POSITION X*10-5-LEN(B$),22:? #6;B$
:RETURN
690 REM END THE GAME
700 X=0:Q=3:IF PLAY(1,0)>21 AND PLAY(2
,0)>21 THEN 810
710 POSITION 0,0:? #6;"
    ":CARD=PLAY(0,2):POSITION 4,3:GOSU
B 1000:GOSUB 1210:IF TC>16 THEN 760
720 PLAY(X,Q)=C(CD):CARD=C(CD):Q=Q+1:C
D=CD+1:IF Q<7 THEN POSITION (Q-2)*4,3
730 IF Q>6 THEN POSITION (Q-2)*4-20,4
740 GOSUB 1000:GOSUB 1210:IF TC>16 THE
N 760
750 GOTO 720
760 IF T1=21 OR TC=21 OR T2=21 THEN PO
SITION 0,1:? #6;"21-HOUSE WINS":PLAY(0
,0)=21:GOTO 810
770 IF T1<17 AND T1<>0 THEN 720:REM TR
```

```
Y WITH THE LOWER HAND
780 IF T2<17 AND T2<>0 THEN 720
790 PLAY(0,0)=TC:IF T2<>0 AND T2<22 TH
EN PLAY(0,0)=T2
795 IF TC>T1 AND TC<22 THEN PLAY(0,0)=
TC:GOTO 810
800 IF T1>T2 AND T1<22 THEN PLAY(0,0)=
T1
810 FOR X=1 TO PL:IF PLAY(X,0)=-1 THEN
 POSITION X*10-9,20:? #6;"blackjack":B
(X,2)=INT(B(X,2)*2.5):GOTO 860
820 IF PLAY(X,0)>21 THEN B(X,2)=0:GOTO
 860
830 IF PLAY(X,0)>PLAY(0,0) THEN B(X,2)
=2*B(X,2):GOTO 860
840 IF PLAY(0,0)<22 AND PLAY(0,0)>PLAY
(X,0) THEN B(X,2)=0:GOTO 860
850 IF PLAY(0,0)>21 AND PLAY(X,0)<22 T
HEN B(X,2)=B(X,2)*2
860 B(X,1)=B(X,1)+B(X,2):B(X,0)=0:POSI
TION X*10-9,22:? #6;"$    ":POSITION X*
10-9,23:? #6;"$    "
870 B$=STR$(B(X,1)):POSITION X*10-5-LE
N(B$),23:? #6;B$:NEXT X
880 POSITION 0,0:? #6;"
    ":IF B(1,1)>=1000 OR B(2,1)>=1000
THEN 950
890 IF B(1,1)<=0 AND B(2,1)<=0 THEN PO
SITION 4,10:? #6;"house wins!":GOTO 98
0
900 POSITION 0,0:? #6;"PLAY AGAIN?":CO
=11:R=0:CS=2:GOSUB 1100
910 IF B$="N" THEN END
920 IF CD>30 THEN 180
930 GOTO 190
940 REM GAME OVER
950 POSITION 5,0:? #6;"game over!":POS
ITION 0,10:? #6;"PLAYER #";
960 PL=2:IF B(1,1)>B(2,1) THEN PL=1
965 REM wins! IS LOWER CASE INVERSE
970 ? #6;PL:POSITION 7,12:? #6;"wins!"
980 IF PEEK(53279)<>6 THEN 980
990 RUN
1000 CV=INT(CARD):S=(CARD-CV)*10
1005 IF S=1 OR S=3 THEN 1020
1010 ? #6;CARD$(CV,CV);:IF CV=10 THEN
```

```
? #6;"0";:REM ZERO IS INVERSE
1015 ? #6;SUIT$(S,S):RETURN
1020 A=ASC(CARD$(CV,CV)):IF A<190 THEN
 ? #6;CHR$(A-32);:IF CV=10 THEN ? #6;C
HR$(144);
1025 IF A>190 THEN ? #6;CHR$(A+32);
1026 A=ASC(SUIT$(S,S)):? #6;CHR$(A-32)
:RETURN
1030 CV1=0:S=INT(CARD/13):CV=CARD-S*13
+1:IF CV=1 THEN CV1=11
1040 RETURN
1050 Q=0:FOR V=0.1 TO 0.4 STEP 0.1:FOR
 X=1 TO 13:C(X+Q)=X+V:NEXT X:Q=Q+13:NE
XT V
1060 FOR X=1 TO 5:FOR Q=52 TO 1 STEP -
1:V=INT(RND(1)*Q):REM PICK A CARD
1070 C(0)=C(V):C(V)=C(Q):C(Q)=C(0):REM
 MOVE THE CARDS
1080 NEXT Q:NEXT X:RETURN :REM DO IT 5
 TIMES
1100 B=0:B$="":OPEN #2,4,0,"K:"
1110 GET #2,K:IF K=155 AND B$<>"" THEN
 CLOSE #2:RETURN
1120 IF K>127 THEN K=K-128:POKE 694,0:
REM RESET INVERSE FLAG
1130 IF CS=1 AND K=126 AND CO>15 THEN
CO=CO-1:B$(CO-14,CO-14)=" ":POSITION C
O,R:? #6;" ":GOTO 1110
1140 IF K>95 THEN K=K-96:POKE 702,64:R
EM SET FOR UPPERCASE
1150 IF CS=1 AND CO<18 AND (K>47 AND K
<58) THEN B$(CO-14,CO-14)=CHR$(K):GOTO
 1190
1160 IF CS=0 THEN IF K=72 OR K=83 OR K
=68 OR K=66 THEN B$(1,1)=CHR$(K):GOTO
1190
1170 IF CS=2 THEN IF K=89 OR K=78 THEN
 B$(1,1)=CHR$(K):GOTO 1190
1180 GOTO 1110
1190 POSITION CO,R:? #6;CHR$(K):IF CS=
1 THEN CO=CO+1
1200 GOTO 1110
1210 A=0:T2=A:TC=A:T1=A:FOR V=1 TO 13:
I=INT(PLAY(X,V)):IF I=0 THEN 1270
1220 IF I>9 THEN TC=TC+10:T1=T1+10:GOT
O 1250:REM ADD 10 FOR FACE CARD OR 10
1230 IF I=1 THEN TC=TC+11:T1=T1+1:A=A+
```

```
1:GOTO 1250:REM COUNT ACE AS HIGH AND
LOW
1240 TC=TC+I:T1=T1+I
1250 IF A=2 THEN T2=T1+10:REM SPLIT AG
AIN - TWO ACES
1260 NEXT V:RETURN :REM MORE THAN 2 CA
RDS IN HAND
1270 IF X>0 AND TC=21 AND V=3 THEN TC=
-1:REM BLACKJACK!
1280 V=13:NEXT V:RETURN :REM GET RID O
F NUMBERS ON STACK
```

do not look right on the screen, check to see that you have the correct variable and the correct location. The first byte of the character set in RAM is stored in the variable CHARSET.

Line 100 moves four characters from ROM into RAM. Since the computer displays numbers with uppercase letters and the graphics characters with lowercase letters, it is necessary to move the four characters that represent the four suits from ROM and replace characters in the RAM character set. In this program we will replace the percent sign, the ampersand, the apostrophe, and the open parenthesis with the heart, club, diamond, and spade. The location of these characters in the character set are read from the DATA line. This value is multiplied by eight and added to variables X and CS. For all four characters, the computer loops through the FOR-NEXT loop reading each of the eight bytes and moving them into RAM.

Line 110 contains the locations of the four characters in the character set.

Line 120 places the values of the cards into CARD$ and the characters that now represent the four suits into SUIT$.

Line 130 makes a sound, clears the screen, and prints a message on the screen, and prints a message on the screen. The PL variable is set to 1. This is the number of players.

Line 140 checks to see if the Select key has been pressed. If it has, the value of PL is increased by 1. If the value reaches three, it is reset to 1.

Line 150 checks to see if the Start key has been pressed. If so, the computer is sent to line 170 to begin the game.

Line 160 prints the number of players on the screen, executes a timing loop to give the player a chance to release the key, and loops back to line 140.

Line 170 places $500 into each player's kitty, and clears the bet and insurance elements of the B array. If only one person is playing the money in the second player's kitty is removed.

Line 180 begins the next part of the program. The message is printed on the screen. The computer uses the subroutine in line 1050 to place the cards into the C array and shuffle them. When the computer returns to this line, the CD variable is set to 0. This variable keeps track of how many cards have been played so the computer knows when to shuffle the cards again.

Line 190 contains two FOR-NEXT loops that clear the elements of the PLAY array.

Line 200 clears the screen and prints some words on the screen.

Line 210 prints the amount of money available to both players. The value of the first element of variable B is placed as a string into B$. The player's money position is calculated and the old amount is cleared.

Line 220 prints the amount of money currently available for this player, then continues the loop for the next player. After this loop is completed, the computer begins a second FOR-NEXT loop. The amount of money the player has is checked. If the player has no money, the computer is directed to line 280. If there is money, the computer will continue to get a bet.

Line 230 makes a sound and asks for the player's bet. The computer uses the subroutine in line 1100 to retrieve the entry.

Line 240 takes the value entered in B$ and places it into B$. This little manipulation gets rid of any spaces that may be at the end of the string. If, for instance, the amount of $100 was entered and then the player changed his mind and used the delete key to change the amount to $25, the extra space at the end of the string would remain. The value of B$ is placed into variable B. If the amount is less than 1 or greater than 500 the computer is sent back to line 230 to get another bet.

Line 250 checks to see if the player has enough money. If the amount of variable B is greater than the amount of money in the first element of the player's array the computer is sent back to line 230 for another try.

Line 260 stores the amount bet into the second element of that player's array and decreases the amount of money in the first element of the array by the amount bet. The amount of the bet is printed on the screen just above the amount of money the player has.

Line 270 places the amount of money the player has as a string

into B$. This amount is then printed on the screen.

Line 280 continues the loop.

Line 290 removes the bid prompt from the screen.

Line 300 begins the next FOR-NEXT loop; it prints two cards for each player on the screen. The computer checks to see how much money the player bid. If no money has been bid, the computer sets the element of the PLAY array that stores how many points the hand is worth to 22 and the computer is directed to line 320.

Line 310 increases the value of CD by 1. This variable points to the next card in the array that will be played. The value of this card is moved into CARD and the computer uses the subroutine in line 1000 to print the card on the screen. The card is then moved into the correct position of the PLAY array.

Line 320 continues the loop. After both cards have been dealt, the R variable is increased by 1, then stored in variables R2 and R3.

Line 330 adds one to variable CD, moves the value of the next card into CARD, then moves the card value into the dealer's area for the array. The value of P1 is increased by 1. If it then equals 2, the computer will use the subroutine in line 1000 to print the card on the screen. Then the computer is directed to line 300 to print the second card on the screen for each player.

Line 340 prints the asterisk on the screen as the dealer's second card. The variable CD is incremented by 1 so it is pointing to the next card to be dealt. The dealer's first card is checked to see if it is an ace. If it is not, the computer goes to line 410 to continue the game.

Line 350 begins another FOR-NEXT loop. In this line the computer checks the second element of each player's array to see if any money has been bet. A zero here means that a person is not playing. The computer goes to line 400 if the player is not playing.

Line 360 makes a prompting sound, clears the first line on the screen, and prints a question mark in the ninth row.

Line 370 places the question on the top row of the screen. The computer wants to know if the player wants insurance. The variable CS is set to 2; this value is used in the input routine. Variables CO and R are set for the column and row, and the computer uses the subroutine in line 1100 to get an input. Only the letters Y or N will be accepted. When the computer returns to this line, it checks the contents of B$. If it is N, the computer goes to line 400—the player does not want insurance.

Line 380 takes half the amount bet and stores it in the third

element of the array for this player. This is the insurance amount. The variable is converted into a string and this amount is printed on the screen just above the amount bet. The insurance is always half the original bet.

Line 390 subtracts the insurance money from the amount of money the player has in the first element of the array. This new amount is converted into a string and printed on the screen.

Line 400 clears the player's entry, the Y or N, from the screen. The loop continues until both players have been given a chance to buy insurance.

Line 410 checks the value of the dealer's first card again. This is the line the computer is directed to if the first card is not an ace. The computer wants to know if this card is an ace, a ten, or a face card. If it is none of these cards, the computer is sent to line 500.

Line 420 begins the routine that checks to see if the dealer has blackjack. If the dealer has an ace, you are given a chance to insure your bet; however, if the dealer has a ten or face card showing, you cannot get insurance. The dealer is allowed to peek at his other card and call a blackjack if he has one. The variable X is set to 0, and the computer uses the subroutine in line 1210 to find out what the two cards total.

Line 430 clears the message from the top row on the screen. The variable TC contains the total value of the two cards. If it is not 21, the dealer does not have blackjack. The money used as insurance is lost, and the computer is sent to line 500 to continue the same.

Line 440 makes a sound and announces that the dealer has blackjack. The value of the card that was face down is placed into CARD. The subroutine at line 1000 prints the card on the screen.

Line 450 begins the loop that checks the hands of both players. If the value of TC is -1, the player has blackjack and is paid even money on the bet. However, if there was no insurance on this bet everything is lost and the second element of the player's array is set to 0.

Line 460 checks to see if the value of TC is not -1; this means that the player's card count is less than 21. If this is the case, both the money bet and the insurance money is lost. Both elements of this player's array are set to 0.

Line 470 adds the values of the three elements of the player's array and stores the total in the first element of the array. The bet and insurance elements are cleared and the values are erased from the screen.

Line 480 places the value of the first element of the player's array into B$, calculates the correct position on the screen, and prints this amount of the screen. The player now knows how much money he has left.

Line 490 continues the loop for the other player. The timing loop gives the players a chance to study the screen; then the computer goes to line 880 and asks for another game.

Line 500 resets the variables R1 and D. The FOR-NEXT loop begins the routine that deals the rest of the hand to the players. The second element of the player's array is checked. If it is zero, the computer is sent to line 620. This person is not playing.

Line 510 makes a sound, then prints a message in the top row of the screen. You are asked if you want a hit or to stand. You also have the option of pressing the B key for a blackjack or the D key to double your bet on the first turn.

Line 520 prints a question mark on the screen under the last card displayed. The variable CS is set to zero and the computer is sent to the subroutine in line 1100 to set your input. Only the letters B, H, S, and D will be accepted.

Line 530 checks the contents of B$. If it is S, the computer uses the subroutine in line 1210 to set the value of your cards. The computer then goes to line 600 to give the next player a turn.

Line 540 is used if the contents of B$ is B. The computer uses the subroutine in line 1210 to get the value of the hand. If the value of TC is – 1, the player has a blackjack and the computer uses the subroutine in line 630 to change the screen color and make a winning sound. The value of variable TC is placed in the zero element of the player's array and the computer is sent to line 620 to continue the same.

Line 550 sends the computer back to line 510 if the player does not have blackjack.

Line 560 sends the computer to line 640 if the contents of B$ is D. A bet can only be doubled on the first turn. When the computer returns to this line, the variable D is set to one. This indicates a successful double.

Line 570 places the next card into the CARD variable. The CD variable is incremented by one so it is pointing to the next card, and the card just taken is placed into this player's array. The computer uses the subroutine at line 1000 to print the card in the correct position on the screen. The variable P1 is increased by 1. This is the next position in the player's hand for the next card. The subroutine in line 1210 is used to total the value of the hand.

Line 575 sends the computer to line 600 if the value of D is one. This player doubled his bet is allowed only one card for a total of three on a double.

Line 580 checks the value of TC. If it is greater than 21, and the values of T1 are 0 or greater than 21, the player broke and the computer is sent to line 620. The value of the player's hand is stored in the zero element of the PLAY array. The variable T1 contains the alternate value of the hand should the hand contain an ace.

Line 590 sends the computer to line 510 for another entry.

Line 600 checks the values of the variables that contain the total value of the hand. If the value of TC is less than 22 and greater than the alternate value of the hand, or if it is – 1, indicating blackjack, then this value is placed in the zero element of the array and the computer is sent to line 620.

Line 610 places the alternate value of the hand into the zero element of the array. However, if the value of T1 is 0 or the value of T1 is greater than 0, the value of TC is placed in this element of the array.

Line 620 erases the player's entry on the screen and resets the variables for the next player. The loop continues for the other player's turn. After both players have taken a turn, the computer is sent to line 700 for the dealer's turn.

Line 630 is used when a player is dealt a blackjack. The computer returns to line 540 after it changes the screen colors and makes the winning sounds.

Line 640 begins the subroutine that checks to see if the player can double down. First the computer checks to see if there is enough money to double the bet. If not, the POP command removes the return address from the stack and the computer does not return to line 560. Instead it goes to line 510 to get another entry.

Line 650 checks how many cards have been dealt to this player. The player can choose the double down option only before receiving a third card. Once he has three or more cards, the option cannot be chosen. If the value of P1 is greater than three, the POP command is used again to remove the return address from the stack. The computer is sent to line 510 for another entry.

Line 660 removes the amount of the bet from the first element of this player's array. The amount of the original bet is doubled, and the amount of money left is converted into B$. The money the player had is erased from the screen.

Line 670 prints the amount of money the player has left in the last row of the screen. The amount of the new bet is placed into

B$ and the old bet is removed from the screen. Line 680 prints the amount of the doubled bet on the screen, then returns to line 560.

Line 700 begins the dealer's turn and the end of the game. The values of both players' hands are checked. If both are over 21, the dealer does not deal to himself and the computer is sent to line 810 to end the game.

Line 710 clears the message from the top row of the screen. The value card that was face down is placed into CARD and the computer uses the subroutine at line 1000 to print this card on the screen. The value of the dealer's hand is calculated in the subroutine at line 1210. If the value of the hand is greater than 16, the computer is sent to line 760.

Line 720 takes the next card from the deck and places it in the computer's hand. This value of this card is placed into variable CARD and variable CD is increased by one. Variable Q counts how many cards have been printed on this line (only six cards can be printed on one line). The position of the card is calculated if this is not the seventh card the computer has drawn.

Line 730 calculates the position the card will be printed if the variable Q is greater than 6.

Line 740 uses the subroutine in line 1000 to print the card on the screen, and the subroutine in line 1210 to get the total value of the dealer's hand. This value is compared to 16. If the dealer's hand is greater than 16, the computer is sent to line 760. The dealer stands on a hard 17.

Line 750 sends the computer to line 720 to draw another card.

Line 760 checks the values of variables T1, TC, and T2. If any of these values equal 21, the house has 21 and wins. This value is stored in the computer's array and the computer is sent to line 810.

Line 770 checks whether the alternate value of the hand is less than 17. If so, the computer is sent to line 720 to draw another card. Variable T1 must contain some value; if it is 0, the dealer does not have an ace, and does not have an alternate value.

Line 780 checks the third possible value the hand could have. If this value is less than 17 and not 0 the computer will go back to line 720 and draw again.

Line 790 places the value of TC in the computer's array. It has to check the alternate value this hand could have. If the value of T2 is not 0 and is less than 22, this value is placed in the computer's array.

Line 795 checks to see if the value of TC is greater than the

value of T1 and less than 22. If so, this is the value placed into the computer's array.

Line 800 compares the values of T1 against T2. If they are greater and less than 22, this value is used. The highest value for the dealer's hand under 22 is used as the hand's value.

Line 810 begins the routine that checks the values of the player's hands. If the value of a player's hand is – 1, the player has blackjack. This is printed on the screen, and the value of the bet is multiplied by 2.5. The computer is sent to line 860 to add up the winnings.

Line 820 checks to see if the player went over 21. If he did, the bet value is set to 0 and the computer is sent to line 860.

Line 830 compares the player's hand against the computer's. If the player's hand is worth more than the dealer's, the player's bet is doubled and the computer is sent to line 860.

Line 840 checks to see if the dealer's hand is less than 22 and the player's hand is less than the dealer's. If this is true, the player loses and the bet is lost.

Line 850 checks to see if the dealer went over 21 and the player is under 22. If this is true, the player wins and the bet is doubled.

Line 860 adds the amount of money the player has to the bet value and stores this amount in the first element of the player's array. The value of the player's hand is cleared from the array and the bet and player's money is cleared from the screen.

Line 870 places the amount of money the player has into B$. This amount is then printed in the last row on the screen. The loop continues for the next player.

Line 880 clears the message from the top row of the screen and checks to see if either player has more than $1000. If either player has, the computer goes to line 950 to end the game.

Line 890 checks to see if both players are out of money. If both are, the house wins and the computer goes to line 980 to end the game.

Line 900 asks the players if they want to play again. The subroutine in line 1100 gets the entry. Press Y to play again, N to quit.

Line 910 ends the program if the N key is pressed.

Line 920 checks to see if more than 30 cards have been dealt. If so, the computer is directed to line 180 to shuffle the cards again.

Line 930 sends the computer to line 190 to play another hand.

Line 950 ends the game when one of the two players amass more than $1000. A message is printed on the screen.

Line 960 sets variable PL to two, then checks to see if player one has more money. If he does, the value of PL is changed to 1.

Line 970 prints the winning player's number on the screen and declares him a winner.

Line 980 loops until the Start key is pressed; line 990 runs the program again. To quit the program at this point, press the System Reset key.

Lines 1000-1040 print the cards on the screen. The value of the card must be stored in CARD before the computer uses this routine. The face values of the cards are taken from CARD$, the suit characters from SUIT$. If the card value is 10, the computer will print zero after the one is printed on the screen. The computer returns to the main program after the card is printed.

Lines 1050-1080 place the cards into the C array and shuffle them. The deck is shuffled five times. The card values are determined by the whole number, the suit by the decimal value added to the card value. The cards are shuffled in the FOR-NEXT loop that counts backwards. A card is chosen at random from the cards available, and its value is stored in a temporary location. The last card available in the deck is moved to this card's position. The card in the temporary location is moved to the last available location. Each time the loop is executed the number of cards in the deck is smaller by one than the previous time. This way all the cards are moved at least once in the shuffle.

Lines 1100-1200 contain the input routine. The variable B points to the location in B$ where the input will be placed. B$ is set to a null string and the keyboard is opened. The computer waits at line 1110 until a key is pressed. The value of this key is compared to 155. This is the Return key; if it is pressed and there is something in B$, the keyboard is closed and the computer returns to the main program. The computer cannot return to the main program until a key is pressed.

The value of K is also compared to 127. If it is greater than 127, the inverse key was pressed. The computer subtracts 128 from this value and POKES location 694 with a zero to reset the flag to normal. If the value of the variable CS is 1 and the value of K is 126, the delete key will function and the entry will be erased from the screen and from B$. If the value of K is greater than 95, the CAPS key was pressed and the key was entered as lowercase. To correct this 96 is subtracted from the value of K giving it the uppercase value, and the location 702 is POKEd with 64 to restore it to uppercase.

If the value of CS is 1, the computer is looking for a number and only the number keys will be considered a valid entry. If the value of CS is 0, the computer will only allow the keys H, S, B, and D to be entered. When the value of CS is 2, only the keys Y and N are considered. The key that has been pressed is printed on the screen if it was a valid entry. The computer remains in this routine until a valid entry and the Return key is pressed.

Lines 1210-1280 count the value of the cards in the hands. All the variables are cleared. The variable X indicates which player's hand is being tallied. The computer counts the first 13 cards. It is nearly impossible to have more than that number in this game. The value of the card is moved to variable I. If this value is 0, there are no more cards in this hand. The computer is sent to line 1270.

Line 1220 checks if the value of this card is greater than 9. If so, it is worth 10 and this amount is added to the hand and alternate hand. The computer is sent to line 1250 to check how many aces are in this hand.

Line 1230 checks to see if this card is an ace. If it is, this card is counted as 11 for the first hand value and 1 for the second or alternate hand value. Variable A is increased by one to count the number of aces in this hand. The computer is sent to line 1250 to check whether it should split the hand again.

Line 1240 adds the value of the card to both hand values.

Line 1260 continues the loop.

Line 1270 checks the value of the variables X, TC, and V. If the value of X is greater than 0, this is the player's hand and not the dealer's. If the value of TC is 21, this could be a blackjack. If the value of V is 3, only two cards have been dealt and this is a blackjack. The value of TC is changed to – 1 to indicate a blackjack. If we left it as 21, we wouldn't be able to tell at the end of the game if this was a blackjack or 21 made with several cards.

Line 1280 sets the variable V to 13 so the computer will end the loop and return to the main program.

Chapter 2

Grid Games

Many of the programs in this chapter are versions of traditional pencil and paper games. Others have been developed specifically for use on the computer. They all have one thing in common: a grid of some sort is used as the playing field. The programs *Boxes, Battleships*, and *Treasure Hunt* use a square grid with X and Y coordinates. *Boxes* is designed for young children; you don't even have to name the locations you use. *Battleships* and *Treasure Hunt* require you to name each location you are going to use.

Hex is played on a grid with six-sided locations. The grid itself is a diamond. It takes a little more concentration to name the points in this game.

In *Cave Dwellers* you are traveling through the grid and never actually see it. It is a decahedron, or ten-sided figure. Each room can lead to three others. You can find yourself going around in endless circles.

BOXES

Objective of the game: To complete more boxes than your opponent.

Directions: This program is designed for two players. One player is blue and the other green. Each player takes turns at the keyboard moving the cursor from one dot to the next. The arrow keys move the cursor. When the cursor is over the dot you want to claim, press the space bar. To start the turn again, press the

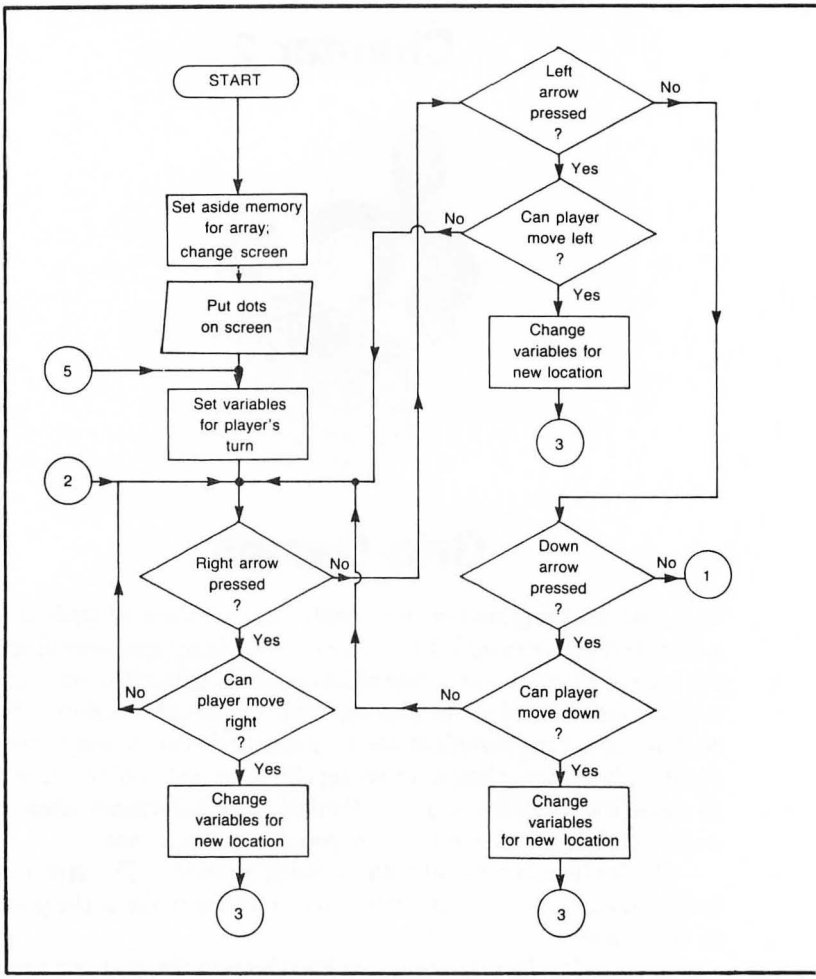
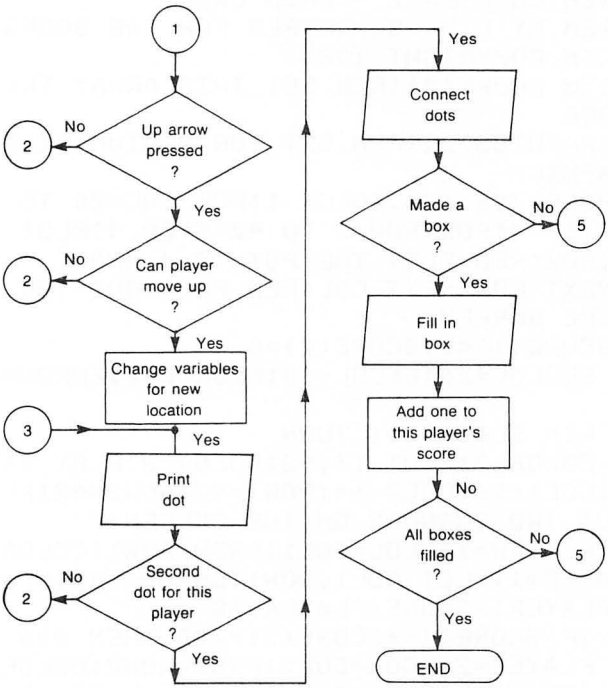


Fig. 2-1. Flowchart for Boxes.

Escape key. Once you have claimed one point, you may continue to claim a neighboring point. When two neighboring points have been claimed, the computer will connect the points. If you complete a box by connecting points, the box will be colored in your color. The game ends when all the boxes have been colored. Figure 2-1 is the flowchart for this program, and Listing 2-1 is the code.

Line 50 sets aside the memory needed for the SCORE array. This array keeps track of how many boxes each player has completed.

Line 60 sets the screen for graphics 21. This is graphics mode



5 with no text window. The dots are medium-sized in this mode.

Line 70 changes the green color to a darker shade. The color value is set to one and and the computer prints a grid of dots on the screen. The dots are placed four rows and columns apart. The ROW variable indicates the row value and the COL variable indicates the column the dot will be plotted in.

Line 80 continues the loop until all the dots are placed on the screen.

Line 90 clears the array that counts how many squares have been colored.

Listing 2-1. Boxes.

```
10 REM BOXES - FOR 2 PLAYERS - WHO CAN
 CONNECT THE MOST BOXES
20 REM CHAPTER 2 - GRID GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM SCORE(2):REM SET THIS ARRAY THE
 SCORE
60 GRAPHICS 21:REM SET FOR MEDIUM SIZE
 GRAPHICS
70 POKE 709,196:COLOR 1:FOR COL=20 TO
60 STEP 4:FOR ROW=2 TO 42 STEP 4:PLOT
COL,ROW:REM PLOT THE POINTS 4 APART
80 NEXT ROW:NEXT COL:REM PLOT OUT THE
ENTIRE SCREEN
90 SCORE(1)=0:SCORE(2)=0
100 BEGCOL=20:ENCOL=60:BEGROW=2:ENROW=
42
110 REM PLAYER #1 TURN
120 COLOR 2:PLOT 16,42:COLOR 3:PLOT 64
,42:COL1=16:COL2=64:ROW1=42:ROW2=42:RE
M PUT TWO CURSORS ON THE SCREEN
130 PLAYER=1:PCOL=COL1:PROW=ROW1:COLOR
 PLAYER+1:PLOT COL1,ROW1:GOSUB 300:SCO
RE(PLAYER)=SCORE(PLAYER)+S
135 IF SCORE(1)+SCORE(2)=100 THEN 900
140 PLAYER=2:PCOL=COL2:PROW=ROW2:COLOR
 PLAYER+1:PLOT COL2,ROW2:GOSUB 300:SCO
RE(PLAYER)=SCORE(PLAYER)+S
145 IF SCORE(1)+SCORE(2)=100 THEN 900
150 GOTO 130
300 OLDCOL=PCOL:OLDROW=PROW:S=0
310 OPEN #2,4,0,"K:":GET #2,KEY:CLOSE
#2:IF OLDCOL=COL1 OR OLDCOL=COL2 THEN
340
320 IF KEY=45 THEN PROW=PROW-4:GOTO 38
0
330 IF KEY=61 THEN PROW=PROW+4:GOTO 38
0
340 IF KEY=43 AND OLDCOL<>COL1 THEN PC
OL=PCOL-4:GOTO 380
350 IF KEY=42 AND OLDCOL<>COL2 THEN PC
OL=PCOL+4:GOTO 380
360 IF KEY=32 AND OLDCOL<>COL1 AND OLD
COL<>COL2 THEN 420
```

```
365 IF KEY=27 THEN COLOR 1:PLOT PCOL,P
ROW:POP :GOTO 120+PLAYER*10
370 GOTO 310
380 IF PROW<BEGROW THEN PROW=BEGROW
390 IF PROW>ENROW THEN PROW=ENROW
400 IF PCOL<BEGCOL THEN PCOL=BEGCOL
410 IF PCOL>ENCOL THEN PCOL=ENCOL
420 IF OLDCOL=COL1 OR OLDCOL=COL2 THEN
 COLOR 0:PLOT OLDCOL,OLDROW:GOTO 440
430 COLOR 1:PLOT OLDCOL,OLDROW
440 COLOR PLAYER+1:PLOT PCOL,PROW
450 IF KEY<>32 THEN 300
460 OLDCOL=PCOL:OLDROW=PROW
470 OLCOL=PCOL:OLROW=PROW:OPEN #2,4,0,
"K:":GET #2,KEY:CLOSE #2
480 IF KEY=45 THEN PROW=PROW-4:GOTO 54
0
490 IF KEY=61 THEN PROW=PROW+4:GOTO 54
0
500 IF KEY=43 THEN PCOL=PCOL-4:GOTO 54
0
510 IF KEY=42 THEN PCOL=PCOL+4:GOTO 54
0
515 IF OLDCOL=PCOL AND OLDROW=PROW THE
N 525
520 IF KEY=32 THEN 590
525 IF KEY=27 THEN COLOR 1:PLOT PCOL,P
ROW:PLOT OLDCOL,OLDROW:POP :GOTO 120+P
LAYER*10
530 GOTO 470
540 IF PROW<BEGROW THEN PROW=BEGROW
550 IF PROW>ENROW THEN PROW=ENROW
560 IF PCOL<BEGCOL THEN PCOL=BEGCOL
570 IF PCOL>ENCOL THEN PCOL=ENCOL
580 IF ABS(PCOL-OLDCOL)>4 OR ABS(PROW-
OLDROW)>4 THEN PCOL=OLCOL:PROW=OLROW:G
OTO 470
585 REM WATCH FOR THE DIAGONAL
590 IF PCOL<>OLDCOL AND PROW<>OLDROW T
HEN PCOL=OLCOL:PROW=OLROW:GOTO 470
595 REM IF BACK IN ORIGINAL POSITION,
DON'T WORRY
600 IF PCOL=OLDCOL AND PROW=OLDROW THE
N 620
610 LOCATE OLDCOL-SGN(OLDCOL-PCOL),OLD
ROW-SGN(OLDROW-PROW),TAKEN:IF TAKEN<>0
```

```
 THEN PCOL=OLCOL:PROW=OLROW:GOTO 470
620 COLOR 1:PLOT OLCOL,OLROW:COLOR PLA
YER+1:PLOT OLDCOL,OLDROW
630 COLOR PLAYER+1:PLOT PCOL,PROW
640 IF KEY<>32 THEN 470
650 COLOR PLAYER+1:PLOT OLDCOL,OLDROW:
DRAWTO PCOL,PROW:COLOR 1:PLOT OLDCOL,O
LDROW:PLOT PCOL,PROW
660 IF OLDCOL<>PCOL THEN 760:REM LINE
WENT ACROSS
670 LOCATE OLDCOL+1,OLDROW,TAKEN:IF TA
KEN=0 THEN 710:REM NO BOX THIS WAY
680 LOCATE PCOL+1,PROW,TAKEN:IF TAKEN=
0 THEN 710:REM NO BOX HERE
690 LOCATE PCOL+4,OLDROW-SGN(OLDROW-PR
OW),TAKEN:IF TAKEN=0 THEN 710:REM NOT
A BOX
700 COLOR PLAYER+1:FOR X=1*SGN(PROW-OL
DROW) TO 3*SGN(PROW-OLDROW) STEP SGN(P
ROW-OLDROW):PLOT OLDCOL,OLDROW+X
705 DRAWTO OLDCOL+3,OLDROW+X:NEXT X:S=
S+1
710 LOCATE OLDCOL-1,OLDROW,TAKEN:IF TA
KEN=0 THEN RETURN :REM NO BOX THIS WAY

720 LOCATE PCOL-1,PROW,TAKEN:IF TAKEN=
0 THEN RETURN :REM NO BOX HERE
730 LOCATE PCOL-4,OLDROW-SGN(OLDROW-PR
OW),TAKEN:IF TAKEN=0 THEN RETURN :REM
NOT A BOX
740 COLOR PLAYER+1:FOR X=1*SGN(PROW-OL
DROW) TO 3*SGN(PROW-OLDROW) STEP SGN(P
ROW-OLDROW):PLOT OLDCOL,OLDROW+X
750 DRAWTO OLDCOL-3,OLDROW+X:NEXT X:S=
S+1:RETURN
760 LOCATE OLDCOL,OLDROW+1,TAKEN:IF TA
KEN=0 THEN 800:REM NOT A BOX
770 LOCATE PCOL,PROW+1,TAKEN:IF TAKEN=
0 THEN 800:REM NOT HERE
780 LOCATE OLDCOL-SGN(OLDCOL-PCOL),PRO
W+4,TAKEN:IF TAKEN=0 THEN 800
790 COLOR PLAYER+1:FOR X=1*SGN(PCOL-OL
DCOL) TO 3*SGN(PCOL-OLDCOL) STEP SGN(P
COL-OLDCOL):PLOT OLDCOL+X,OLDROW
795 DRAWTO OLDCOL+X,OLDROW+3:NEXT X:S=
S+1
```

```
800 LOCATE OLDCOL,OLDROW-1,TAKEN:IF TA
KEN=0 THEN RETURN :REM NO BOX
810 LINE=810:LOCATE PCOL,PROW-1,TAKEN:
IF TAKEN=0 THEN RETURN
820 LOCATE OLDCOL-SGN(OLDCOL-PCOL),PRO
W-4,TAKEN:IF TAKEN=0 THEN RETURN
830 COLOR PLAYER+1:FOR X=1*SGN(PCOL-OL
DCOL) TO 3*SGN(PCOL-OLDCOL) STEP SGN(P
COL-OLDCOL):PLOT OLDCOL+X,OLDROW
840 DRAWTO OLDCOL+X,OLDROW-3:NEXT X:S=
S+1:RETURN
900 GRAPHICS 2:POSITION 2,4:? #6;"game
 over!"
910 POSITION 2,6:? #6;"GREEN - ";SCORE
(1)
920 POSITION 2,8:? #6;"BLUE - ";SCORE(
2)
925 REM NEXT MESSAGE INVERSE
930 POSITION 2,10:? #6;"PRESS start TO
 PLAY"
940 IF PEEK(53279)<>6 THEN 940
950 GOTO 60
```

Line 100 sets the variables that determine where the grid begins and ends on the screen.

Line 120 uses the second color, green, for the first player and the third color, blue, for the second player. The two colored cursors are placed on the screen. The column that each cursor is in is stored in variables COL1 and COL2. The value for the row for each cursor is stored in variables ROW1 and ROW2.

Line 130 begins the turn for the first player. The variable PLAYER is set to 1. This player's cursor values are moved to PCOL and PROW. The color value is set to the value of PLAYER + 1. The computer then uses the subroutine in line 300 to move the cursor. When the computer returns to this line, the score for this player is updated. Line 135 checks to see if all 100 boxes have been filled in. If they have, the computer goes to line 900 to end of the game.

Line 140 gives the second player his turn. The variables that hold the cursor position for this variable are placed into variables PCOL and PROW. The color value is changed for the second player and the computer uses the subroutine in line 300 to move the cursor. When the computer returns to this line, the score for the second player is updated.

Line 145 checks to see if all 100 boxes have been filled in. If they have, the computer goes to line 900 to end the game.

Line 150 sends the computer back to line 130 to continue the same.

Line 300 is the subroutine that moves the cursor on the screen. The values in PCOL and PROW are moved to variables OLDCOL and OLDROW. The S variable is set to 0. The value in this variable will change if a box is closed and colored in.

Line 310 opens the keyboard for a read. The computer waits until a key is pressed, then the keyboard is closed. The value of OLDCOL is compared to the values of COL1 and COL2. The cursor is not on the grid if either of these values are equal. If the cursor is not on the grid, the only way to move it is to the right or left, depending on whose turn it is. The computer is sent to line 340 and skips the lines that would allow the cursor to move up and down.

Line 320 checks to see if the up arrow key was pressed. If it was, the value of PROW is decreased by 4 and the computer goes to line 380 to print the cursor in the new position.The rows and columns are four pixels apart.

Line 330 checks to see if the down arrow was pressed. If it was, the variable PROW is increased by 4.

Line 340 checks the value of KEY to see if the left arrow key was pressed. This line also compares the value of OLDCOL to COL1. The cursor cannot move to the left if it is the first player's turn and the cursor is not on the grid. If the cursor *can* move to the left, the variable PCOL is decreased by 4.

Line 350 checks to see if the right arrow key has been pressed. If it has, and if this is not the first turn for the second player, the cursor will move one dot to the right. The value of the PCOL variable is increased by 4.

Line 360 checks to see if the space bar has been pressed. If it has and the cursor has moved, the computer goes to line 420 to change the color of the dot the cursor is on. If the variable OLDCOL is equal to COL1 or COL2 the cursor has not moved, so there is no dot to change.

Line 365 checks to see if the Escape key has been pressed. This is one way to change your mind about a dot you have captured. If this key has been pressed, the color value changes to 1, and the dot that was in the player's color is changed back to the grid color. The POP command removes the return address from the stack and

the computer is sent back to the beginning of the program line for this player.

Line 370 sends the computer back to line 310 for another key entry.

Line 380 checks the value of the PROW variable with the value of BEGROW. If the value of PROW is less than the value of BEGROW, PROW is reset. This keeps the cursor on the grid.

Line 390 checks to see if the new value in PROW would put the cursor off the grid on the bottom. If the value of PROW is greater than the value of ENROW the cursor would move off the grid, so PROW is reset to the bottom row of the grid.

Line 400 compares the value of PCOL with the value of BEGCOL. If PCOL is less than BEGCOL, the cursor is not on the grid. The value of PCOL is reset to BEGCOL.

Line 410 checks to see if the new value for PCOL would place the cursor off the right side of the grid. If PCOL is greater than EDCOL the cursor would be off the grid, so PCOL is reset to keep it on the grid.

Line 420 checks to see if the value of OLDCOL is the same as COL1 or COL2. If so, the color value is set to 0 so the cursor will be erased from the screen when it is moved to the new position. The cursor is erased by plotting the old position and the computer is sent to line 440.

Line 430 colors the dot the cursor was over.

Line 440 changes the color back to the player's color. This is the color the dot in the new location will be changed to. The PLOT command places the cursor on the screen in the new position.

Line 450 checks the value of the KEY variable again. If it is not 32, the space bar, the computer goes back to line 300 to move the cursor again.

Line 460 stores the current position of the cursor in variables OLDCOL and OLDROW.

Line 470 stores the values again in variables OLCOL and OLROW. The keyboard is opened again and the computer waits for a key to be pressed. The value of the pressed key is stored in the KEY variable and the keyboard is closed.

Lines 480-510 are similar to lines 320-350. This time the computer does not have to check if the cursor can move in a direction because the cursor is already in the grid. The computer will have to make sure that the cursor stays on the grid.

Line 515 checks to see if the cursor has moved; if it hasn't,

it must skip the next program line because you cannot claim the same dot twice in the same turn.

Line 520 checks to see if the space bar has been pressed. If it has, the computer is directed to line 590 to change the color of the dot and draw the line.

Line 525 checks to see if the Escape key has been pressed. If it has, the color value is changed to one and the dot that has been changed and the dot the cursor is on are changed back to their original color. The POP command is used to remove the return address from the stack, and the computer is sent back to the program line that begins the turn for this player.

Line 530 sends the computer to line 470 to wait for another entry.

Lines 540-570 check the position of PROW and PCOL to make sure the cursor will not be printed off the grid. If any value is out of range, it is reset to the edge of the grid.

Line 580 now subtracts the new column value from the column value of the first cursor, and the new row value from the row value of the first cursor. If the difference between any two values is more than 4, the cursor cannot move and values of PROW and PCOL are reset with the values of OLROW and OLCOL. The computer is sent back to line 470 to wait for a new entry.

Line 590 checks to see if the new location is on the diagonal. If the column values and the row values are different. the cursor was not moved in a straight line. The values of PROW and PCOL are reset with the values of OLROW and OLCOL. If the cursor was moved in a straight line, the row values or the column values would be the same. The computer is sent back to line 470 to wait for another entry.

Line 600 checks to see if the cursor has been moved to its original position. This can happen the second time the cursor is moved; for example, on the first turn the first dot in the first row was captured. On the second turn the cursor was moved to the first row, second column. For one reason or another the player decided he would rather move to the first column, second row. The cursor would have to move back to the first column, first row before it could move up to the second row. The computer has to be able to determine whether this dot was already taken so that it couldn't be taken twice. If this dot was taken already, the computer skips the next line and goes directly to line 620.

Line 610 uses the LOCATE command to see if a line already exists where the player wants to make his line. By using the SGN

command and subtracting the new column and row positions from the old ones, the computer will point to the location just inside the box along the side where the line will made. If the pixel at this location is not a zero, then a line is there already and the values of PCOL and PROW are restored to the old values. The computer is sent to line 470 to wait for another turn.

Line 620 changes the color of the dot where the cursor was to the original color, then uses the player's color to plot the cursor in the original position.

Line 630 plots the cursor in the new position.

Line 640 checks again to see if the space bar has been pressed. If it hasn't, the computer is sent to line 470 to wait for another entry.

Line 650 draws a line from the first cursor to the second. Then the dots at the beginning and the end of the line are restored to their original color.

Line 660 checks the values of the column variables. If these variables are not the same, the line was drawn horizontally on the screen rather than vertically. The computer is directed to line 760.

Line 670 checks the pixel just to the right of the end dot of the line just drawn. If this pixel value is 0, there is no box drawn in that direction and the computer is directed to line 710.

Line 680 checks the pixel to the right of the other end dot on the line just drawn. If this pixel is a zero, there is no box in that direction.

Line 690 checks one more point to see if a box has been completed. If this point also contains a zero, then no box has been completed in this direction.

Lines 700-705 fill in the completed box. The computer uses the FOR-NEXT loop to draw in the lines that color the box. Once the box is completely colored the S variable is increased by 1. This variable keeps track of how many boxes have been filled in. It is possible to complete more than one box in one turn.

Lines 710-730 continue to look for a completed box based on the vertical line drawn on the screen.

Lines 740-750 color the completed box, increase the variable S by 1 and return to the main program.

Lines 760-780 begin to look for a box that has been completed with the horizontal line. If it cannot form a box, the computer is sent to line 800.

Lines 790-795 fill in the completed box. The S variable increased by 1 to keep count of how many boxes have been completed.

Lines 800-820 continue to look for a completed box. If no box can be found, the computer returns to the main program.

Lines 830-840 color completed box. The S variable is increased by 1 and the computer returns to the main program.

Lines 900-930 end the game. The game is over when all 100 boxes have been filled in. The computer has been keeping track of the number of boxes each player has completed. The number of boxes each player has colored in is printed on the screen.

Line 940 loops until the Start key is pressed.

Line 950 sends the computer back to line 60 to play again. To end the program, press the System Reset key.

BATTLESHIPS

Objective of the game: To destroy the enemy's ships before yours are destroyed.

Directions: You play this game against the computer. You have ten ships to position on your grid. Each ship can be placed horizontally or vertically. Some ships are only one square long, others are two, three, or four squares long. The ships are displayed under the grid. The blue ship is the one you are trying to place on the grid. Enter the letter and the number of the square where you want the ship to begin. Then press the up arrow to place the ship vertically on the screen, or the right arrow to place the ship horizontally. Press the Return key after the letter and number are entered to validate the location, the delete key to erase the entry. After you enter the placement direction, press the Return key to validate that location, the delete key to enter a new one. If the ship will not fit at that location because it would be off the grid or on top of another ship, the program will tell you and you will be asked to place the ship again. After all your ships are placed, the computer will put its ships on its grid.

To play the game you are asked to enter a letter and a number of a square. If you hit one of the computer's ships, the screen flashes and the computer makes a sound. The computer in turn tries to hit your ships. When it does, the screen flashes and the computer makes a sound. The hits and misses are marked on the grid so you know where you have scored. The game is over when one player has hit all the opponent's ships. Figures 2-2 and 2-3 is the flowchart for this program, and Listing 2-2 contains the program code.

Line 50 sets aside the memory needed for the strings and array. The array CMP is used by the computer to keep track of which

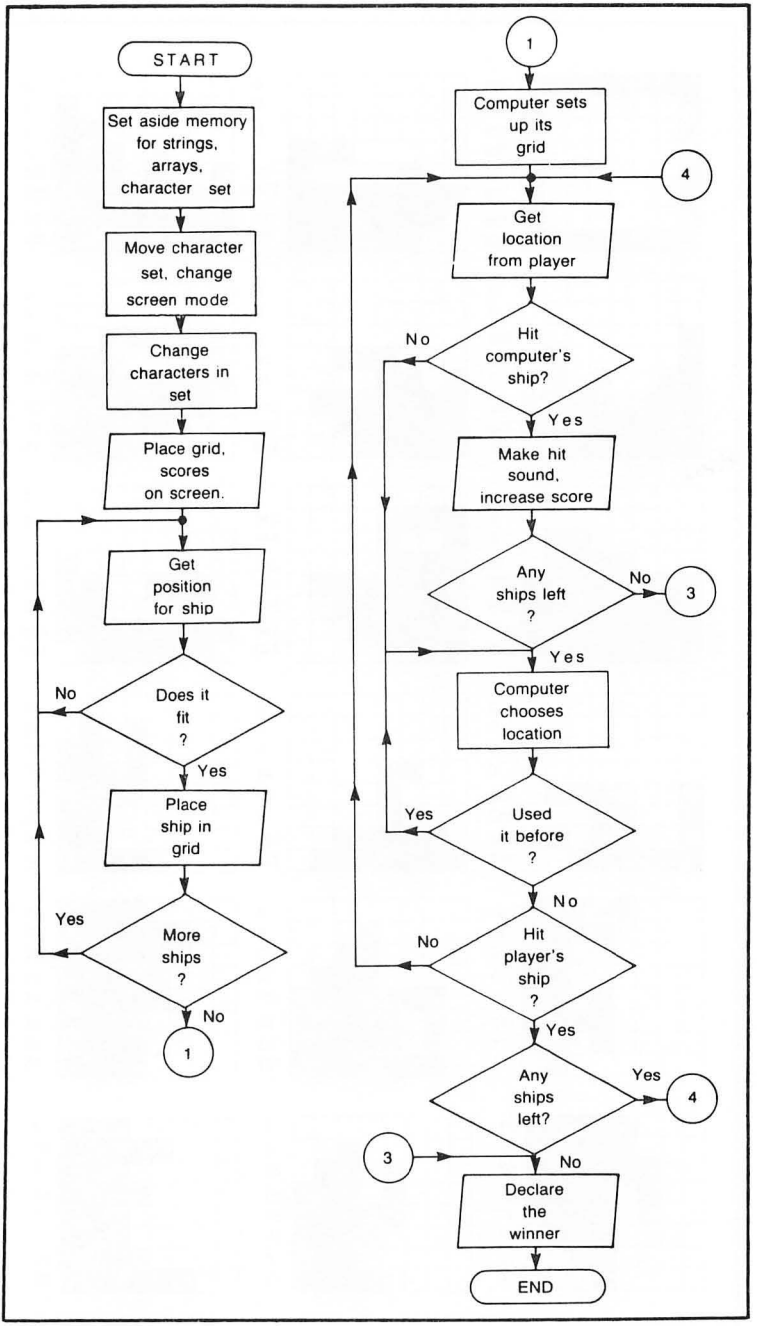


Fig. 2-2. Flowchart for Battleships.

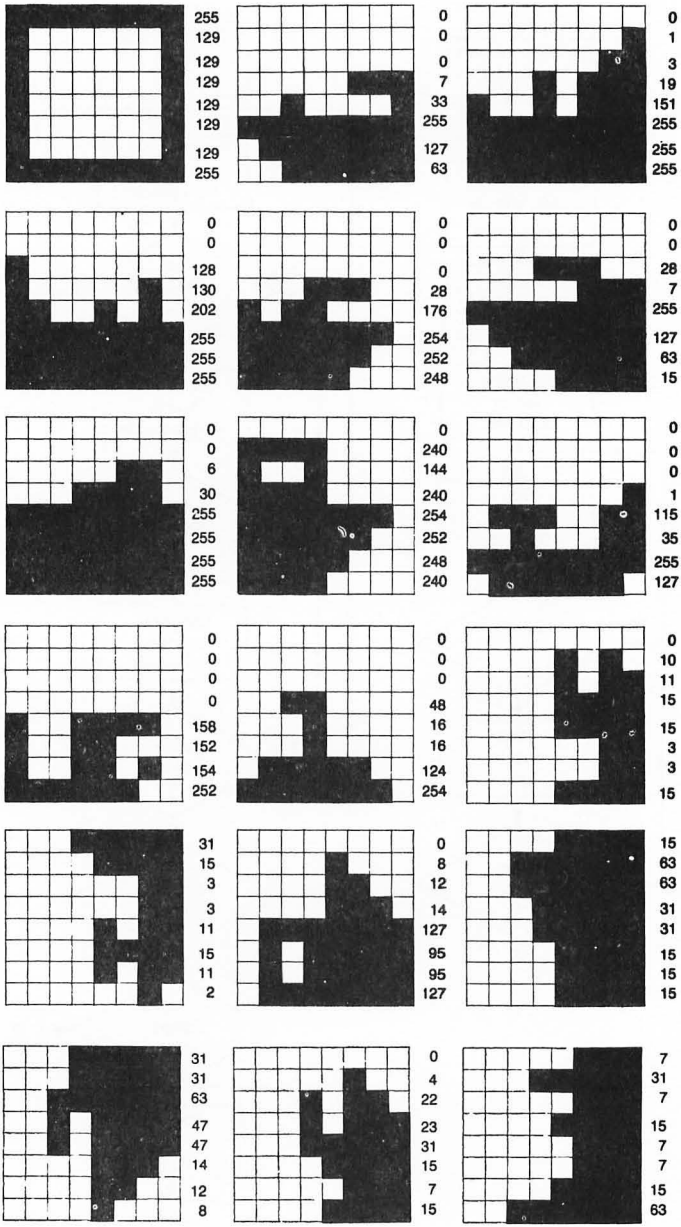
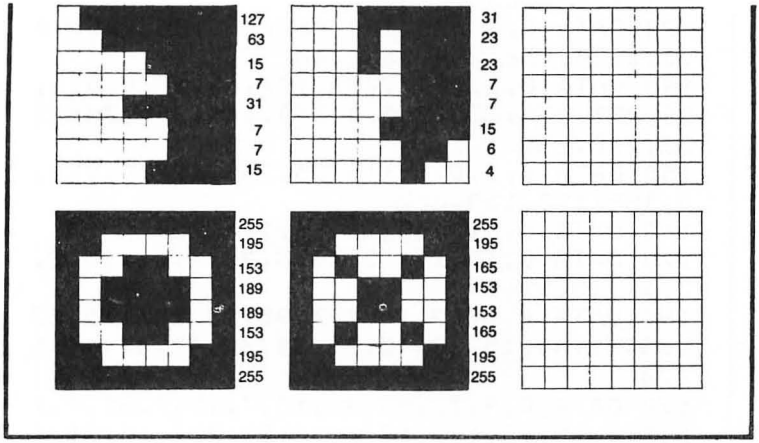


Fig. 2-3. Character set for Battleships.



Listing 2-2. Battleships.

```
10 REM BATTLE SHIPS FOR 2 PLAYERS - TR
Y TO DESTROY THE ENEMY'S SHIPS BEFORE
YOURS ARE DESTROYED
20 REM CHAPTER 2 - GRID GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM CMP(10,10),HG$(100),DIR$(6),SHI
P$(4),CG$(100)
55 MISS=1080:HIT=1070:KEYBOARD=900:TIM
E=1000:CLSCR=1010:SHIP=1020:PACK=1090
60 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM ROUTINE TO MOVE THE CHARACT
ER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
REM MOVE IT AND USE IT
100 CHARSET=A*256+24:FOR X=CHARSET TO
CHARSET+103:READ V:POKE X,V:NEXT X:FOR
 X=CHARSET+184 TO CHARSET+239
105 READ V:POKE X,V:NEXT X:FOR X=CHARS
ET+472 TO CHARSET+487:READ V:POKE X,V:
NEXT X
110 DATA 255,129,129,129,129,129,129,2
55
120 DATA 0,0,0,7,33,255,127,63,0,1,3,1
```

```
9,151,255,255,255
130 DATA 0,0,128,130,202,255,255,255,0
,0,0,28,176,254,252,248
140 DATA 0,0,28,7,255,127,63,15,0,0,6,
30,255,255,255,255,0,240,144,240,254,2
52,248,240
150 DATA 0,0,0,1,115,35,255,127,0,0,0,
0,158,152,154,252
160 DATA 0,0,0,48,16,16,124,254
170 DATA 0,10,11,15,15,3,3,15,31,15,3,
3,11,15,11,2
180 DATA 0,8,12,14,127,95,95,127,15,63
,63,31,31,15,15,15,31,31,63,47,47,14,1
2,8
190 DATA 0,4,22,23,31,15,7,15,7,31,7,1
5,7,7,15,63,127,63,15,7,31,7,7,15,31,2
3,23,7,7,15,6,4
195 DATA 255,195,153,189,189,153,195,2
55,255,195,165,153,153,165,195,255
200 POSITION 0,0:? #6;"computer      h
uman":POSITION 0,1:? #6;"  ":POSITION
16,1:? #6;"  "
205 REM ALPHABET AND POUND SIGNS ARE I
NVERSE
210 FOR X=12 TO 3 STEP -1:POSITION 4-(
X=12),X:? #6;X-2:NEXT X:POSITION 5,2:?
 #6;"abcdefghij"
220 FOR X=1 TO 10:POSITION 5,X+2:? #6;
"##########":NEXT X
230 SQ=3:SH=1:POSITION 3,14:? #6;"$%&'
 ()* +, -"
240 GOSUB CLSCR:POSITION 2,17:? #6;"en
ter letter            and number"
250 GOSUB KEYBOARD:IF KEY<65 OR KEY>74
 THEN 250:REM NOT A LETTER
260 POSITION 9,19:? #6;CHR$(KEY);"\":P
1=KEY-64
270 GOSUB KEYBOARD:IF KEY=126 THEN 240

275 IF KEY<49 OR KEY>57 THEN 270
280 POSITION 11,19:P2=KEY-48:? #6;CHR$
(KEY)
290 GOSUB KEYBOARD:IF KEY=155 THEN 320

295 IF KEY=126 THEN 240
300 IF P2=1 AND KEY=48 THEN P2=10:POSI
```

```
TION 12,19:? #6;"0"
310 GOTO 290:REM LOOP UNTIL RETURN KEY
 IS PRESSED
320 POSITION 2,21:? #6;"up or across":
REM LOWER CASE INVERSE
330 IF PEEK(764)=255 THEN 330:REM GET
THE DIRECTION
340 DIR=PEEK(764):POKE 764,255:IF DIR=
14 OR DIR=7 THEN 360
345 IF DIR=52 THEN 240
350 GOTO 330
360 DIR$="ACROSS":IF DIR=14 THEN DIR$=
"UP"
380 GOSUB SHIP:POSITION 8,22:? #6;DIR$
390 P=LEN(SHIP$):ON DIR/7 GOTO 400,430
400 IF P1+SQ>10 THEN 420
410 FOR X=P1+4 TO P1+4+SQ:LOCATE X,P2+
2,CH:IF CH=35 THEN NEXT X:POSITION P1+
4,P2+2:? #6;SHIP$:GOTO 460
420 GOTO 240
430 IF P2-SQ<1 THEN 420
440 FOR X=P2+2 TO P2+(2-SQ) STEP -1:LO
CATE P1+4,X,CH:IF CH<>35 THEN 240
450 NEXT X:FOR X=P2+2 TO P2+(2-SQ) STE
P -1:POSITION P1+4,X:? #6;SHIP$(P,P):P
=P-1:NEXT X
460 IF PEEK(764)=255 THEN 460
470 EN=PEEK(764):POKE 764,255:IF EN=12
 THEN 530
480 IF EN<>52 THEN 460
500 P=SQ+1:SHIP$="":FOR X=1 TO P:SHIP$
(X,X)="#":NEXT X:ON DIR/7 GOTO 510,520
510 POSITION P1+4,P2+2:? #6;SHIP$:GOTO
 240
520 FOR X=P2+2 TO P2+(2-SQ) STEP -1:PO
SITION P1+4,X:? #6;SHIP$(P,P):P=P-1:NE
XT X:GOTO 240
530 SH=SH+1:IF SH=2 OR SH=3 THEN POSIT
ION 3,14:? #6;"      ()*":SQ=2:GOTO 240
:REM CHARACTERS ARE INVERSE
540 IF SH>3 AND SH<7 THEN POSITION 8,1
4:? #6;"    +,":SQ=1:GOTO 240:REM CHAR
ACTERS ARE INVERSE
550 IF SH>6 AND SH<11 THEN POSITION 12
,14:? #6;"   -":SQ=0:GOTO 240:REM MINU
S SIGN IS INVERSE
```

```
560 GOSUB CLSCR:POSITION 14,14:? #6;"
 ":POSITION 2,17:? #6;"PLACING MY SHIP
S":SH=1:SQ=3:REM STARTING SHIPS
570 GOSUB PACK:FOR X=1 TO 10:FOR V=1 T
O 10:CMP(X,V)=0:NEXT V:NEXT X:REM CLEA
R THE ARRAYS
580 P1=INT(RND(1)*10)+1:P2=INT(RND(1)*
10)+1:DIR=INT(RND(1)*2)+1:ON DIR GOTO
590,620
590 IF P1+SQ>10 THEN 580:REM OUT OF SQ
UARES
600 FOR X=P1 TO P1+SQ:IF CMP(X,P2)=0 T
HEN NEXT X:FOR X=P1 TO P1+SQ:CMP(X,P2)
=SQ+1:NEXT X:GOTO 650:REM CHECK
610 GOTO 580:REM CAN'T PUT IT THERE
620 IF P2-SQ<1 THEN 580:REM OFF THE ED
GE
630 FOR X=P2 TO P2-SQ STEP -1:IF CMP(P
1,X)=0 THEN NEXT X:FOR X=P2 TO P2-SQ S
TEP -1:CMP(P1,X)=SQ+1:NEXT X:GOTO 650
640 GOTO 580:REM WON'T FIT
650 SH=SH+1:IF SH=2 OR SH=3 THEN SQ=2:
GOTO 580:REM PLACE THE NEXT TWO BOATS
660 IF SH>3 AND SH<7 THEN SQ=1:GOTO 58
0:REM AND THE NEXT THREE
670 IF SH>6 AND SH<11 THEN SQ=0:GOTO 5
80:REM AND THE LAST FOUR
680 CG$(1)="#":CG$(100)="#":CG$(2)=CG$
:REM MAKE THE GRID - POUND SIGN IS INV
ERSE
690 FOR X=1 TO 10:POSITION 5,X+2:? #6;
CG$((X-1)*10+1,10*X):NEXT X
710 REM MESSAGE IS LOWER CASE AND INVE
RSE
720 POSITION 2,17:? #6;"enter a letter
          and number":POSITION 9,19:?
 #6;"      "
730 GOSUB KEYBOARD:IF KEY<65 OR KEY>74
 THEN 730:REM NOT A LETTER
740 POSITION 9,19:? #6;CHR$(KEY);"\":P
1=KEY-64
750 GOSUB KEYBOARD:IF KEY=126 THEN 720
755 IF KEY<49 OR KEY>57 THEN 750
760 POSITION 11,19:P2=KEY-48:? #6;CHR$
(KEY)
770 GOSUB KEYBOARD:IF KEY=155 THEN 800
```

```
775 IF KEY=126 THEN 720
780 IF P2=1 AND KEY=48 THEN P2=10:POSI
TION 12,19:? #6;"0"
790 GOTO 770:REM LOOP UNTIL RETURN KEY
 IS PRESSED
800 IF CMP(P1,P2)>0 THEN GOSUB HIT:SHI
P$="_":CMP(P1,P2)=-1:GOTO 830:REM SHIP
$ IS INVERS UNDERLINE
810 IF CMP(P1,P2)<>0 THEN POSITION 9,1
8:? #6;"      ":GOTO 720
820 GOSUB MISS:SHIP$="^":CMP(P1,P2)=-1
:REM SHIP$ IS INVERSE UP ARROW
830 POSITION P1+4,P2+2:? #6;SHIP$:GOSU
B TIME:IF SHIP$="^" THEN GOSUB 1180:GO
TO 840:REM UP ARROW IS INVERSE
831 HMHT=HMHT+1:POSITION 16,1:? #6;HMH
T:IF HMHT=20 THEN 1120
832 GOTO 720
840 FOR X=1 TO 10:POSITION 5,X+2:? #6;
HG$((X-1)*10+1,X*10):NEXT X
841 REM MESSAGE IS INVERSE
845 POSITION 2,17:? #6;"HIT  ON
":POSITION 7,18:? #6;"POSITION   ":POS
ITION 9,19:? #6;"      "
847 IF CMHT=0 THEN 864
848 FOR X=1 TO 10:FOR V=1 TO 10:REM FI
ND A HIT
850 GOSUB 1170:IF CH<>95 THEN POSITION
 0,20:GOTO 860
852 IF V<10 THEN V=V+1:GOSUB 1170:V=V-
1:IF CH<>94 AND CH<>95 THEN V=V+1:GOTO
 862
854 IF V>1 THEN V=V-1:GOSUB 1170:V=V+1
:IF CH<>94 AND CH<>95 THEN V=V-1:GOTO
862
856 IF X<10 THEN X=X+1:GOSUB 1170:X=X-
1:IF CH<>94 AND CH<>95 THEN X=X+1:GOTO
 862
858 IF X>1 THEN X=X-1:GOSUB 1170:X=X+1
:IF CH<>94 AND CH<>95 THEN X=X-1:GOTO
862
860 NEXT V:NEXT X:GOTO 864
862 P1=V:P2=X:GOTO 865
864 P1=INT(RND(1)*10)+1:P2=INT(RND(1)*
10)+1
865 POSITION 9,19:? #6;"      ":LOCATE
```

```
P1+4,P2+2,CH:IF CH=95 OR CH=94 THEN 86
4
870 POSITION 9,19:? #6;CHR$(P1+64);"\"
;P2:IF CH=35 THEN GOSUB MISS:SHIP$="^"
:GOTO 890
880 GOSUB HIT:SHIP$="_":CMHT=CMHT+1
890 GOSUB TIME:POSITION P1+4,P2+2:? #6
;SHIP$
891 IF SHIP$="^" THEN GOSUB PACK:GOTO
690
892 IF SHIP$="_" THEN POSITION 0,1:? #
6;CMHT:IF CMHT=20 THEN 1110
895 GOTO 848
900 OPEN #2,4,0,"K:":REM READ THE KEYB
OARD
910 GET #2,KEY:REM GET THE KEY BEING P
RESSED
920 IF KEY=155 OR KEY=126 THEN 960
930 IF KEY>127 THEN KEY=KEY-128:POKE 6
94,0:REM SET FOR NORMAL VIDEO
940 IF KEY>95 THEN KEY=KEY-32:POKE 702
,64:REM SET FOR UPPER CASE
950 IF KEY<48 OR KEY>74 THEN 910:REM N
OT A GOOD KEY
960 CLOSE #2:RETURN
1000 FOR TME=1 TO 200:NEXT TME:RETURN
1010 FOR X=17 TO 22:POSITION 2,X:? #6;
"                    ":NEXT X:RETURN
1020 ON DIR/7 GOTO 1030,1050
1025 REM ALL THE CHARACTERS FOR THE SH
IPS ARE INVERSE
1030 GOTO 1030+SH
1031 SHIP$="$%&'":RETURN
1032 SHIP$="()*":RETURN
1033 SHIP$="()*":RETURN
1034 SHIP$="+,":RETURN
1035 SHIP$="+,":RETURN
1036 SHIP$="+,":RETURN
1037 SHIP$="-":RETURN
1038 SHIP$="-":RETURN
1039 SHIP$="-":RETURN
1040 SHIP$="-":RETURN
1050 GOTO 1050+SH
1051 SHIP$="=>?@":RETURN
1052 SHIP$=":;<":RETURN
1053 SHIP$=":;<":RETURN
```

```
1054 SHIP$="./":RETURN
1055 SHIP$="./":RETURN
1056 SHIP$="./":RETURN
1057 SHIP$="-":RETURN
1058 SHIP$="-":RETURN
1059 SHIP$="-":RETURN
1060 SHIP$="-":RETURN
1070 FOR X=1 TO 10:POKE 712,X*10:SOUND
 0,100,4,15:NEXT X:POKE 712,0:SOUND 0,
0,0,0
1075 RETURN
1080 FOR X=1 TO 10:SOUND 0,X*10,10,10:
NEXT X:SOUND 0,0,0,0:RETURN
1090 C=1:FOR X=1 TO 10:FOR V=1 TO 10:L
OCATE V+4,X+2,CH:HG$(C,C)=CHR$(CH):C=C
+1:NEXT V:NEXT X:REM PACK THE GRID
1100 RETURN
1110 POSITION 2,17:? #6;"I WIN!!!
                        ":GOTO 1130
1120 POSITION 2,17:? #6;"YOU WIN!!!!.
INVERSE                 "
1130 POSITION 9,19:? #6;"press start":
REM START IS INVERSE
1140 IF PEEK(53279)<>6 THEN 1140
1150 HMHT=0:CMHT=0:GOSUB CLSCR:GOTO 20
0
1170 LOCATE V+4,X+2,CH:RETURN
1180 C=1:FOR X=1 TO 10:FOR V=1 TO 10:L
OCATE V+4,X+2,CH:CG$(C,C)=CHR$(CH):C=C
+1:NEXT V:NEXT X:RETURN :REM PACK GRID
```

locations have hits and misses. HG$ contains the grid for the human player's ships. DIR$ holds the words for the up and down directions. SHIP$ holds the characters for the ships, and CG$ contains the grid for the computer's ships.

Line 55 sets several variables to line numbers used frequently in the program. This saves on memory in the program since numbers are stored as several bytes and variables are only one byte. MISS is the routine used when a ship is not at that point on the grid. HIT is the routine used when a ship is hit. KEYBOARD sets the key that was pressed. TIME is a timing loop. CLSCR clears the screen. SHIP prints the ship on the screen, and PACK takes the grid on the screen and makes it into one long string.

Line 60 finds out how much memory is in the computer by

PEEKing at location 106. In this program we subtract eight, or 1K, from this amount. As the screen only occupies 513 bytes, the character set can fit above the screen area. This value and the beginning address of the character set in ROM is stored in locations 204 and 206. This information will be used in the assembly language subroutine that moves the character set from ROM into RAM.

Line 70 contains the FOR-NEXT loop that moves the code for the assembly language subroutine from the data line into memory locations 1536-1555.

Line 80 contains the decimal codes for the assembly language subroutine that moves the character set from ROM into RAM.

Line 90 changes the screen to graphics 17; this is graphics one with no text window. The computer uses the USR command to execute the assembly language subroutine that begins in memory location 1536. When the computer returns to this line it POKES the address of the beginning of the character set in RAM into location 756.

Line 100 sets the variable CHARSET to the first byte of the third character in the character set. This is the first character we will change. In all we will change 24 characters in this program. The first FOR-NEXT loop changes 13 characters in the character set: all the characters from the pound sign (#) to the slash (/). We need numbers in this program,so we will leave the next 10 characters intact. The second FOR-NEXT loop changes the seven characters following the numbers.

Line 105 contains the third FOR-NEXT loop, changing four more characters. The reason the characters are changed in different parts of the character set is because we want to keep the numbers and the letters intact.

Lines 110-195 contain the codes for the new characters.

Line 200 prints the screen heading. The computer is playing against the human.

Line 210 prints the numbers and the letter for the grid. The grid is centered on the screen. The numbers are printed in rows 3-12, so the loop starts at 3 and ends with 12. We want the rows numbered from 1 to 10, and we want the numbers to line up properly. The Position command contains an equation as part of the column. The computer looks at the value of X and compares it to 12. If X is 12, the equation X = 12 is true and the equation is equal to −1. If X is not 12, the equation is false or 0. When the computer subtracts the value of X = 12 from four the numbers line up correctly. Four minus zero is four. The numbers from 1 to 9 are printed

in the fourth column. Four minus one is three, so the number 10 is printed in the third column. This is faster than using an IF-THEN statement to line up the numbers. The number that will be printed on the screen is two less than the value of X. Remember, X began at three and we want to label the rows from 1 to 10. After the numbers are printed, the computer prints the letters across the top of the grid.

Line 220 contains another FOR-NEXT loop. This loop prints 10 rows of pound signs on the screen. The pound sign has been changed into the grid character.

Line 230 sets the SQ variable to 3. This is one less than the number of squares the first ship will use. The second variable, SH, is the ship number that will be placed. The computer then prints a set of characters on the screen. The first four characters should be in inverse video. The rest of the characters are normal video. The characters are grouped four, three, two, and one. These are the four ships that will be placed on the grid.

Line 240 uses the subroutine in line 1010, the clear screen routine. The whole screen is not cleared, only the lines under the grid. The message that tells the player to enter a letter and a number is printed on the screen.

Line 250 uses the subroutine in line 900 to get an entry from the keyboard. If the value of the key entered is not between 65 and 74 the key was not a letter and the computer repeats this line. This line loops until a letter key is entered.

Line 260 prints the entered letter and a backslash. The value of KEY less 64 is stored in variable P1. We subtract 64 from the value of KEY because we want to convert the letter into a column number. The letter A has the value 65. It is the first column in the grid. It is easier to locate the grid of the letter value is changed into a number.

Line 270 uses the subroutine at line 900 again to get the next entry. This time we are looking for a number. If the value of the key pressed is 126, the Delete key was pressed and the computer goes to line 240 to erase the letter value that was entered and start again.

Line 275 checks to see if a number key was pressed. If it was not, the computer goes back to line 270 to get another key. The computer loops between these two lines until a number key or the Delete key is pressed.

Line 280 prints the number of the entered key on the screen. This time we subtract 48 from the value of KEY. The one key pro-

duces a value of 49. To get the actual key, we must subtract 48 from the key's value.

Line 290 uses the keyboard routine at 900 again. This time the computer is looking for a Return key. If the value of KEY is 155, the Return key was pressed and the computer goes to line 320 to place the ship.

Line 295 checks the value of KEY to see if the delete key was pressed. If it was, the value of KEY would be 126 and the computer would go to line 240 to get an entirely new entry.

Line 300 checks to see if a zero was entered. The only time the computer will accept 0 is if the last key entered was 1. If it was, the value of P2 is changed to 10 and the zero is printed on the screen.

Line 310 sends the computer back to line 290. The computer loops through these lines until the Return key or the delete key is pressed.

Line 320 sets the second part of the message. Now the computer needs to know how to place the ship on the screen, horizontally or vertically. A message is printed on the screen.

Line 330 looks at location 764 to see if a key has been pressed. If one has not, the value of 764 remains 255. This value changes once a key has been pressed.

Line 340 places the value of location 764 into the variable DIR. The location 764 is then changed back to 255. If the value of DIR is 7 or 14 an arrow key has been pressed and the computer is sent to line 360.

Line 345 checks to see if the value of DIR is 52. If it is, the Delete key has been pressed and the computer goes to line 240 to get a new entry.

Line 350 sends the computer back to line 330. The key that was pressed cannot be used in this routine. The computer loops through these lines until the up arrow, right arrow, or Delete key is pressed.

Line 360 places the word ACROSS into DIR$. Then it checks the value of DIR. If this value is 14, the computer changes the word in DIR$ to UP.

Line 380 sends the computer to the subroutine that begins at line 1020. If the correct ship is placed into SHIP$ and the computer prints the contents of DIR$ on the screen.

Line 390 finds the length of the ship and sends the computer to the correct routine depending on which direction the ship should be printed in.

Line 400 adds the value of P1, the column position, to the number of squares this ship will take up. If this sum is greater than 10 the ship will not fit on the grid. The computer is sent to line 240 to try again.

Line 410 places the ship on the grid. The computer adds 4 to the value of P1 because the first grid square is in the fifth column of the screen. Before the computer prints the ship on the screen it looks at that location on the screen to see if the square is empty. If it is, that location will contain a value of 35. If all the squares are empty the loop will be completed and the ship will be printed on the screen. The computer is sent to line 460 to continue placing ships on the screen.

Line 420 is executed if the FOR-NEXT loop in the previous line reaches a square that is not empty. The computer is sent to line 240 to clear the screen and start again.

Line 430 starts the routine that places a ship vertically on the grid. This time the computer subtracts the number of squares this ship will need from the value of the first square or row in which the ship will be printed. If the difference is less than one, the ship will not fit on the grid and the computer is sent to line 240 to get another entry.

Line 440 begins at the row value P2 + 2 and continues up the grid the number of squares this ship needs. To figure out the last square the computer should look at, the value of SQ is subtracted from 2; then this difference is added to P2. The computer looks at the grid to see if the square is empty. If it is, the variable CH will be 35. If CH is not 35, the computer is sent to line 240 to trv again.

Line 450 continues the loop. If the loop is completed the computer begins another loop to print the ship, one character at a time, on the screen. Since we are printing from bottom to top on the screen, each element of the string must be printed separately.

Line 460 looks at location 764 to see if a key has been pressed. This location will contain 255 until a key is pressed.

Line 470 stores the value of location 764 in the variable EN. This location is reset by POKEing it with 764. If the value of EN is 12, the Return key has been pressed and the computer goes to line 530 to continue placing ships on the screen.

Line 480 sends the computer back to line 460 for another key input if the value of EN is not 52.

Line 500 is executed if the Delete key was pressed. The Delete key has a value of 52. The bottom of the screen is cleared. The

value of P is one more than the value of SQ. P now contains the actual number of squares the ship comprised. The string SHIP$ is filled with pound signs. The computer is directed to the correct line, depending on which way the ship was printed on the screen.

Line 510 prints the cleared grid squares on the screen if the ship was printed horizontally. The computer is sent to line 240 to get a new position.

Line 520 prints the empty squares one at a time up the column on the grid. Once the ship is erased, the computer goes to line 240 for a new entry.

Line 530 clears the lower portion of the screen. The variable SH is increased by 1. This value is checked to see if it is 2 or 3. If it is, the first ship is erased and the characters for the second ship are printed in inverse video. The variable SQ is changed to 2 and the computer is sent to line 240 for a new entry.

Line 540 checks to see if SH is between 4 and 6. If it is, the second ship is erased and the characters for the third ship are printed on the screen in inverse video. The variable SQ is set to 1 and the computer is sent to line 240 for a new entry.

Line 550 checks to see if the value of SH is between 7 and 10. If it is, the third ship is erased from the screen and the fourth ship is printed in inverse video. The computer goes to line 240 to place these ships.

Line 560 clears the lower portion of the screen, erases the last ship and prints a message on the screen. The variables SH and SQ are reset.

Line 570 uses the subroutine in line 1090 to pack the grid characters into one string. When the computer returns to this line it clears the CMP array from any information that may be stored in it. The computer does not clear an array when the program is run.

Line 580 chooses three random numbers. The first number, which is placed in the variable P1, is the column position of the ship. The second number, placed in P2, is the row position, and the third number is the direction in which the ship will be placed on the screen. Based on the value of the variable DIR, the computer will go to either line 590 or line 620.

Line 590 is used if the value of DIR is 1. The ship will be placed horizontally if there is enough room on the grid and the ship will not run into another ship. The value of P1, the column position, is added to the value of SQ. If the sum is greater than 10, the ship will not fit on the grid and the computer goes back to line 580 to pick another set of numbers.

Line 600 checks each element of the CMP array to see if there is a number in the element. If there isn't, the loop continues until all the elements the ship would cover are checked. Next the computer places the value SQ + 1 into the elements of the array. This value is the ship number that is placed in the array. It is also the number of squares this ship occupies on the grid. After the ship is placed in the array, the computer is sent to line 650 for the next ship.

Line 610 sends the computer back to line 580 if the ship cannot be placed in this location on the grid.

Line 620 is used when the variable DIR is 2. The ship will be placed vertically on the grid. The number of squares the ship will take up is subtracted from the row value in P2. If this difference is less than 1, the ship will not fit on the grid and the computer is sent back to line 580 to pick a new location.

Line 630 checks each element of the CMP array that the ship will be positioned in to see if it is empty. If it is, the value of the element is 0. If the loop is completed, the value of SQ + 1 is placed in each element of the array that will contain the ship. The computer is sent to line 650 to continue the ship placement.

Line 640 sends the computer back to line 580. The ship will not fit in the chosen position because another ship is already there.

Line 650 checks the value of SH to see if it is 2 or 3. If it is, the value of SQ is changed to 2. The computer is sent to line 580 to place the new ship.

Line 660 checks to see if the value of SH is between 4 and 6. If it is, the value of SQ is 1. The computer goes to line 580 to place these ships.

Line 670 checks the value of SH to see if it is between 7 and 10. The value of SQ is 0 and the computer places the last four ships.

Line 680 clears the computer's grid string and fills every element of it with pound signs or the empty grid squares.

Line 690 prints this empty grid on the screen.The player is looking at the computer's grid. At this time it is empty, because the player has not made an attempt to hit the computer's ships. Once the player starts to enter possible positions of the computer's ships, the grid will contain marks that indicate a hit or a miss.

Line 720 prints a message on the screen. The player is asked to enter a letter and a number. This is the position of a possible ship.

Line 730 uses the keyboard subroutine to set the value of a key that was pressed. If the value of the variable KEY is not between the values of 65 and 74, the key pressed was not a letter. The com-

puter will loop at this line until a letter key is pressed.

Line 740 prints the value of the key on the screen. The backslash is printed after the letter. A value 64 less than the value of the key is stored in variable P1. This is the column value.

Line 750 uses the keyboard subroutine to set the next key value. If the value of the key pressed is 126, the Delete key has been pressed and the computer goes back to line 720 for another entry.

Line 755 checks to see if the value of KEY is a number value. The number values are between 48 and 57. This line does not want a key whose value is 48 because that is the 0.

Line 760 places the number value of the key in the variable P2 and prints this number on the screen.

Line 770 uses the keyboard subroutine to get another key entry. If the value of KEY is 155, the Return key has been pressed and the player is satisfied with the position. The computer is sent to line 800.

Line 775 sends the computer to line 720 if the value of KEY is 126, the Delete key.

Line 780 checks to see if the value of P2 is 1 and the value of KEY is 0. This is the only time the zero key is accepted. The value of P2 is changed to 10 and the zero is printed on the screen.

Line 790 sends the computer back to line 770. The computer will loop through these lines until the Return key is pressed.

Line 800 looks at the value of the CMP array at the position indicated by the variables P1 and P2. If the value of this element is greater than zero, the player scored a hit. The underline character is placed into SHIP$ and the element of the array is changed to a negative one. The computer is sent to line 830.

Line 810 erases the position the player entered if the value of the new positioning CMP is not a zero. This means that the player has already entered that position and it was a hit or a miss. Either way, the computer placed a -1 value in that location. The computer is sent back to line 720 to get a new location.

Line 820 sends the computer to the MISS subroutine. The contents of SHIP$ are changed and the contents of the element of the array are changed to a -1.

Line 830 prints the contents of SHIP$ on the screen at the location in the grid entered by the player. This can be the hit character or the miss character depending on whether a ship was hit. The timing subroutine is used, then if the location was a miss the computer uses the subroutine in line 1180 to pack the grid into one string. The computer uses this routine only after the player has

missed. The computer goes to line 840 for the computer's turn.

Line 831 adds one to the value of HMHT. This is the number of hits the player has. This value is printed under HUMAN on the screen. If the value of HMHT is 20, all the ships have been hit and the player wins. The computer is sent to line 1120.

Line 832 sends the computer to line 720 for another input. The player continues to play as long as he gets hits. When he misses, then it's the computer's turn.

Line 840 begins the computer's turn. The contents of HG$ are printed on the screen. This is the player's grid. The hits and misses the computer made are stored on this grid along with the player's ships.

Line 845 prints a message on the screen.

Line 847 checks the value of the variable CMHT. If it is a zero, the computer has not hit the player's ship and the computer is sent to line 864 to pick a random position in the grid.

Line 848 begins two FOR-NEXT loops. The computer will check each element of the player's grid to see where a hit was made. The computer is very honest. It only looks to see if that square was a hit. It does not look to see if there is a ship in that position.

Line 850 uses the subroutine in line 1170 to get the value of the grid square. If the value is not 95 (a hit), the computer goes to line 860 to continue the loop.

Line 852 checks to see if the value of V is less than 10. If it is, the computer checks the value of the square in the column after the hit. If this square is not a hit or a miss, the computer goes to line 862 to hit this location.

Line 854 checks to see if the value of V is greater than 1. If it is the column before this one can be checked. If this square is not a hit or a miss, the computer goes to line 862 and hits this location.

Line 856 now looks at the squares under the location that was hit. If this location is not a hit or a miss, the computer goes to line 862 and hits it.

Line 858 checks the squares above the one that has been hit. If this square does not contain a hit or a miss, the computer goes to line 862.

Line 860 continues the loops. If the computer cannot find a square to hit, the computer goes to line 864 to pick a random location.

Line 862 transfer the values of V and X to P1 and P2. The computer goes to line 865 to hit this location.

Line 864 chooses two random numbers as a position to hit.

Line 865 checks this position to make sure it does not contain a hit or a miss. If it does, the computer goes to line 864 and chooses another location.

Line 870 prints the position that will be hit on the screen. If the value of CH is 35, the location did not contain a ship and it is a miss. The computer is sent to the MISS subroutine, then places the miss character into SHIP$. The computer is sent to line 890 to print the character on the screen.

Line 880 uses the hit subroutine, then prints the hit character on the screen. The computer's score is increased by one.

Line 890 uses the timing subroutine. The hit or miss character is printed on the screen.

Line 891 sends the computer to the subroutine that packs the grid into a string if the computer missed the ship. The computer then goes to line 690 for the player's turn.

Line 892 prints the computer's new score on the screen. If the variable CMHT is equal to 20, the computer has hit all the player's ships and won the same. The computer is sent to line 1110 to end the same.

Line 895 sends the computer to line 848 to try to hit another ship.

Line 900 begins the keyboard routine. The keyboard is opened to be read.

Line 910 waits for a key to be entered. The value of the key is stored in the KEY variable.

Line 920 checks to see if the Return key or the Delete key was pressed. If it was, the computer goes to line 960.

Line 930 checks to see if the value of KEY is greater than 127. If it is, 128 is subtracted from the value of KEY and the flag is restored for normal video by POKEing location 694 with a zero.

Line 940 checks to see if the value of KEY is greater than 95. If it is, the CAPS key was pressed. The computer subtracts 32 from the value of KEY and POKEs location 702 with 64 to restore the computer to uppercase.

Line 950 checks to see if the key is a number or letter key. If it isn't the computer goes back to line 910 to set another key.

Line 960 closes the keyboard and sends the computer back to the main program.

Line 1000 is the timing loop. Change the value 200 to make the loop longer or shorter.

Line 1010 clears the bottom portion of the screen.

Lines 1020-1060 place the correct characters into SHIP$. The line that the computer uses depends on the value of DIR. If the value is 7, the first ten lines are used to place the characters that represent the ship into SHIP$. If the value of DIR is 14, the computer uses the second set of ten lines for the ship's characters.

Line 1070 makes the hit sound.

Line 1080 makes the miss sound.

Line 1090 uses the LOCATE command to take each element of the grid and place it in HG$. By storing it this way, it takes up less room than an array and is easily taken apart when it is printed on the screen.

Lines 1110-1130 are the end of the game. Which message will be printed on the screen depends on who reaches 20 points first.

Line 1140 loops until the Start key is pressed.

Line 1150 clears the variables that hold the scores, clears the bottom portion of the screen and goes to line 200 to play another game.

Line 1170 is the LOCATE command, used regularly to see what is on that location on the screen.

Line 1180 packs the computer's grid into CG$.

HEX

Objective of the game: To connect a series of hexagons from one end of the grid to the other before your opponent does.

Directions: The screen displays a diamond containing hexes. A cursor appears at the bottom of the screen. You can move this cursor by pressing the letter keys in the center of the keyboard. The Y key moves the cursor up, T moves the cursor up and left, U moves the cursor up and right. The N key moves the cursor down, B down and to the left, and the M key moves the cursor down and right. Press the H key to claim the hex location.

The game alternates between two players. One player is light purple, the other dark purple. The player number is displayed at the bottom of the screen under the cursor. When the player presses H to claim a square, the computer makes a sound and prints that player's character in that location. The second player now has a turn. Each location can only be occupied by one player. The first player to complete a path from one side of the grid to the other is the winner. The path must connect that player's borders. For example, if player one is light purple, his path must begin at the lower right side and end at the upper left side. Player Two, on the other hand, is dark purple: his path must begin at the lower left

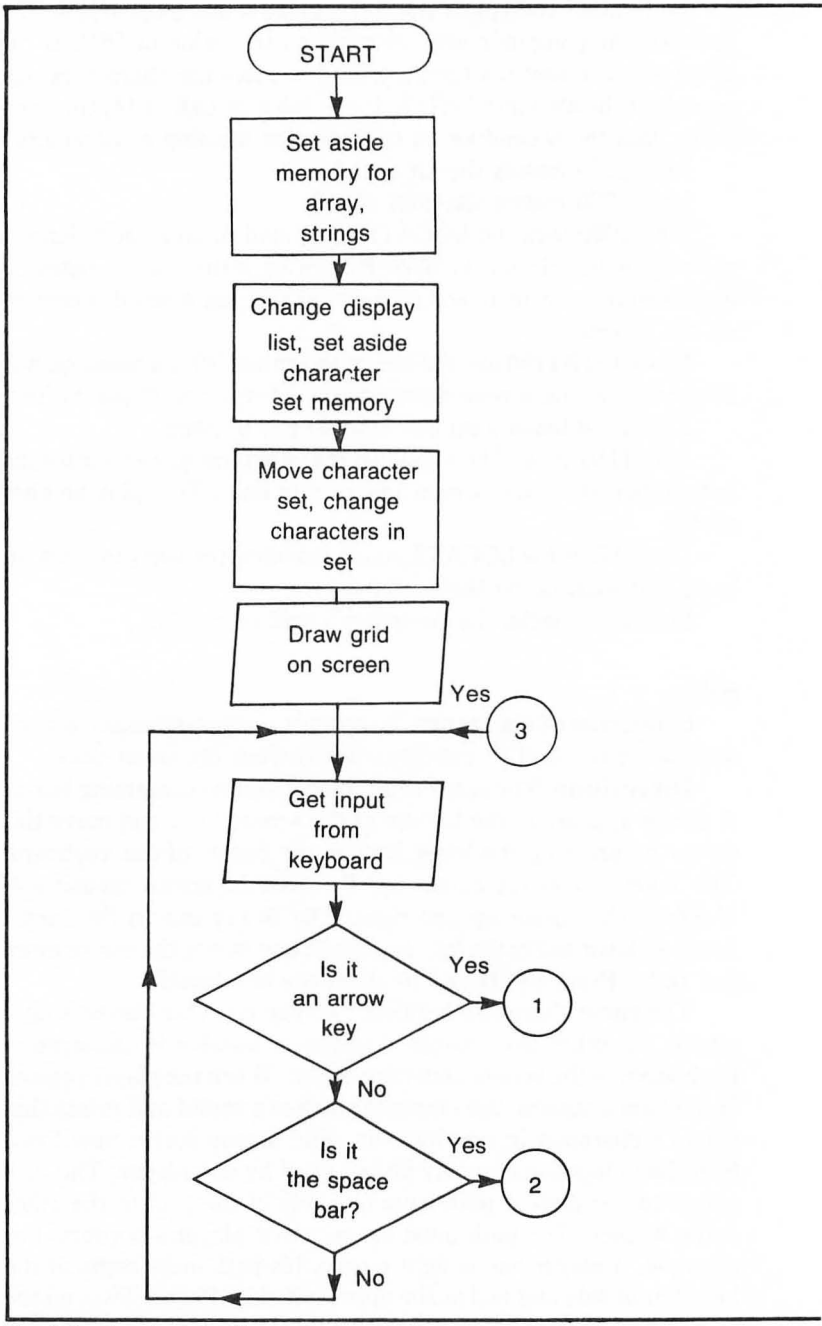
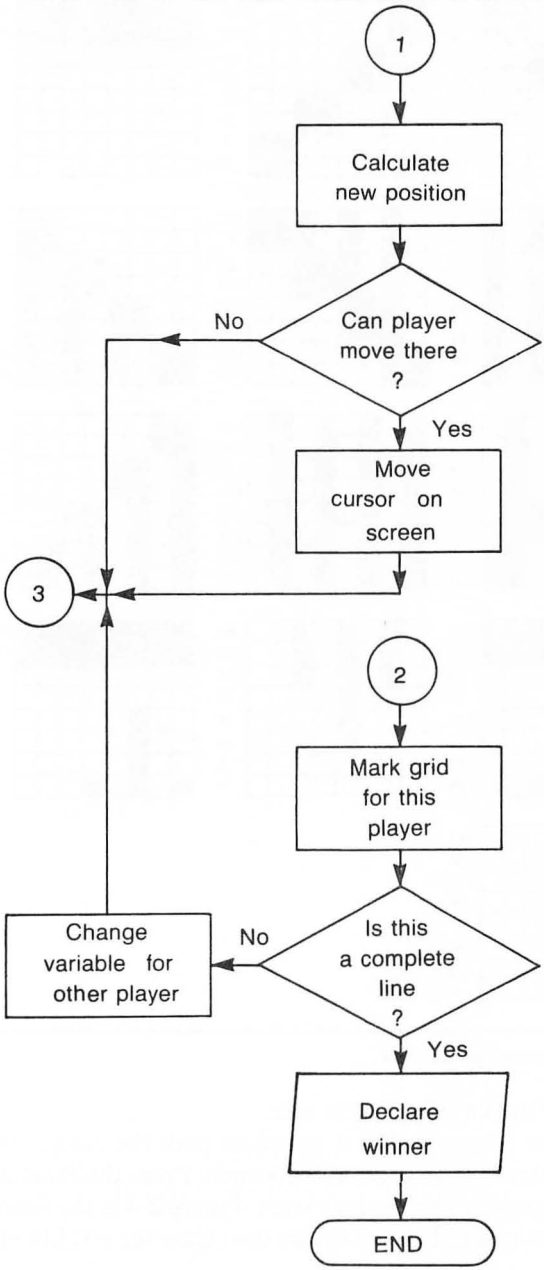


Fig. 2-4. Flowchart for Hex.


1
Calculate new position
No
Can player move there ?
Yes
Move cursor on screen
3
2
Mark grid for this player
Change variable for other player
No
Is this a complete line ?
Yes
Declare winner
END

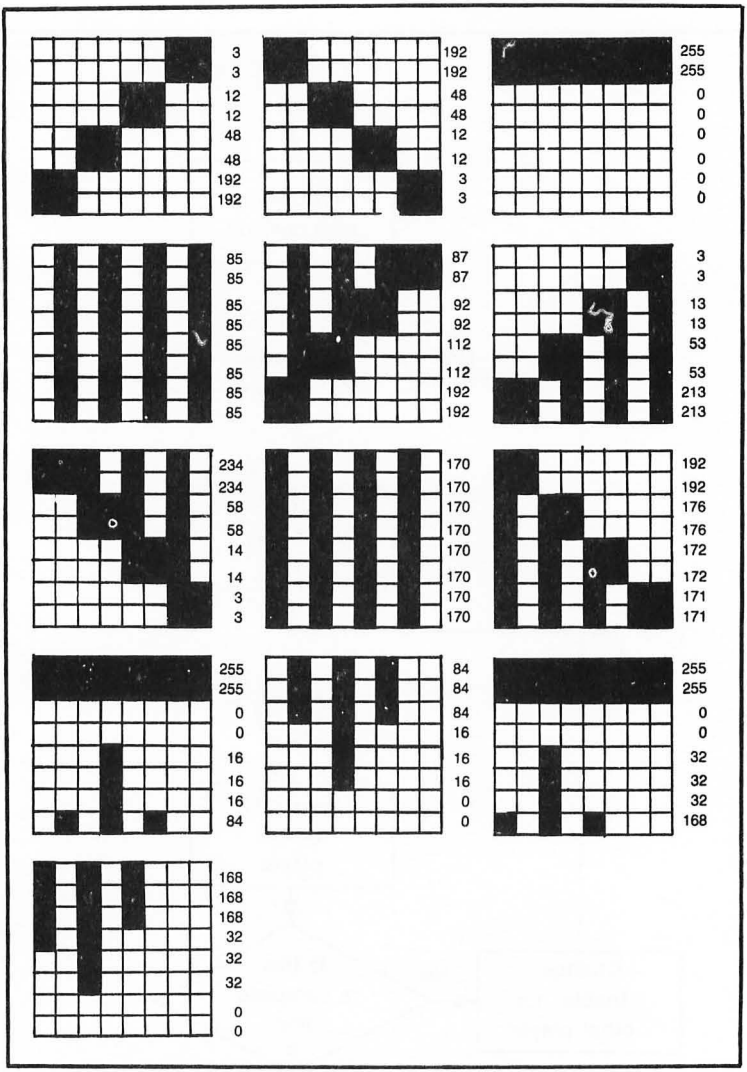


Fig. 2-5. Character set for Hex.

side and end at the upper right side.

When one player makes a complete path the computer will display the winning message on the screen. Press the Start key to play again. Press System Reset to quit. Figure 2-4 is the flowchart for this program, and Fig. 2-5 shows the character set. Listing 2-3 is the code.

Line 50 sets aside the memory needed for the string and ar-

Listing 2-3. Hex.

```
10 REM HEX - TRY TO MAKE A CONNECTING
LINE TO THE OTHER SIDE BEFORE YOUR OPP
ONENT
20 REM CHAPTER 2 - GRID GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM GRID(9,9),H$(4)
60 GRAPHICS 0:DLIST=PEEK(560)+PEEK(561
)*256:POKE DLIST+3,68:FOR X=6 TO 24:PO
KE DLIST+X,4:NEXT X
70 FOR X=25 TO 28:POKE DLIST+X,6:NEXT
X:REM CHANGE DISPLAY LIST TO ANTIC 4 A
ND GRAPHICS 2
80 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756):POKE 708,74:POKE 709,100:POKE
 710,38
90 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE CHARACTER SET
100 DATA 104,162,4,160,0,177,205,145,2
03,200,208,249,230,206,230,204,202,208
,242,96
110 Q=USR(1536):POKE 756,A:REM MOVE IT
 & USE IT
120 CHARSET=A*256+520:FOR X=CHARSET TO
 CHARSET+103:READ V:POKE X,V:NEXT X
130 DATA 3,3,12,12,48,48,192,192,192,1
92,48,48,12,12,3,3,255,255,0,0,0,0,0,0
,85,85,85,85,85,85,85,85
131 DATA 87,87,92,92,112,112,192,192,3
,3,13,13,53,53,213,213,234,234,58,58,1
4,14,3,3
132 DATA 170,170,170,170,170,170,170,1
70,192,192,176,176,172,172,171,171
133 DATA 255,255,0,0,16,16,16,84,84,84
,84,16,16,16,0,0
134 DATA 255,255,0,0,32,32,32,168,168,
168,168,32,32,32,0,0
140 RESTORE 140:FOR X=1 TO 4:READ V:H$
(X,X)=CHR$(V):NEXT X:DATA 1,3,2,32
150 R=0:G=1:FOR X=17 TO 1 STEP -2:G1=G
:POSITION X,R:? H$;:G1=G1-1:IF G1=0 TH
EN 170
160 ? H$;:G1=G1-1:IF G1<>0 THEN 160
170 R=R+1:G=G+1:NEXT X:G=G-3
```

```
180 FOR X=4 TO 2 STEP -1:H$(X,X)=H$(X-
1,X-1):NEXT X:H$(X,X)=" ":REM MOVE THE
 SHAPE
190 G1=G:POSITION X,R:? CHR$(2);H$;:G=
G1-1:IF G1=0 THEN 210
200 ? H$;:G1=G1-1:IF G1<>0 THEN 200
210 ? " ";CHR$(1):R=R+1:G=G+2:FOR X=1
TO 4:READ V:H$(X,X)=CHR$(V):NEXT X:DAT
A 3,2,32,1
220 FOR X=2 TO 16 STEP 2:G1=G:POSITION
 X,R:? H$;:G1=G1-1:IF G1=0 THEN 240
230 ? H$;:G1=G1-1:IF G1<>0 THEN 230
240 ? H$(1,1):R=R+1:G=G-1:NEXT X:POSIT
ION X,R:? H$(1,1):G=16
250 FOR X=0 TO G:POSITION X,G1:? CHR$(
4):NEXT X:POSITION X,G1:? CHR$(5):G=G-
2:G1=G1+1:IF G>-2 THEN 250
260 G=36
270 POSITION G-1,G1:? CHR$(6):FOR X=G
TO 38:POSITION X,G1:? CHR$(4);:NEXT X:
G=G-2:G1=G1+1:IF G1<>18 THEN 270
280 G=20:G1=0
290 POSITION G-1,G1:? CHR$(7):FOR X=G
TO 38:POSITION X,G1:? CHR$(8):NEXT X:G
=G+2:G1=G1+1:IF G<38 THEN 290
300 G=0:TL=50
310 FOR X=0 TO G:POSITION X,G1:? CHR$(
8);:NEXT X:? CHR$(9):G=G+2:G1=G1+1:IF
G1<>18 THEN 310
330 FOR X=0 TO 9:FOR V=0 TO 9:GRID(X,V
)=0:NEXT V:NEXT X:REM CLEAR GRID FOR N
EW GAME
340 SOUND 0,100,10,10:GOSUB 1000:SOUND
 0,75,10,10:GOSUB 1000:POSITION 4,21:?
"  PLAYER ONE ":PLAYER=10:GOSUB 1010
350 SOUND 0,150,10,10:GOSUB 1000:SOUND
 0,125,10,10:GOSUB 1000:POSITION 4,21:
? "  player two ":PLAYER=12:GOSUB 1010
360 GOTO 340
1000 FOR TIME=1 TO TL:NEXT TIME:RETURN
1010 SOUND 0,0,0,0:G=18:R=18:POSITION
G,R:? CHR$(PLAYER);CHR$(29);CHR$(30);C
HR$(PLAYER+1):OV=3:OV1=32:GR=0:GC=0
1020 OPEN #2,4,0,"K:":REM OPEN KEYBOAR
D FOR A READ
1030 GET #2,KEY:IF KEY>127 THEN KEY=KE
```

```
Y-128:POKE 694,0:REM SET FOR NORMAL VI
DEO
1040 IF KEY>95 THEN KEY=KEY-32:POKE 70
2,64:REM SET FOR UPPER CASE
1050 IF KEY<66 OR KEY>89 THEN 1030:REM
 NOT A GOOD KEY
1060 IF KEY=72 OR KEY=84 OR KEY=89 OR
KEY=85 OR KEY=66 OR KEY=78 OR KEY=77 T
HEN CLOSE #2:GOTO 1080
1070 GOTO 1030
1080 IF R=18 AND KEY<>89 THEN 1020
1090 OR=R:OG=G:OGR=GR:OGC=GC:IF KEY=89
 THEN R=R-2:GC=GC+1:GR=GR+1:GOTO 1170:
REM MOVE UP
1100 IF KEY=78 THEN R=R+2:GC=GC-1:GR=G
R-1:GOTO 1170:REM MOVE DOWN
1110 IF KEY=85 THEN R=R-1:G=G+2:GC=GC+
1:GOTO 1170:REM UP AND RIGHT
1120 IF KEY=84 THEN R=R-1:G=G-2:GR=GR+
1:GOTO 1170:REM UP AND LEFT
1130 IF KEY=66 THEN R=R+1:G=G-2:GC=GC-
1:GOTO 1170:REM DOWN AND LEFT
1140 IF KEY=77 THEN R=R+1:G=G+2:GR=GR-
1:GOTO 1170:REM DOWN AND RIGHT
1150 IF KEY=72 THEN 1220:REM STAY THER
E
1160 GOTO 1020
1170 IF (GR<1 OR GR>9 OR GC<1 OR GC>9)
 AND R<>18 THEN G=OG:R=OR:GR=OGR:GC=OG
C:GOTO 1020
1180 LOCATE G,R,V:LOCATE G,R+1,V1:IF V
<>10 AND V<>12 AND V<>3 THEN G=OG:R=OR
:GR=OGR:GC=OGC:GOTO 1020
1190 POSITION OG,OR:? CHR$(OV);CHR$(29
);CHR$(30);CHR$(OV1)
1200 POSITION G,R:? CHR$(PLAYER);CHR$(
29);CHR$(30);CHR$(PLAYER+1):OV=V:OV1=V
1
1210 GOTO 1020
1220 IF GRID(GC,GR)<>0 THEN SOUND 0,50
,10,10:GOSUB 1000:SOUND 0,0,0,0:GOTO 1
020:REM NOT AVAILABLE
1230 GRID(GC,GR)=PLAYER:SOUND 0,150,10
,10:GOSUB 1000:SOUND 0,0,0,0:REM MARK
IT FOR THAT PLAYER
1240 IF PLAYER=10 THEN GOTO 1410
```

```
1250 V=1:X=1:OX=0:OV=0
1260 CX=X:CV=V:IF GRID(CV,CX)<>12 THEN
 X=X+1:IF X<>10 THEN 1260
1270 IF X=10 THEN 1400
1275 F=0:IF GRID(CV,CX)=12 AND CV=9 TH
EN 1390
1280 V1=0:X1=0:IF CV<9 THEN IF GRID(CV
+1,CX)=12 AND OV<>CV+1 THEN V1=CV+1:X1
=CX:F=1
1290 V2=0:X2=0:IF CV<9 AND CX<9 THEN I
F GRID(CV+1,CX+1)=12 AND OV<>CV+1 AND
OX<>CX+1 THEN V2=CV+1:X2=CX+1:F=1
1300 V3=0:X3=0:IF CX<9 THEN IF GRID(CV
,CX+1)=12 AND OX<>CX+1 THEN V3=CV:X3=C
X+1:F=1
1310 V4=0:X4=0:IF CX>1 THEN IF GRID(CV
,CX-1)=12 AND OX<>CX-1 THEN V4=CV:X4=C
X-1:F=1
1315 IF F=0 THEN GRID(CV,CX)=-12:IF CV
=1 THEN V=1:X=X+1:GOTO 1260
1320 IF CX=OX AND CV=OV AND F=0 THEN V
=1:X=X+1:GOTO 1260
1330 IF F=0 THEN CX=OX:CV=OV:GOTO 1270
1340 IF V1<>0 AND X1<>0 THEN OV=CV:OX=
CX:CV=V1:CX=X1:GOTO 1275
1350 IF V2<>0 AND X2<>0 THEN OV=CV:OX=
CX:CV=V2:CX=X2:GOTO 1275
1360 IF V3<>0 AND X3<>0 THEN OV=CV:OX=
CX:CV=V3:CX=X3:GOTO 1275
1370 IF V4<>0 AND X4<>0 THEN OV=CV:OX=
CX:CV=V4:CX=X4:GOTO 1275
1380 REM !! AND #2 IN INVERSE
1390 POSITION 4,21:? "  winner!! #2 ":
GOTO 1600
1400 FOR X=1 TO 9:FOR V=1 TO 9:GRID(V,
X)=ABS(GRID(V,X)):NEXT V:NEXT X:RETURN
1410 V=1:X=1:OX=0:OV=0
1420 CX=X:CV=V:IF GRID(CV,CX)<>10 THEN
 V=V+1:IF V<>10 THEN 1420
1430 IF V=10 THEN 1580
1440 F=0:IF GRID(CV,CX)=10 AND CX=9 TH
EN 1570
1450 V1=0:X1=0:IF CX<9 THEN IF GRID(CV
,CX+1)=10 AND OX<>CX+1 THEN X1=CX+1:V1
=CV:F=1
1460 V2=0:X2=0:IF CV<9 AND CX<9 THEN I
```

```
F GRID(CV+1,CX+1)=10 AND OV<>CV+1 AND
OX<>CX+1 THEN V2=CV+1:X2=CX+1:F=1
1470 V3=0:X3=0:IF CV<9 THEN IF GRID(CV
+1,CX)=10 AND OV<>CV+1 THEN V3=CV+1:X3
=CX:F=1
1480 V4=0:X4=0:IF CV>1 THEN IF GRID(CV
-1,CX)=10 AND OV<>CV-1 THEN V4=CV-1:X4
=CX:F=1
1490 IF F=0 THEN GRID(CV,CX)=-10:IF CX
=1 THEN X=1:V=V+1:GOTO 1420
1500 IF CX=OX AND CV=OV AND F=0 THEN X
=1:V=V+1:GOTO 1420
1510 IF F=0 THEN CX=OX:CV=OV:GOTO 1430

1520 IF V1<>0 AND X1<>0 THEN OV=CV:OX=
CX:CV=V1:CX=X1:GOTO 1440
1530 IF V2<>0 AND X2<>0 THEN OV=CV:OX=
CX:CV=V2:CX=X2:GOTO 1440
1540 IF V3<>0 AND X3<>0 THEN OV=CV:OX=
CX:CV=V3:CX=X3:GOTO 1440
1550 IF V4<>0 AND X4<>0 THEN OV=CV:OX=
CX:CV=V4:CX=X4:GOTO 1440
1560 GOTO 1500
1570 POSITION 6,21:? "  WINNER!! #1 ":
GOTO 1600
1580 FOR X=1 TO 9:FOR V=1 TO 9:GRID(V,
X)=ABS(GRID(V,X)):NEXT V:NEXT X:RETURN
1600 IF PEEK(53279)<>6 THEN 1600
1610 GOTO 140
```

ray. The GRID array keeps track of which locations are clear and which are occupied. H$ contains the characters that make up the shape of the grid.

Line 60 sets the screen to graphics 0. This is the text mode, to which the computer defaults. This program will change the display table and use the screen as a graphics mode, so the computer must execute a graphics command first. The first byte of the display list is placed into the variable DLIST. The address of the display list is stored in locations 560 and 561. The fourth location of the display list is changed to 68. This is the instruction code plus the display mode for the top row on the screen. Nineteen lines of the display list are changed to ANTIC mode 4. This mode will display four colors on the screen. The characters are eight pixels high and eight pixels wide.

Line 70 changes the text window from the text mode to graphics 1. This is ANTIC 6. This value is POKEd into the next four rows of the display list.

Line 80 finds out how much memory the computer has. This value is decreased by eight. The character set will be moved and placed just before the screen memory. The location of the new character set is stored in memory location 204. The location of the first byte of the ROM-based character set is stored in location 206. These two values are used by the assembly language subroutine that moves the character set from ROM into RAM. The colors of the characters are also changed to light and dark purple.

Line 90 reads the code from line 100 and stores it in memory locations 1536-1555. This is the assembly language subroutine that will move the character set from ROM into RAM. Line 100 contains the decimal codes for the assembly language subroutine.

Line 110 uses the USR command to execute the assembly language subroutine at memory location 1536. When the computer returns to this line the character set will be moved to RAM. To use this character set, POKE location 756 with the value of A.

Line 120 sets the value of CHARSET to the first byte of the character that will be changed. The FOR-NEXT loops read in the codes that will change 13 characters in the character set.

Lines 130-134 contain the codes that change the characters in the character set.

Line 140 uses the RESTORE command to ensure that the computer is pointing to this line. The characters of the four numbers in this line are placed in H$. These are the characters that will form the grid for the same.

Line 150 sets the variable R to zero. This is the row the grid characters will be printed in. The variable G is the number of grids that will be printed in that row. The first row contains one grid. The variable X is the column in which the grid will be printed. The loop steps backwards from 17 to 1. The value of G is moved to variable G1. The computer prints the grid on the screen. Variable G1 is decreased by one. If it is zero, the computer is sent to line line 170 to adjust for the next row.

Line 160 prints the grid on the screen again. Because the program uses a semicolon after printing H$, every H$ will be connected to the previous H$, making a complete grid across the screen. The variable G1 is decreased again. The computer loops at this line until G1 variable is zero.

Line 170 adds 1 to the value of R. This is the next row on which

the grid will be printed. The variable G is increased by 1. Every row will have one more grid in it. The loop continues until the first half of the grid is on the screen, then the value of G is decreased by 3. The second half of the grid contains a decreasing number of grids in it.

Line 180 changes the shape of the grid in H$. The computer was printing the top part of the grid when it was printing the top half of the grid. Now it has to change the shape of the character in H$ to print the bottom half of the grid.

Line 190 stores the value of G in the variable G1. The second character is printed on the screen followed by the characters in H$. The value of G is changed to one less than the value of G1. If G1 is 0, the computer goes to line 210 to print the rest of the grid.

Line 200 prints the remaining parts of the grid in this row. The variable G1 is decreased by 1 each time a portion of the grid is printed on the screen. When the row is complete, the computer continues with the next line.

Line 210 prints a space and then the first character to complete the grid row. The value of R is increased by 1 for the next row and the value of G is increased by 2. The characters in H$ are changed again. The middle row of the grid needed special treatment because it is the connecting row between the top and bottom of the grid. The rest of the grid can be printed like the top half was with the characters in H$.

Line 220 begins the FOR-NEXT loop. The variable S is the column where the grids for that row will begin. The value of G is stored in variable G1. The characters in H$ are printed on the screen. The value of G1 is decreased by 1. If the value G1 is 0, the computer is sent to line 240 to finish the row.

Line 230 prints the characters in H$ again. The computer loops at this line until the value of G1 is 0. This is the end of the row.

Line 240 prints the first character of H$ to close the grid row. The value of R is increased to point to the next row, and the value of G is decreased by 1. Each row will contain one less hex shape than the row before it. The loop continues until all the rows are printed on the screen. The last hex shape is completed and the variable G is set to 16.

Line 250 fills in the upper left side of the screen with light purple. The last character of each line replaces the grid character with the background color and the grid character.

Line 260 sets the variable G to 36. This is the column position of the lower right side of the screen.

Line 270 prints the grid character, then the color characters on this side of the screen. The loop continues until the lower right side of the screen is filled in with light purple.

Line 280 sets the variables G to 20 and the variable G1 to 0. Now the computer will fill in the other two corners.

Line 290 begins the FOR-NEXT loop that fills in the upper right corner of the screen. The first character printed is the grid character with the background color. The rest of the row is filled in with the background color.

Line 300 changes the value of G to 0 and the value of TL to 50.

Line 310 fills in the lower left corner of the screen.

Line 330 clears the values from the GRID array. The computer does not clear an array when the program is run. If we don't clear it before we use it, it may contain erroneous information.

Line 340 makes a sound, prints the player's number on the bottom of the screen, sets variable PLAYER to 10 and uses the subroutine in line 1010 to move the cursor on the screen. The value of variable PLAYER indicates which cursor will be printed on the screen.

Line 350 makes a different sound to indicate that it is the second player's turn. The value of PLAYER is changed to 12, and the computer uses the subroutine in line 1010 to move this cursor.

Line 360 sends the computer back to line 340. The computer loops at these lines until one player makes a path from one end of the grid to the other.

Line 1000 is a timing loop. This subroutine is used to slow the program down.

Line 1010 begins the subroutine that moves the cursor on the screen. The sound is turned off. Then the variables G and R are set to 18. This is the row and column at the bottom of the grid. The cursor for the player is printed at this position. The cursor is made up of two characters. The top half of the cursor is the value PLAYER. This is the top part of the cursor and the bottom line of the grid. The two values printed on the screen after the top part of the cursor are the backspace and the down arrow. This lowers the invisible cursor one row and moves it back one column. Now the bottom part of the cursor can be printed under the top part of the cursor. The variables OV, OV1, GR, and GC are set. These variables keep track of the position of the cursor on the grid.

Line 1020 opens the keyboard for a read.

Line 1030 waits until a key is pressed. The value of this key is stored in the variable KEY. If the value of KEY is greater than

127, the inverse key was pressed and the computer subtracts 128 from the value of KEY. The location 694 is POKEd with a zero to reset the flag to normal video.

Line 1040 checks to see if the value of KEY is greater than 95. If it is, the CAPS key was pressed and the key is in lowercase. Subtract 32 from the value of KEY to get the uppercase value of the key. The location 702 is POKEd with 64 to reset the keyboard for uppercase only.

Line 1050 checks to see if the value of the key is between 66 and 89. If it isn't, the computer goes back to line 1030 to set another input.

Line 1060 compares the value of KEY to see if it is one of the keys that will move the cursor. If it is, the keyboard is closed and the computer is sent to line 1080. The keys that move the cursor are the T, Y, U, B, N, and M. The H key will capture the hex location on which the cursor is resting.

Line 1070 sends the computer back to line 1030 to get another input. The computer will loop through these lines until one of the seven keys are pressed.

Line 1080 checks to see what row the cursor is in. If it is in row 18, the only direction the cursor can move is up. If the value of KEY is not 89 (the Y key) the computer will go back to line 1020 to get another input. If the direction of the cursor were not limited when it was outside the grid, it could be moved anywhere and not necessarily on the grid.

Line 1090 sets the values of the variables used to move the cursor. It is more complicated to move through hex locations than through squares. The value of KEY is checked to see if it is a Y. If it is, the value of R is decreased by 2. Each hex location is actually two rows high. The variables GC and GR are also increased by 1. The computer is sent to line 1170 to move the cursor up.

Line 1100 checks to see if the N key was pressed. If it was, the cursor will move down. The value of R is increased by 2 and the values of GC and GR are decreased by 1. The computer is sent to line 1170 to move the cursor down.

Line 1110 checks for the U key. This moves the cursor up and to the right. This location is only one row above the present row. The value of R is decreased by 1 and the variable G is increased by 2. The variable GC is increased by 1. The computer is sent to line 1170 to move the cursor.

Line 1120 checks the value of KEY for 84, a T. This moves the cursor up and to the left. The value of R is decreased by 1,

moving the cursor up one row; the value of G is decreased by 2, and the variable GR is increased by 1. The computer uses the routine at line 1170 to print the cursor on the screen.

Line 1130 checks for the letter B. This moves the cursor down and to the left. The variable R is increased by 1 to move it down one row. The variable G is decreased by 2 and the variable GC is decreased by 1. The computer is sent to line 1170 to print the cursor in the new location.

Line 1140 compares the value of KEY to 77. If the M key was pressed, the cursor will be moved down and to the right. The value of R is increased by 1, the value of G is increased by 2, and the value of GR is decreased by 1. The computer is sent to line 1170 to print the cursor.

Line 1150 checks to see if the H key was pressed. If it was, none of the values of the variables are changed. The computer is sent to line 1220 to print the cursor permanently in this location.

Line 1160 sends the computer to line 1020 to set another input. Although the computer should only use these lines if one of the seven keys are pressed, this is a safeguard in case the routine was used and one of the seven keys was not pressed.

Line 1170 checks the values of variables GR and GC. This is the grid columns and rows. If these variables are not between the values 1 and 9 and the value of R is not 18, the cursor is on the grid, but the new values would place it off the grid. The variables are reset to their original values and the computer is sent to line 1020 to set another input.

Line 1180 uses the LOCATE command to see if the location is already occupied. If the value of this location is not the top of one of the player's cursors and not an empty grid location, it is not a possible position for the cursor. The values of the variables are reset and the computer is sent to line 1020 to set another entry.

Line 1190 prints the contents of the grid in the position the cursor is in right now. If there is a piece in this position it is restored. If the grid location was empty, it is cleared again. This way, two pieces can occupy the same grid location when one piece is passing through.

Line 1200 prints the current cursor in the new position. The values of the character that occupied the location before this character was placed there are stored in the variables OV and OV1.

Line 1210 sends the computer to line 1020 to get another entry.

Line 1220 is used when the H key is pressed. The GRID array is checked to see if this position is available. The variables GC and

GR point to the column and row of the array. If this location does not contain a zero, the computer will make a sound, use the timing loop in line 1000, and send the computer back to line 1020 for another entry. Only one player can permanently occupy one location.

Line 1230 places the value of PLAYER in this location in the GRID array. The computer makes a sound that tells the player that he has captured this location.

Line 1240 checks to see which player has taken the location. if the value of PLAYER is 10, the computer goes to line 1410.

Line 1250 resets the variables V, X, OX, and OV. These variables will be used in the following routine, that checks to see if the player has made a complete path from one side of the grid to the other.

Line 1260 stores the values of X and V in CX and CV. The value of this location in GRID array is checked. If it is not 12, the value of X is increased by 1. If it is not 10, the line is repeated. This line loops until a value of 12 is found or the value of X equals 10. The computer is looking for this player's marker along the right bottom side of the grid.

Line 1270 sends the computer to line 1400 if the value of X is 10. This means there is no marker along this edge of the grid, so this player could not possibly have completed a path from one side to the other.

Line 1275 clears the variable F and checks to see if the variable CV is 9. This could be the last column of the grid. If the last column of the grid contains a 12, the computer is sent to line 1390. This is a completed path.

Lines 1280-1380 check the hex positions around the location that contains this player's cursor. If the next location also contains a 12, the variables are updated and the loop continues until the path ends or the computer reaches the other side of the grid. If the variable F is not set, that location is set to a negative number so the computer does not end up going back and forth between two locations forever.

Line 1390 declares this player a winner. This program line is reached only when a complete path from one side of the screen to the other is completed. The computer then goes to line 1600 and waits until the Start key is pressed.

Line 1400 resets the entire grid to its original values. Any numbers that were changed to negative numbers are reset to positive. The computer returns to the main lines of the program.

Lines 1410-1580 perform the same routine but in the opposite

order. This time the computer searches the grid from the top left side to the bottom right side to see if Player One has made a complete path across the grid. If he has, the computer declares this player the winner. If he hasn't, the computer resets all the variables in the grid and returns to the main program to give Player One another turn.

Lines 1600-1610 are the end of the program. The computer loops at these lines after one player wins. The computer continues the loop until the Start key or the System Reset key is pressed. Pressing the Start key sends the computer back to line 140 for another game. Pressing System Reset restores the screen to the text mode.

TREASURE HUNT

Objective of the game: To find all the lost treasures before running out of air.

Directions. A grid similar to the one used in "Battleship" is printed on the screen. In addition to the grid letters and numbers, the directions North, South, East, and West are printed on the screen. The diver appears in the upper left corner of the screen. You are instructed to enter a letter and a number. The letter entered should be one of the coordinates A-J. Then enter a comma, then the number coordinate. If you enter the wrong letter or number, the Delete key can be pressed and the number or letter will be removed.

Once you are satisfied with the coordinates you have entered, press the Return key. The diver will appear at that location on the grid. Look at the four letters that indicate direction. Some will be light green, others will be pink. This is a clue. The diver must travel toward the pink letters to find the treasure.

The units of oxygen are displayed near the bottom of the screen. Each time the diver moves, he uses 10 units of oxygen. When the diver finds the treasure, he surfaces, places the treasure on the screen and receives 100 units of oxygen. The grid clears and the diver is ready to be sent down for the second treasure. The game continues until the diver runs out of oxygen or finds the three treasures. If the diver finds all three treasures, the screen keeps flashing until the Return key is pressed. This restores the program for another game. If the diver runs out of air, the location of the treasure is displayed on the grid and the coordinates are printed under the grid. The computer waits until the Return key is pressed. In either case, the program ends after three games are played. The

treasures collected are displayed near the top of the screen. Figure 2-6 is the flowchart, Fig. 2-7 is the character set for this program, and Listing 2-4 shows the code.

Line 50 checks to see how much memory is in the computer. The computer subtracts 8 from this amount; this is where the character set will be moved. The computer stores this value at memory location 204. The computer stores the location of the character set in ROM at location 206. This information will be used in the assembly language subroutine that moves the character set from ROM to RAM.

Line 60 reads the code in line 70 and places it into memory locations 1536-1555. This code is the assembly language subroutine that moves the character set. Line 70 contains the decimal codes for the routine that moves the character set from ROM into RAM.

Line 80 changes the screen to graphics 17. This is graphics 1 with no text window. The USR command sends the computer to the assembly language subroutine at memory location 1536. When the computer returns to this line, it places the address of the character set in RAM into location 756.

Line 90 places the location of the first byte of the character that will be changed into the variable CHARSET. Seven characters will be changed in this character set. The FOR-NEXT loop reads the code for the new characters and places it in the area for the existing characters.

Lines 100-120 contain the codes for the new characters.

Line 130 prints the diver in the upper left corner of the screen. The units of oxygen are set to 100 and the variable G is set for the first game. The FOR-NEXT loop prints the grid on the screen.

Line 140 prints the four directions on the screen.

Line 150 prints the letters across the top of the grid and the numbers along the side. When X is equal to 10, the statement X = 10 will be equal to 1. The column value will then be 3 rather than 4. When X is not equal to 10, the statement is equal to 0. This keeps the number even on the screen.

Line 155 picks two random numbers. The variable TR is the row value for the treasure. The variable TC is the column variable.

Line 160 prints the units of oxygen and the game number on the screen.

Line 170 uses the subroutine in line 600 to make a sound. The computer then asks for a letter and a number. The cursor is placed on the screen.

Line 180 sets the variables AC and AR. This is the row and

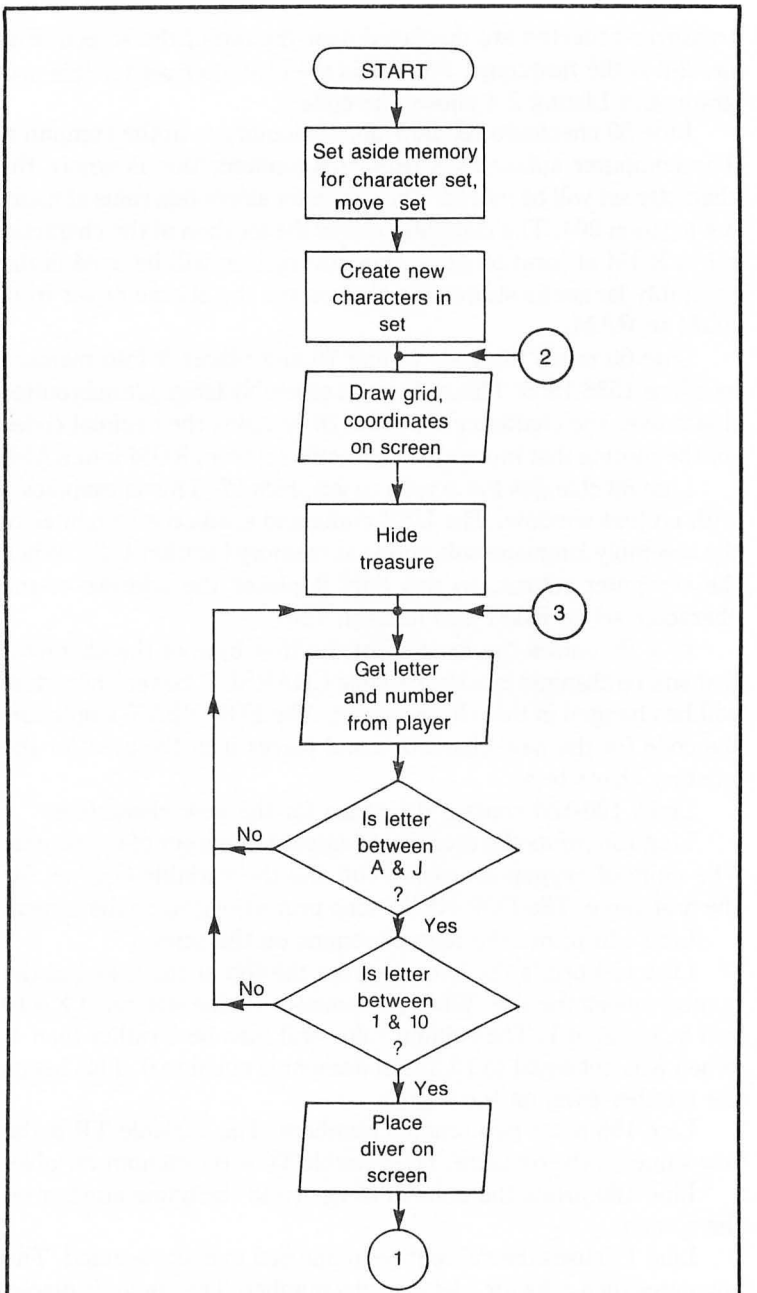
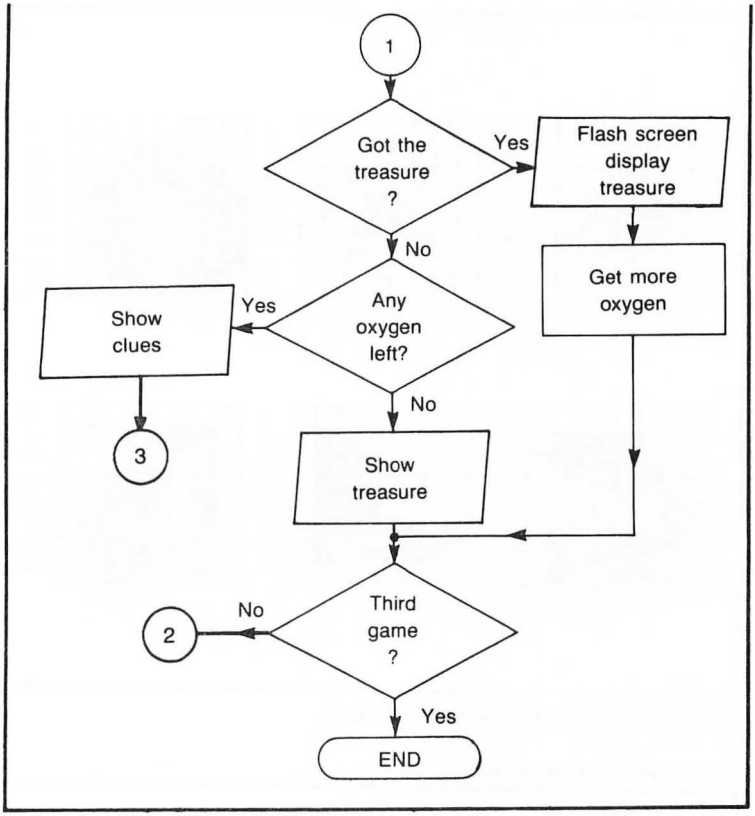


Fig. 2-6. Flowchart for Treasure Hunt.



column of the cursor. The letter that is entered is placed in this position on the screen. The keyboard is opened for a read.

Line 190 waits until a key is pressed. The value of this key is placed in the KEY variable.

Line 200 checks to see if the Return key is pressed. If the value of KEY is 155 and the value AC is greater than 11, the Return key will be accepted and the computer will go to line 350.

Line 210 checks to see if the value of KEY is greater than 127. If it is, the inverse video was pressed. The computer subtracts 128 from the value of KEY and POKEs location 694 with zero to reset the flag for normal video.

Line 220 checks to see if the value of KEY is 126. If it is, the Delete key was pressed. The computer goes to line 330 to erase the entry.

Line 230 checks the value of KEY to see if it is greater than 95. If it is, the computer subtracts 32 from this amount and resets

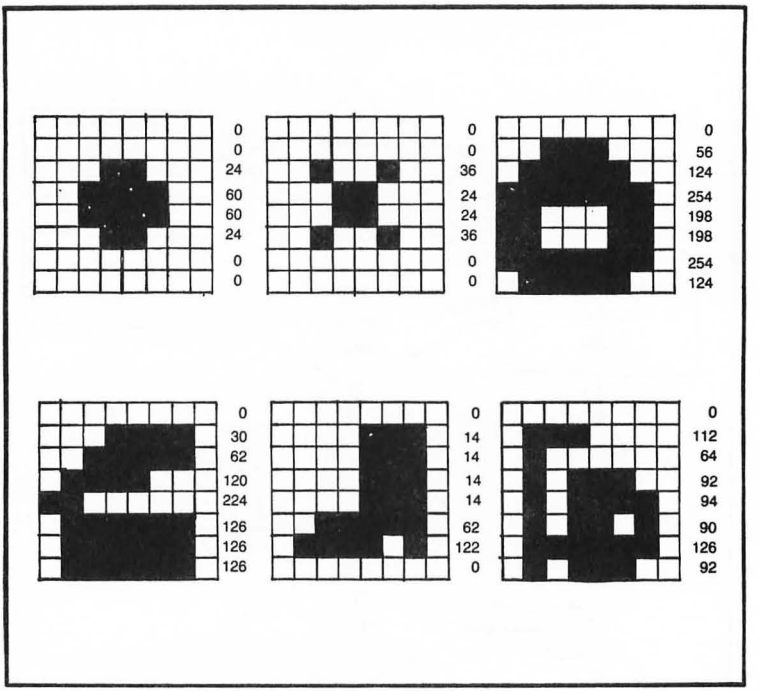


Fig. 2-7. Character set for Treasure Hunt.

Listing 2-4. Treasure Hunt.

```
10 REM TREASURE HUNT - FIND THE HIDDEN
 TREASURES IN THE LEAST NUMBER OF TRIE
S
20 REM CHAPTER 2 - GRID GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS

40 REM COPYRIGHT 1983
50 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756)
60 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM ROUTINE TO MOVE THE CHARACT
ER SET
70 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
80 GRAPHICS 17:Q=USR(1536):POKE 756,A:
REM MOVE IT AND USE IT
90 CHARSET=A*256+24:FOR X=CHARSET TO C
```

```
HARSET+55:READ V:POKE X,V:NEXT X:REM C
REATE NEW CHARACTERS
100 DATA 0,0,24,60,60,24,0,0,0,0,36,24
,24,36,0,0
110 DATA 0,56,124,254,198,198,254,124,
0,30,62,120,224,126,126,126
120 DATA 0,14,14,14,14,62,122,0,0,112,
64,92,94,90,126,92,0,126,126,126,126,1
26,126,0
125 REM OPEN PARENTHESIS & POUND SIGNS
 ARE INVERSE
130 R=0:C=0:POSITION C,R:? #6;"(":U=U+
100:G=G+1:FOR X=5 TO 14:POSITION 5,X:?
 #6;"##########":NEXT X
140 POSITION 10,1:? #6;"n":POSITION 1,
9:? #6;"w":POSITION 17,9:? #6;"e":POSI
TION 10,16:? #6;"s":REM DIRECTIONS
145 REM alphabet IS LOWER CASE INVERSE

150 POSITION 5,3:? #6;"abcdefghij":FOR
 X=1 TO 10:POSITION 4-(X=10),X+4:? #6;
X:NEXT X
155 TR=INT(RND(1)*10)+1:TC=INT(RND(1)*
10)+1:REM HIDING PLACE
160 POSITION 0,23:? #6;"UNITS ";U:POSI
TION 14,23:? #6;"GAME ";G;
165 REM letter, number AND CLOSE PAREN
THESIS ARE LOWER CASE INVERSE
170 GOSUB 600:POSITION 0,18:? #6;"ente
r letter,number":POSITION 9,20:? #6;"
    ":POSITION 9,20:? #6;")"
180 AC=9:AR=20:OPEN #2,4,0,"K:":REM RE
AD THE KEYBOARD
190 GET #2,KEY:REM GET THE KEY BEING P
RESSED
200 IF KEY=155 AND AC>11 THEN 350:REM
RETURN KEY PRESSED
210 IF KEY>127 THEN KEY=KEY-128:POKE 6
94,0:REM SET FOR NORMAL VIDEO
220 IF KEY=126 THEN 330:REM DELETE ENT
RY
230 IF KEY>95 THEN KEY=KEY-32:POKE 702
,64:REM SET FOR UPPER CASE
240 IF AC=9 THEN IF KEY<65 OR KEY>74 T
HEN 190:REM NOT A LETTER
250 IF AC=10 AND KEY<>44 THEN 190:REM
NOT A COMMA
```

```
260 IF AC>10 THEN IF KEY<48 OR KEY>57
THEN 190:REM NOT A NUMBER
270 IF AC=13 THEN 190
280 IF AC=9 THEN HC=KEY-64:GOTO 320
290 IF AC=11 THEN HR=KEY-48:GOTO 320
295 REM CLOSE PARENTHESIS IS INVERSE
300 IF AC=12 AND (KEY<>48 OR HR<>1) TH
EN POSITION AC,AR:? #6;")":GOTO 190
310 HR=10
320 POSITION AC,AR:? #6;CHR$(KEY+128);
")":AC=AC+1:GOTO 190:REM PRINT THE LET
TER,NUMBER
330 AC=AC-1:IF AC<9 THEN AC=9
335 REM CLOSE PARENTHESIS IS INVERSE
340 POSITION AC,AR:? #6;") ":GOTO 190
350 CLOSE #2:REM GOT THE ENTRY
360 IF R=0 AND C=0 THEN POSITION R,C:?
 #6;" ":GOTO 380:REM ERASE THE DIVER I
N THE CORNER
370 POSITION C+4,R+4:? #6;"$":REM X MA
RKS THE SPOT
380 R=HR:C=HC:POSITION C+4,R+4:? #6;"(
":REM PUT DIVER ON THE SPOT
390 IF R=TR AND C=TC THEN 500:REM GOT
IT
400 POSITION 10,1:? #6;"n":POSITION 1,
9:? #6;"w":POSITION 17,9:? #6;"e":POSI
TION 10,16:? #6;"s":REM DIRECTIONS
410 IF TC<C THEN POSITION 1,9:? #6;"w"
:REM GIVE CLUES - w IS LOWER CASE INVE
RSE
420 IF TC>C THEN POSITION 17,9:? #6;"e
":REM e IS LOWER CASE INVERSE
430 IF TR<R THEN POSITION 10,1:? #6;"n
":REM n IS LOWER CASE INVERSE
440 IF TR>R THEN POSITION 10,16:? #6;"
s":REM s IS LOWER CASE INVERSE
450 U=U-10:POSITION 5,23:? #6;"      ":
IF U=0 THEN 470
460 GOTO 160
470 SOUND 0,200,10,10:GOSUB 610:POSITI
ON 0,18:? #6;"  TREASURE WAS AT   ":PO
SITION 9,20:? #6;CHR$(TC+64);",";TR
480 POSITION TC+4,TR+4:? #6;CHR$(G+36)
:SOUND 0,0,0,0:IF G=3 THEN 550
490 OPEN #2,4,0,"K:":GET #2,KEY:CLOSE
#2:GOTO 130
```

```
500 POSITION 19-G,0:? #6;CHR$(36+G):RE
M PUT TREASURE ON SCREEN
510 IF G=3 THEN 550:REM GAME OVER
520 FOR X=1 TO 10:POKE 712,X:SOUND 0,2
00-X*10,10,10:GOSUB 610:NEXT X:SOUND 0
,0,0,0:POKE 712,0:GOTO 130
550 FOR X=1 TO 10:POKE 712,X:POKE 711,
X*10:POKE 710,X*10+5:POKE 709,X*5:SOUN
D 0,200-X*10,10,10:GOSUB 610:NEXT X
560 SOUND 0,0,0,0:IF PEEK(764)=255 THE
N 550
570 POSITION 15,0:? #6;"      ":G=0:U=0
:POKE 709,202:POKE 710,148:POKE 711,70
:POKE 712,0:GOTO 130
600 FOR T=1 TO 2:SOUND 0,50,10,10:GOSU
B 610:SOUND 0,0,0,0:GOSUB 610:NEXT T:R
ETURN :REM BEEP
610 FOR TIME=1 TO 20:NEXT TIME:RETURN
```

the keyboard for uppercase by POKEing location 702 with 64.

Line 240 validates the value of the key that was pressed. If the column position for the cursor is 9, the key must be a letter. If the value of KEY is not between 65 and 74, the key was not a letter from A-J. The computer is sent back to line 190 to get another input.

Line 250 checks to see if the cursor is in the tenth column. If it is, the only entry allowed is the comma. If the comma is not entered after the letter, the computer goes back to line 190 and waits for the comma to be entered.

Line 260 checks to see if a number key was entered if the value of AC is greater than 10. The computer goes back to line 190 if a number was not pressed.

Line 270 sends the computer back to line 190 if AC is equal to 13. Only the Return key or the Delete key can be entered if the value of AC is 13.

Line 280 takes the letter value of KEY and places it in the variable HC if the value of AC is 9.

Line 290 takes the number value of KEY and places it in the variable HR if the value of AC is 11.

Line 300 will accept the value of KEY only if the value of AC is 12, the value of KEY is 0, and the value of HR is 1. This way the player cannot enter other two-digit numbers for the row value.

Line 310 changes the value of HR to 10 because the value of HR was 1 and the 0 key was pressed.

Line 320 prints the number or the letter that was entered. The AC variable is incremented by 1 and the computer goes back to line 190 for another entry.

Line 330 is used when the Delete key is pressed. The computer subtracts one from the value of AC. If the value of AC is less than 9, the value is reset to 9.

Line 340 prints the cursor with a space after it. This erases the cursor from its previous position. The computer goes to line 190 for another entry.

Line 350 closes the keyboard after an entry has been made.

Line 360 checks the values of R and C. If both are 0, this is the first time the diver is moving. The computer erases the diver from the top left corner, then goes to line 380.

Line 370 removes the diver from its position on the grid. The diver is replaced with an X so the player will know which positions on the grid were already tried.

Line 380 places the values of HR and HC into the variables H and C. The diver is printed on the screen in the new location.

Line 390 checks to see if the row and column the diver is in matches the row and column the treasure is in. If both match, the computer goes to line 500.

Line 400 prints the four directions on the screen again. This restores any direction that was in a different color.

Line 410 checks the value of TC against the value of C. If the treasure's column position is less than the value of the variable C, the diver needs to move west. The letter W is printed in inverse.

Line 420 checks to see if the value of the treasure's column is greater than the position of the column. If it is, the diver should move east, and the letter E is printed in inverse.

Line 430 checks the variable TR against the value of R. If the treasure is in a row with a value less than that of the row the diver is in, the diver should move north. The letter N is printed in inverse.

Line 440 checks to see if the value of the treasure's row is greater than the row the diver is in. If it is, the diver should move south and the letter S is printed in inverse.

Line 450 subtracts 10 units of oxygen from the amount the diver has. If the diver has no oxygen left, the computer goes to line 470 to give the location of the treasure.

Line 460 sends the computer to line 160 for another entry.

Line 470 makes a sound, and prints the location of the treasure on the screen.

Line 480 places the treasure on the grid. The character that

is the treasure is determined by the value of G. The computer goes to line 560 if this was the third game.

Line 490 waits for a key to be pressed. The keyboard is opened for a read, and the computer waits until the player presses a key. The keyboard is closed and the computer goes to line 130 for another game after the key is pressed. The value of the key is disregarded.

Line 500 prints the treasure at the top of the screen. The treasure can be a ring, a chest, or an old boot.

Line 510 checks to see if this is the third game. If it is, the computer goes to line 550.

Line 520 makes a sound and changes the color of the screen, then sends the computer to line 130 for another game.

Lines 550-560 end the game. The computer changes the color of the screen and makes sounds. If no key is pressed, the computer will continue to loop between these two lines.

Line 570 resets the variables for the number of games and the units of oxygen, restores the colors to the screen and sends the computer to line 130 to play another set of games. To end the program completely, press the System Reset key.

Line 600 makes the prompt sound.

Line 610 is the timing loop.

THE GREAT ABYSS

Objective of the game: To capture the creature without getting killed or falling into the bottomless pit.

Directions: You begin at one position of interconnecting rooms and locations. The swamp creature lives in one of these rooms. There are flying creatures and a bottomless pit in this area. You have five arrows. You are trying to capture the creature with one of these arrows. You can capture the creature by shooting an arrow into the room where he is hiding. It's not simple; if you walk into the room where he is, he will capture you.

There are many hazards throughout the caves. One of the flying creatures can come into the room with you, pick you up, and deposit you in a different room. This could be an ordinary room, the room where the swamp creature is, or the bottomless pit! You have no control over the flying creature.

The computer will give you clues when you are near hazards. The screen will turn light blue when you are one room away from the bottomless pit. A worm will crawl across the screen when you are one room away from the swamp creature. The computer will

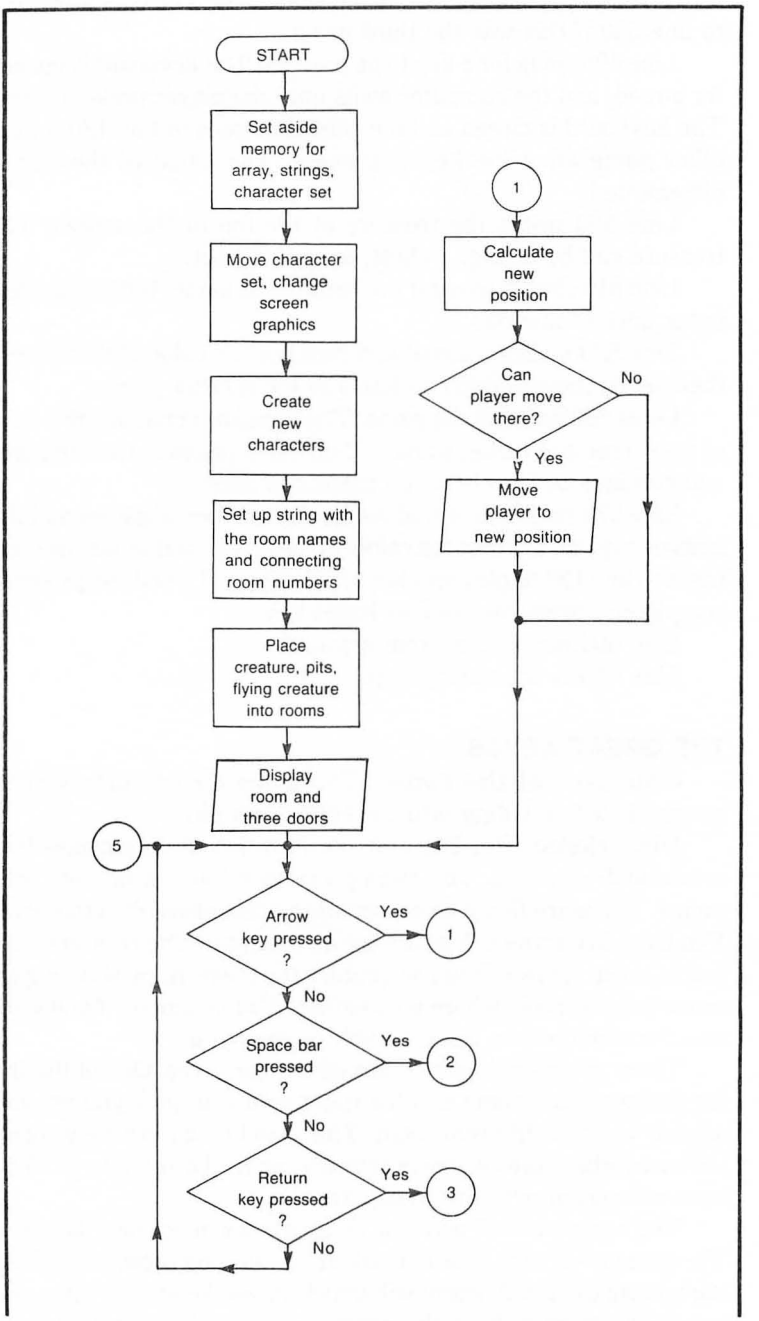
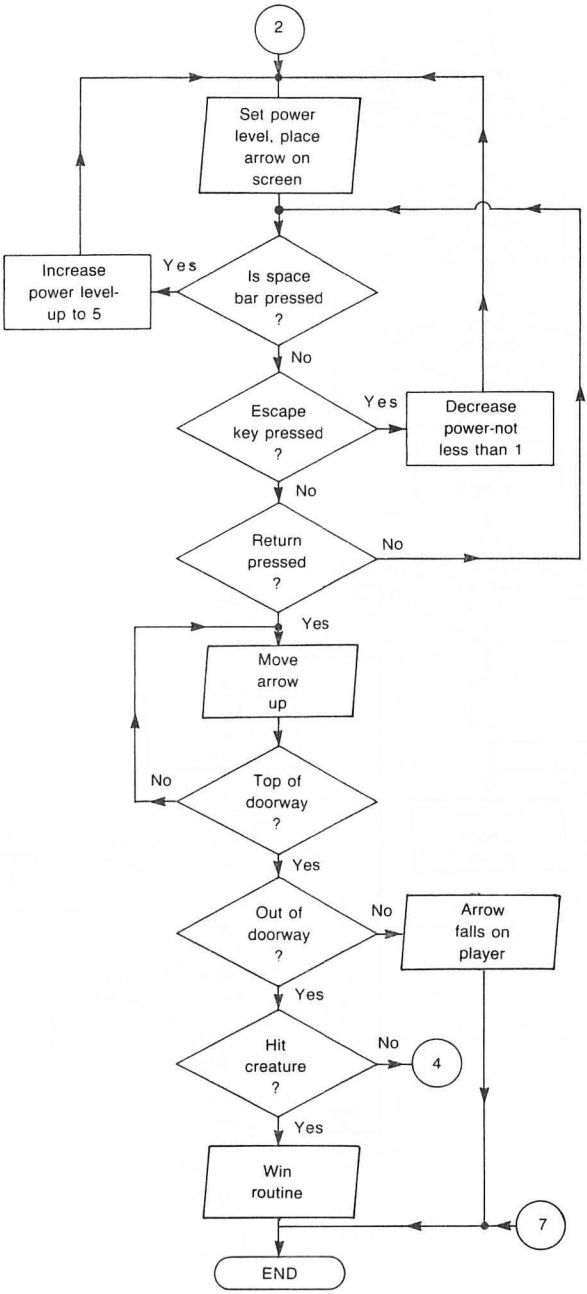
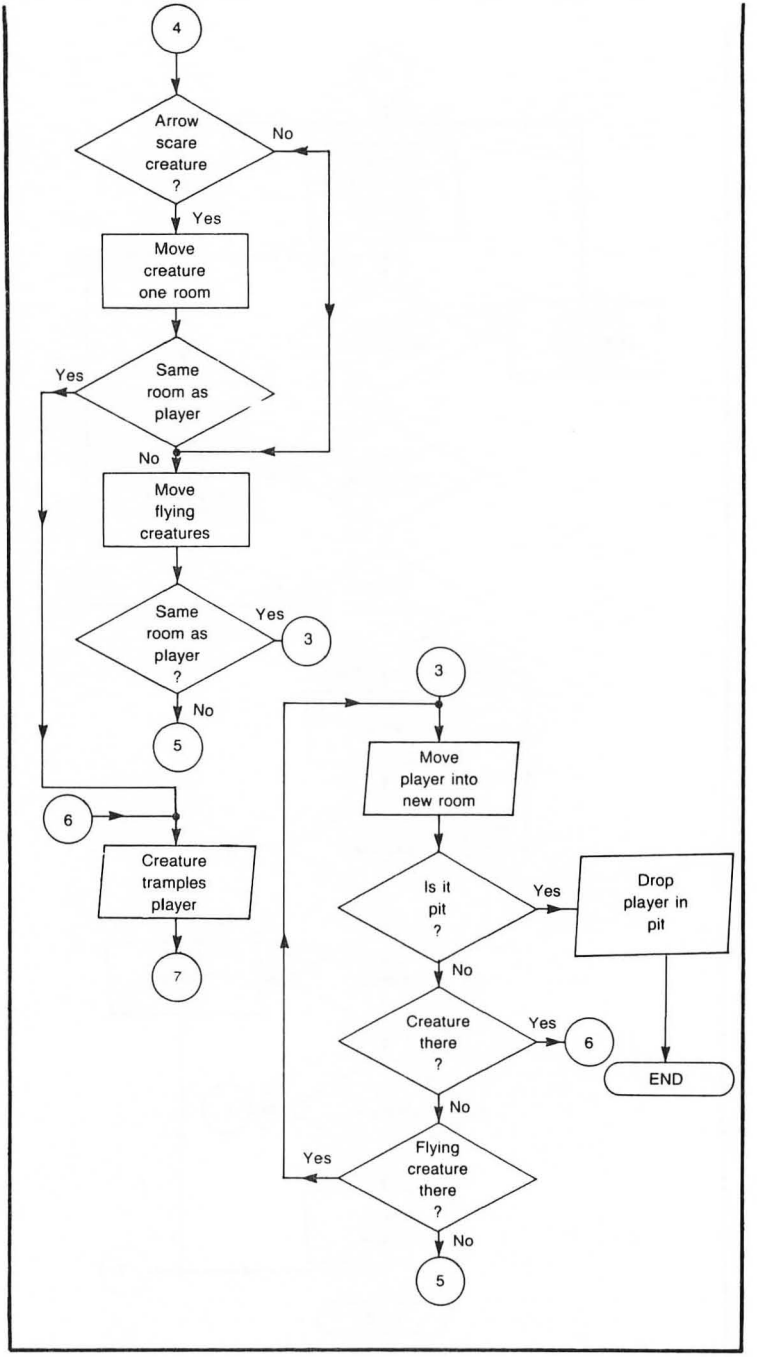


Fig. 2-8. Flowchart for The Great Abyss. (Continued through page 120.)


2
Set power level, place arrow on screen
Is space bar pressed ?
Yes
Increase power level-up to 5
No
Escape key pressed ?
Yes
Decrease power-not less than 1
No
Return pressed ?
No
Yes
Move arrow up
Top of doorway ?
No
Yes
Out of doorway ?
No
Arrow falls on player
Yes
Hit creature ?
No
4
Yes
Win routine
7
END


4
Arrow scare creature ?
No
Yes
Move creature one room
Same room as player
Yes
No
Move flying creatures
Same room as player ?
Yes
3
No
5
6
Creature tramples player
7
3
Move player into new room
Is it pit ?
Yes
Drop player in pit
No
Creature there ?
Yes
6
END
No
Flying creature there ?
Yes
No
5

make a sound when the flying creatures are near.

To move yourself, press the left and right arrows. To go into the next room, press the Return key. To shoot an arrow, press the space bar. The power of the arrow begins with 1. This is the distance or number of rooms the arrow can travel. To lower the power, press the Escape key. To fire the arrow, press the Return key. If you capture the swamp creature, a message will appear on the screen. If the arrow goes through a room where the creature is hiding, he will run into another room. He can only move into one room at a time. If he comes into your room, he will capture you. Figure 2-8 is the flowchart, Fig. 2-9 is the character set for this program, and listing 2-5 is the code.

Line 50 sets aside the memory needed for the strings and the

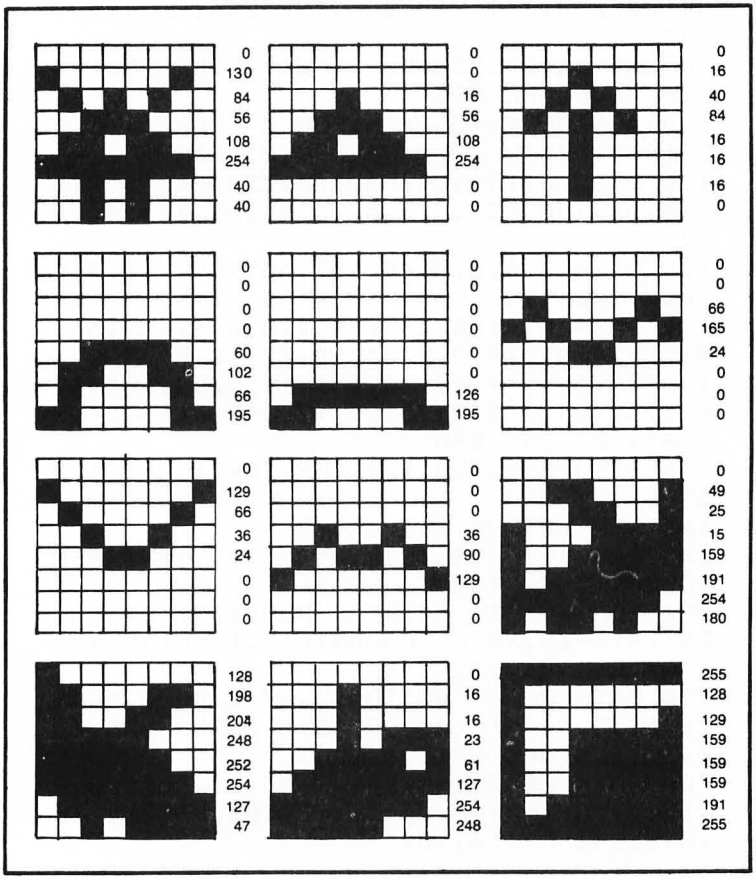


Fig. 2-9. Character set for The Great Abyss.

Listing 2-5. The Great Abyss.

```
10 REM THE GREAT ABYSS - CAPTURE THE C
REATURE WITHOUT FALLING INTO THE BOTTO
MLESS PIT!
20 REM CHAPTER 2 - GRID GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM C(20,3),R$(320),ROOM$(16)
60 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 18:Q=USR(1536):POKE 756,A:
REM MOVE THE SET
100 CHARSET=A*256+24:FOR X=CHARSET TO
CHARSET+95:READ V:POKE X,V:NEXT X
110 DATA 0,130,84,56,108,254,40,40,0,0
,16,56,108,254,0,0,0,16,40,84,16,16,16
,0
120 DATA 0,0,0,0,60,102,66,195,0,0,0,0
,0,0,126,195
130 DATA 0,0,66,165,24,0,0,0,0,129,66,
36,24,0,0,0,0,0,0,0,36,90,129,0,0
140 DATA 0,49,25,15,159,191,254,180,12
8,198,204,248,252,254,127,47,0,16,16,2
3,61,127,254,248
150 DATA 255,128,129,159,159,159,191,2
55
160 REM SET UP THE CAVES
170 R$(1)=" ":R$(320)=" ":R$(2)=R$:REM
 CLEAR THE STRING
180 FOR X=1 TO 20:READ ROOM$,R1,R2,R3:
R$(X*16-15,X*16)=ROOM$:C(X,1)=R1:C(X,2
)=R2:C(X,3)=R3:NEXT X
190 DATA ROCKY BOTTOM,5,19,2,MOSS HEAP
,1,11,3,SWAMP LAND,2,13,4,MARSH GROVE,
3,15,5,STONE WALL,4,17,1
200 DATA PEBBLE CREEK,7,20,10,GLADES,8
,12,6,ICE ARENA,9,14,7,CATACOMBS,10,16
,8,SLIME PIT,6,18,9
210 DATA BAT HAVEN,20,2,12,LOVER LEAP,
11,7,13,SATAN PIT,12,3,14,QUICKSAND,13
```

```
,8,15,DEVIL TOWER,14,4,16
220 DATA GAS MIRE,15,9,17,ARTIC VIEW,1
6,5,18,TREMBLING RAVINE,17,10,19,TIDEL
AND,18,1,20,GREMLIN PEAK,19,6,11
230 CREATURE=INT(RND(1)*19)+2:REM PLAC
E THE CREATURE
240 PIT1=INT(RND(1)*19)+2:IF PIT1=CREA
TURE THEN 240:REM MAKE A PIT - CREATUR
E CANNOT LIVE IN THE PIT
250 PIT2=INT(RND(1)*19)+2:IF PIT2=PIT1
 OR PIT2=CREATURE THEN 250:REM MUST BE
 DIFFERENT
260 HENS=INT(RND(1)*19)+2:REM HENS CAN
 BE ANYWHERE
270 REM START THE GAME IN ROOM 1
280 A=5:ROOM=1:PC=9:PR=10:MW=0:GOSUB 1
000
290 T=0:KEY=PEEK(764):POKE 764,255
300 IF KEY=7 THEN PC=PC+1:IF PC=19 THE
N PC=18:GOTO 290:REM MOVE RIGHT
310 IF KEY=6 THEN PC=PC-1:IF PC=-1 THE
N PC=0:GOTO 290:REM MOVE LEFT
320 IF KEY=33 THEN P=1:T=100:PC=PC+1:G
OTO 390:REM SHOOT ARROW
330 IF KEY=12 THEN 600:REM MOVE TO NEX
T ROOM
340 IF MW THEN POSITION MW,9:? #6;" ";
CHR$(38+(MW/2=INT(MW/2)));CHR$(39-(MW/
2=INT(MW/2)));" ":MW=MW-1
350 IF MW=1 THEN POSITION MW,9:? #6;"
   ":MW=17
360 IF KEY=255 OR PC=0 OR PC=18 THEN 2
90
370 SOUND 0,200,8,8:GOSUB 1100:SOUND 0
,0,0,0:POSITION PC,PR:? #6;" ";CHR$(35
+(PC/2=INT(PC/2))+128);" "
380 GOTO 290
385 REM power IS LOWER CASE INVERSE
390 POSITION 7,3:? #6;"power ";P:POW=P
EEK(764):POKE 764,255:IF POW=33 THEN P
=P+1:IF P=6 THEN P=5
400 IF POW=28 THEN P=P-1:IF P=0 THEN P
=1
410 IF POW<>12 THEN 390
420 FOR X=100 TO 0 STEP -5:SOUND 0,X,1
2,10:NEXT X:SOUND 0,0,0,0:T=T/P:FOR X=
PR-1 TO 4 STEP -1:LOCATE PC,X,V
```

```
425 REM PERCENT SIGN (%) IS INVERSE
430 POSITION PC,X:? #6;"%":GOSUB 1100:
POSITION PC,X:? #6;CHR$(V):NEXT X
440 IF V=174 OR V=142 OR V=14 THEN 460
:REM THROUGH THE TUNNEL
445 REM PERCENT SIGN (%) IS INVERSE
450 FOR X=4 TO PR:LOCATE PC,X,V:POSITI
ON PC,X:? #6;"%":GOSUB 1100:POSITION P
C,X:? #6;CHR$(V):NEXT X:GOTO 1380
460 PC=PC-1:IF V=14 THEN V=1:GOTO 490
470 IF V=174 THEN V=2:GOTO 490
480 V=3
490 R3=ROOM:FOR R1=1 TO P:R2=C(R3,V):V
=INT(RND(1)*3)+1:R3=R2:IF R3=CREATURE
AND R1<>P THEN W=1:GOTO 520
500 NEXT R1:IF R3=CREATURE THEN 1190:R
EM GOT IT
505 REM MISSED IS INVERSE
510 A=A-1:POSITION A,0:? #6;" ":POSITI
ON 7,3:? #6;"MISSED  ":T=100:GOSUB 110
0:IF A=0 THEN 1200:REM YOU LOOSE
520 IF W=1 OR MW>0 THEN V=INT(RND(1)*3
)+1:CREATURE=C(CREATURE,V):W=0:REM MOV
E THE CREATURE
530 IF CREATURE=ROOM THEN 1200:REM CRE
ATURE GOT YOU - YOU LOOSE
540 HENS=INT(RND(1)*20)+1:IF HENS=ROOM
 THEN GOSUB 1250:GOTO 560:REM MOVE THE
 MARSH HENS - YOU CAN MOVE
550 GOSUB 1000:GOTO 290:REM PLAY AGAIN

560 ROOM=INT(RND(1)*20)+1:IF ROOM=HENS
 THEN 560:REM NEW ROOM FOR YOU
570 GOSUB 1000:IF ROOM=PIT1 OR ROOM=PI
T2 THEN 1350:REM YOU REALLY LOOSE
580 GOTO 290
590 REM MOVE TO THE NEXT ROOM
600 IF PC>0 AND PC<5 THEN V=1:GOTO 640
610 IF PC>6 AND PC<11 THEN V=2:GOTO 64
0
620 IF PC>12 AND PC<17 THEN V=3:GOTO 6
40
630 GOTO 290
640 ROOM=C(ROOM,V):GOSUB 1000:REM GO T
O NEW ROOM
650 IF ROOM=PIT1 OR ROOM=PIT2 THEN 135
```

```
0:REM YOU LOOSE - DOWN YOU GO
660 IF ROOM=CREATURE THEN 1200:REM YOU
 LOOSE AGAIN
670 IF ROOM=HENS THEN GOSUB 1250:GOTO
700:REM YOU MOVE
680 HENS=INT(RND(1)*20)+1:IF ROOM=HENS
 THEN GOSUB 1250:GOTO 700
690 GOSUB 1030:GOTO 290
700 ROOM=INT(RND(1)*20)+1:IF ROOM=HENS
 THEN 700
710 GOTO 570:REM MOVE TO NEW ROOM
995 REM } IS THE CODE FOR CLEAR THE SC
REEN
1000 POKE 712,0:MW=0:? #6;"}":ROOM$=R$
(ROOM*16-15,ROOM*16):FOR P=16 TO 6 STE
P -1:IF ROOM$(P,P)=" " THEN NEXT P
1010 P=10-P/2:POSITION P,1:? #6;ROOM$:
FOR P=1 TO A:POSITION P-1,0:? #6;"%":N
EXT P
1015 REM NEXT LINE CONSISTS OF FOUR CO
NTROL N's, FOUR INVERSE PERIODS, AND F
OUR INVERSE SPACES.
1020 FOR X=4 TO 8:POSITION 2,X:? #6;"
  ....  ":NEXT X:POSITION PC,PR
:? #6;" #"
1030 FOR X=1 TO 3:IF C(ROOM,X)=CREATUR
E THEN MW=17
1040 IF C(ROOM,X)=PIT1 OR C(ROOM,X)=PI
T2 THEN POKE 712,150
1050 IF C(ROOM,X)<>HENS THEN 1070
1060 T=50:FOR V=1 TO 4:SOUND 0,200,10,
10:GOSUB 1100:SOUND 0,250,10,10:GOSUB
1100:NEXT V:SOUND 0,0,0,0
1070 NEXT X
1090 RETURN
1100 FOR DELAY=1 TO T:NEXT DELAY:RETUR
N
1190 POSITION 4,2:? #6;"YOU GOT IT!!":
GOTO 1380
1195 REM PLUS SIGN, COMMA, AND DASH AR
E INVERSE
1200 T=10:FOR X=0 TO PC-1:POSITION X,P
R:? #6;" +,-":SOUND 0,200,12,10:GOSUB
1100:NEXT X:SOUND 0,0,0,0
1210 GOTO 1380
1250 T=50:FOR X=1 TO PC+1:LOCATE X,3,V
:POSITION X,3:? #6;CHR$(168+(X-INT(X/3
```

```
)*3)):GOSUB 1100:POSITION X,3
1260 ? #6;CHR$(V):NEXT X:GOSUB 1310:FO
R X=4 TO PR-1:LOCATE PC+1,X,V:POSITION
 PC+1,X:? #6;CHR$(168+(X-INT(X/3)*3))
1270 GOSUB 1100:POSITION PC+1,X:? #6;C
HR$(V):NEXT X:POSITION PC+1,PR-1:? #6;
CHR$(169):GOSUB 1310:V=0:V1=0
1280 FOR X=PR-1 TO 3 STEP -1:IF X<>9 T
HEN LOCATE PC+1,X,V:LOCATE PC+1,X+1,V1

1290 POSITION PC+1,X:? #6;CHR$(168+(X-
INT(X/3)*3)):POSITION PC+1,X+1:? #6;"#
"
1300 GOSUB 1100:POSITION PC+1,X:? #6;C
HR$(V):POSITION PC+1,X+1:? #6;CHR$(V1)
:NEXT X
1310 FOR V=1 TO 4:SOUND 0,200,10,10:GO
SUB 1100:SOUND 0,250,10,10:GOSUB 1100:
NEXT V:SOUND 0,0,0,0:RETURN
1350 LOCATE PC+1,7,V1:ROOM$(1)=CHR$(V1
):ROOM$(16)=ROOM$(1):ROOM$(2)=ROOM$
1360 T=10:FOR X=2 TO 11:POSITION 2,X:?
 #6;ROOM$:NEXT X:FOR X=4 TO 11:LOCATE
9,X,V:POSITION 9,X:? #6;"#"
1370 GOSUB 1100:SOUND 0,X*10,12,10:POS
ITION 9,X:? #6;CHR$(V):NEXT X:SOUND 0,
0,0,0:REM FALLING
1380 POSITION 5,9:? #6;"press START":I
F PEEK(53279)<>6 THEN 1380
1390 GOTO 230
```

arrays. There are 20 rooms. Each room leads into three other rooms. The rooms this room connects to are stored in the C array. The name of the room is stored in R$. ROOM$ is used to read the name from the data list.

Line 60 checks to see how much memory is in the computer. This amount is decreased by 8. This new value is the beginning location of the character set that will be moved into RAM. This value is stored in memory location 204. The beginning location of the ROM-based character set is stored in location 206. This information is used in the assembly language subroutine that moves the character set from ROM into RAM.

Line 70 reads the code for the assembly language subroutine and places it into memory locations 1536-1555; line 80 contains the

decimal code for the routine.

Line 90 changes the screen to graphics 18. This is graphics 2 with no text window. The computer sees the USR command to execute the assembly language subroutine that begins in location 1536. When the computer returns to this line, it POKEs the value of A into memory location 756. This tells the computer to use the character set in RAM.

Line 100 finds the address of the first character in the character set that will be changed. This address is stored in the variable CHARSET. The computer reads the codes for the new characters and stores them in the area set aside for the character set.

Lines 110-150 contain the decimal codes for the new characters.

Line 170 clears the information from R$, because the computer does not clear out the area set aside for strings and arrays when a program is run.

Line 180 reads the information from the data lines and stores it in R$ and C array. Each time the loop is executed, the computer will read the name of the room and the numbers of the three rooms connected to this room.

Lines 190-220 contain the names of the rooms and the connecting room numbers.

Line 230 picks one of the 20 rooms for the creature. The number of the room the creature is in is stored in the variable CREATURE. The first room cannot be used for the creature, the pits, or the flying creatures.

Line 240 picks a number for the bottomless pit. The computer compares this number with the room number of the creature. If they are both the same, the computer picks another number for the pit. The creature cannot live in the pit.

Line 250 picks a number for the second pit. There are two pits in this program. The computer compares the number of this pit with the last pit and the creature's room. If this number is the same as either of the others, this line is repeated.

Line 260 picks a room for the flying creatures. Their room number is stored in the variable HENS. Since they fly, they can be in any room, even the bottomless pits!

Line 280 begins the game. The variable A is the number of arrows you have. You begin the game with five arrows. The ROOM variable is the room you are in. You begin in room 1. The variables PC and PR indicate the row and column in which the player will be printed. The variable MW is set to 0. When this variable is not 0, the worm moves across the screen. The computer uses the

subroutine at line 1000 to put the room and the three doors on the screen.

Line 290 clears the variable T, then gets the value of location 764. The computer POKEs this location with 255 to clear it.

Line 300 checks to see if the right arrow key was pressed. If it was, the computer increases the value of PC by 1; if the value of PC is equal to 19, it is reset to 18. The player must stay on the screen.

Line 310 subtracts 1 from the value of PC if the left arrow key is pressed. If the value of PC is less than 1, the value of PC is reset to 0.

Line 320 checks to see if the space bar has been pressed. If it has, the variable P is set to 1, the variable T is set to 100, the column variable is increased by 1, and the computer is sent to line 390.

Line 330 checks to see if the Return key has been pressed. If it has, the computer goes to line 600 to move the player.

Line 340 checks the variable MW. If it is not 0, the player is near the swamp creature's room. The two characters that make up the worm are printed on the screen. The way the command is written for printing and the characters, the characters will alternate their pattern depending on the value of MW. When MW is an odd number, character 38 will be printed before character 39. When MW is an even number, character 39 will be printed before character 38. This is because the statement

$$MW/2 = INT\ (MW/2)$$

is equal to 1 when it is true, and 0 when it is false. This value is added to 38 and 39. After the worm is printed, the last position is erased and the variable MW is decreased by 1. This keeps the creature moving across the screen.

Line 350 erases the character from the screen when MW reaches 1 and resets MW to 17. If the worm was not cleared from the screen, it would remain there until the player changes rooms because the last segment of the worm is erased in the line where the worm is printed. If it is at the beginning of the line, the computer will not wrap around to the left edge of the screen.

Line 360 checks to see if the variable KEY is 255, 0, or 18. If it is, the computer is sent to line 290 to get another input.

Line 370 makes a sound and prints the player in the new location. There are two characters that can be used as the player. When

the value of PC is even, the second, shorter character is printed. When PC is odd, the taller character is printed. The positions on both sides of the character are erased.

Line 380 sends the computer back to line 290 for another entry.

Line 390—when the space bar is pressed, the power value is printed at the top of the screen. The computer looks at location 764 to see if the space bar has been pressed again. If it has, the power level is increased by 1. The power level cannot exceed 5.

Line 400 checks to see if the Escape key has been pressed. If it has, the power level is decreased by 1. The power level can never go below 1.

Line 410 sends the computer to line 390 if the Return key has not been pressed. The computer loops through these lines until the Return key is pressed.

Line 420 shoots an arrow through the doorway the player is standing in front of. The computer makes a sound, then begins a FOR-NEXT loop that prints the arrow on the screen. The computer uses the LOCATE command to get the value of the character over which the arrow is going to be printing. It stores this value in the variable V.

Line 430 prints the arrow on the screen, uses the subroutine in line 1100, then prints the character on the screen, erasing the arrow. The loop continues until the arrow reaches the top of the door.

Line 440 looks at the last value of V. If it is 174, 142, or 14, then the arrow went through the doorway and into the tunnel. The computer is sent to line 460.

Line 450 reverses the direction of the arrow. Since the player was not in front of a door when the arrow was shot, the arrow has hit the wall and is now rebounding toward the player. The computer uses the same routine in reverse. The LOCATE command gets the value of the character over which the arrow will be printing. The arrow is printed on the screen, and the subroutine at line 1100 delays its movement for a few seconds. The character that was in that location is printed back on the screen. The loop continues until the arrow is back on the bottom of the screen. The computer is then sent to line 1380. The game is over, the player got shot with its own arrow.

Line 460—the value of PC is decreased by 1 so the player is in his or her original position on the screen. If the value of V is 14, the arrow was shot through the first room. The value of V is changed to 1.

Line 470 changes the value of V to 2 if the value of V is 174.

Line 480 changes the value of V to 3.

Line 490 stores the value of ROOM in variable R3. ROOM is the room number the player is in. The FOR-NEXT loop takes the arrow through a number of rooms based on the power of the arrow. The variable R2 is the connecting room based on the value of V. Then the computer chooses a number from 1 to 3; this is the direction the arrow will go in if the arrow travels through any more rooms. The value of R2 is moved into the variable R3. If the creature is in the same room the arrow is in, and this is not the last room the arrow will be in, the computer goes to line 520. The creature is disturbed.

Line 500 continues the loop. If the arrow lands in the room where the creature is, he is captured. The computer goes to line 1190.

Line 510 decreases the number of arrows the player has. The message is printed on the screen, and the computer uses the timing loop in line 1100. If there are no arrows left, you lose the game.

Line 520 checks if the variable W is set to 1, or the value of MW is greater than 0. The creature is disturbed and will move. The computer chooses one of the three possible directions in which the creature can move. The creature is moved to the new room and the variable W is reset to 0.

Line 530 checks to see if the creature has moved into your room. If it has, the computer goes to line 1200. You lose.

Line 540 gives the flying creatures a chance to move. If they move into the room you are in, the computer goes to the subroutine in line 1250 to move you, then to line 560.

Line 550 restores the screen and goes back to line 290 to play the game.

Line 560 chooses a new room for you to land in. If the room you are in is chosen, the line repeats until the computer finds you a new room.

Line 570 uses the subroutine in line 1000 to print the room on the screen. If you have been dropped into one of the pits, the computer is sent to line 1350, where you fall down the screen into the bottomless pit.

Line 580 sends the computer back to line 290 to finish the game.

Line 600 moves the player to the next room. If the player's column value is between 1 and 4, the player stands before the first doorway. The value of V is set to 1 and the computer goes to line 640 to move.

Line 610 checks to see if the column value of the player is between 7 and 10. If it is, the player is in front of the second or middle doorway. The value of V is set to 2 and the computer goes to line 640.

Line 620 checks to see if the column value of the player is between 13 and 16. If it is, the player is in front of the third doorway and variable V is set to 3.

Line 630 sends the computer back to line 290. The player cannot move through the wall.

Line 640 changes the value of the ROOM variable to the value pointed to by the variable V. The computer uses the subroutine in line 1000 to update the screen.

Line 650 checks to see if the player fell into the pit. If he did, the computer is sent to line 1350 to end the game.

Line 660 checks to see if the player wandered into the swamp creature's room. If he did, the computer is sent to line 1200 and the player loses the game.

Line 670 checks to see if the flying creatures are in this room. If they are, the computer uses the subroutine in line 1250 to move you up. When the computer returns to this line, it goes to line 700 to move the flying creatures.

Line 680 picks a room for the flying creatures. If they move into your room, the computer uses the subroutine at line 1250 to move you. Then the computer uses the routine at line 700 to move the flying creatures again.

Line 690 sends the computer to line 1030 to update the information, then to line 290 to continue the game.

Line 700 picks a new room for the flying creatures. If the computer picks the room they are currently in, the line repeats itself until the flying creatures move.

Line 710 sends the computer to line 570 to see if the player is in the pits.

Line 1000 prints the room and the doorways on the screen. The memory location 712 is POKEd with 0 to blacken the screen. The screen is cleared. The name of the room the player is in is moved into ROOM$. The FOR-NEXT loop looks for the end of the word in ROOM$.

Line 1010 finds the center of the word and prints the name of the room centered on the screen. The arrows are printed in the top left corner of the screen.

Line 1020 prints the doorways on the screen. The characters between the quotation marks are four Ctrl-Ns, a space, four inverse

periods, a space, and four inverse spaces. The player is printed on the screen. It is an inverse pound sign.

Line 1030 checks to see if the swamp creature is in any of the three adjoining rooms. If it is, the variable MW is set to 17. The worm will crawl across the screen when the computer returns to the main part of the program.

Line 1040 checks to see if any of the adjoining rooms is a pit. If it is, the screen color is changed.

Line 1050 checks to see if the flying creatures are in adjoining rooms. If they aren't, the computer is sent to line 1070.

Line 1060 makes the warning sound if the flying creatures are in an adjoining room.

Line 1070 continues the loop.

Line 1090 returns the computer to the main program.

Line 1100 is a delay loop. The value of the variable T determines the length of the delay. The computer returns to the main program after executing this line.

Line 1190 is used when the player gets the creature. The message is printed on the screen and the computer goes to line 1380 to end the game.

Line 1200 moves the swamp creature across the screen and onto the player. This line is used when the player and the swamp creature occupy the same room. The characters that make up the swamp creature are printed in inverse.

Line 1210 sends the computer to line 1380 to end the game.

Line 1250 is used when the flying creatures are in the same room as the player. The FOR-NEXT loop prints the creature flying across the screen. The character that is on the screen in the position the creature will be printed in is stored in the variable V.

Line 1260 prints the character on the screen and continues the loop. The loop continues until the creature is above the player. The following loop brings the creature down onto the player. Again, the character over which the creature will be printed is stored in variable V.

Line 1270 continues the loop until the creature is on top of the player.

Lines 1280-1300 contain the routine that picks the player up. The creature and the player fly to the top of the screen.

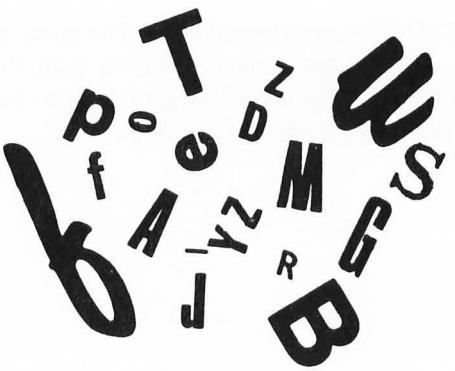
Line 1310 makes a sound and the computer returns to the main program.

Lines 1350-1370 drop the player down into the pits. The entire ROOM$ is filled with the character for this particular door-

way. This pattern fills the screen. The player begins at the top of the screen and falls to the bottom. The computer makes a falling sound.

Lines 1380-1390 end the program. The message is printed on the screen and the computer loops here until the Start key is pressed. When it is, the computer goes back to line 230. To end the game, press the System Reset key.

Chapter 3



Word Games

Whether you enjoy crossword puzzles, anagrams, word searches, or other puzzles involving letters or words, this chapter has several games for the young and old.

JOTTO

Objective of the game: To figure out the word in the fewest number of tries.

Directions: JOTTO is a traditional pencil-and-paper game similar to *Mastermind* or *Bagels*. Instead of playing with numbers or colors, words are used. This makes the game easier for young children, who may be able to deduce the word from the letter combinations rather than by the clues: how many letters are correct and in the correct place.

The game can be played by one or two players. If the game is played by two players, each player receives a different word. There are three levels of play. The words can be four, five, or six letters long.

A question mark appears on the screen and the computer makes a sound to indicate it is ready to receive a word. In the one-player version, the question mark is near the center of the screen. In the two-player version, the question mark for the first player is near the left edge of the screen. Enter a word and press the Return key. The word should be made up of the number of letters chosen at the beginning of the game. If an incorrect letter is entered, press

the Delete key and the letter will be erased. After you have entered your word, the computer compares your word with its word. Two numbers appear on the screen. The first number tells you how many letters are correct and in the correct position. The second number indicates how many letters are correct but in the wrong position.

If this is a two-player game, the second player's question mark appears near the center of the screen. This player should now enter a word. The game continues until the correct word is entered or 10 guesses have been made. When the correct word is entered, the program ends and the computer congratulates the player. If the correct word is not entered within 10 tries, the computer tells you the word. In either case, pressing the Start key will begin the program again. To end the program, press the System Reset key. Figure 3-1 is the flowchart for this program, and Listing 3-1 is the code.

Line 50 sets aside the string space for this program. WORD1$ stores the word for the first player. WORD2$ holds the word for the second player. TEMP$ is a temporary storage space, and GUESS$ holds the word entered by the player.

Line 60 clears the screen and sets the computer for graphics 2 with no text window.

Line 70 asks for the number of players. The question mark at the end of the message is needed because the Input command will not be used. If the question mark were not included, the computer would not print one on the screen.

Line 80 opens the keyboard so a key can be pressed and read without having to press the Return key. The value of the pressed key is stored in the variable P. The keyboard is closed.

Line 90 compares the value of the variable P to 49 and 50. Only two values are acceptable for this question—the 1 and 2. The ASCII value of the 1 key is 49, and the ASCII value of the 2 key is 50. If the value of P is greater or less than these two values, some other key was pressed and the computer repeats line 80 until a correct response is given.

Line 100 prints the number on the screen.

Line 110 subtracts 48 from the value of P to arrive at the actual number pressed. The value of the 1 key is 49, so if the 1 key was pressed, P would contain 49. By subtracting 48 we arrive at the actual number pressed.

Line 120 asks for the level of play. There are three levels of play in this game. The words can be four, five, or six letters long.

Line 130 opens the keyboard so it can be read without using

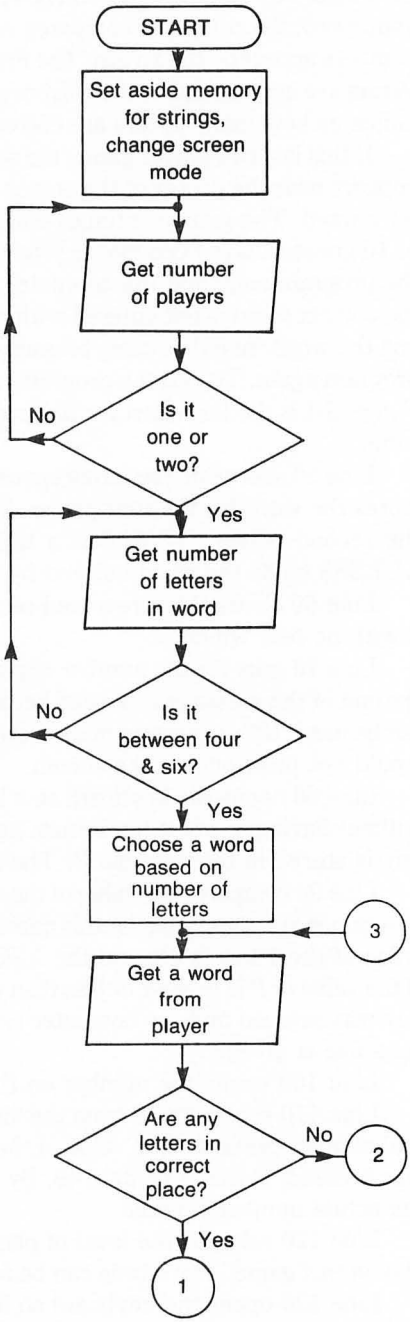
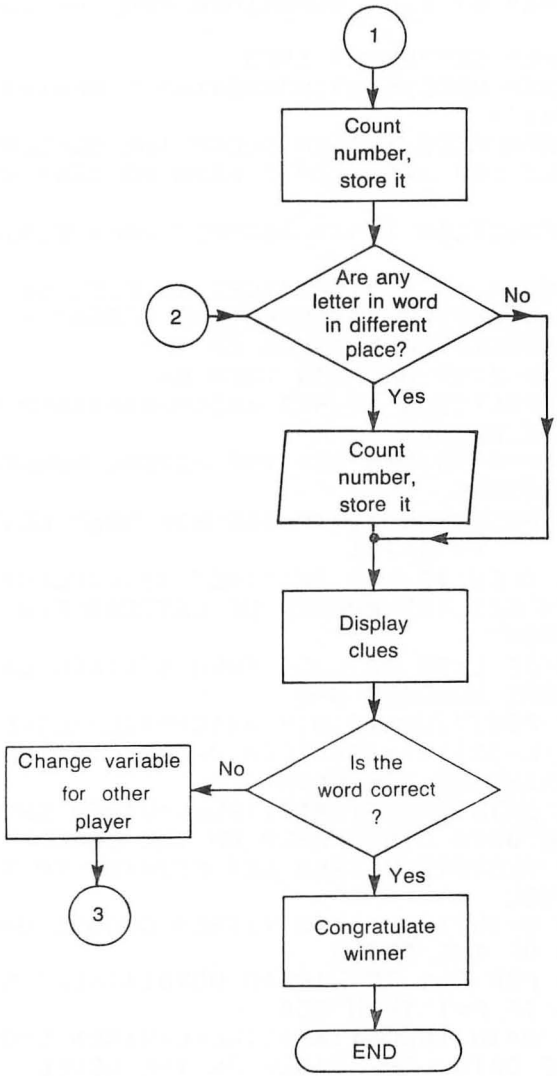


Fig. 3-1. Flowchart for Jotto.


1
Count number, store it
2
Are any letter in word in different place?
No
Yes
Count number, store it
Display clues
Change variable for other player
No
Is the word correct ?
3
Yes
Congratulate winner
END

Listing 3-1. Jotto.

```
10 REM JOTTO - GUESS THE WORD FROM THE
 CLUES
20 REM CHAPTER 3 - WORD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS

40 REM COPYRIGHT 1983
50 DIM WORD1$(6),WORD2$(6),TEMP$(6),GU
ESS$(6)
60 GRAPHICS 18:REM CLEAR THE SCREEN AN
D SET FOR GRAPHICS 2 WITH NO TEXT WIND
OW
70 POSITION 1,4:? #6;"HOW MANY PLAYERS
?";
80 OPEN #2,4,0,"K:":GET #2,P:CLOSE #2:
REM OPEN THE KEYBOARD FOR A READ - STO
RE NUMBER OF PLAYERS IN 'P'
90 IF P<49 OR P>50 THEN 80
100 POSITION 19,4:? #6;CHR$(P):REM SHO
W THE NUMBER
110 P=P-48:REM GET THE ACTUAL NUMBER O
F PLAYERS
120 POSITION 1,7:? #6;"HOW MANY LETTER
S      (4-6)?";
130 OPEN #2,4,0,"K:":GET #2,L:CLOSE #2
:REM GET THE NUMBER OF LETTERS FOR THE
 WORDS
140 IF L<52 OR L>54 THEN 130:REM ONLY
ACCEPT NUMBERS 3-6
150 POSITION 10,8:? #6;CHR$(L):L1=L-48
:L=(L-52)*10+1000:REM CALCULATE THE BE
GINNING OF THE DATA
160 W=INT(RND(1)*10):WL=L+W:REM CHOOSE
 ONE DATA LINE BASED ON THE LEVEL
170 RESTORE WL:REM SET POINTER TO THAT
 LEVEL
180 W=INT(RND(1)*5)+1:REM CHOOSE ONE W
ORD OF THE GROUP
190 FOR R=1 TO W:READ WORD1$:NEXT R
200 IF P=1 THEN 250
210 W=INT(RND(1)*10):WL=L+W:REM CHOOSE
 ONE DATA LINE BASED ON THE LEVEL
220 RESTORE WL:REM SET POINTER TO THAT
 LEVEL
230 W=INT(RND(1)*5)+1:REM CHOOSE ONE W
ORD OF THE GROUP
```

```
240 FOR R=1 TO W:READ WORD2$:NEXT R
250 GRAPHICS 18:REM CLEAR THE SCREEN
260 R=2:IF P=1 THEN C1=5:C2=0:GOTO 280
:REM SET ROW AND COLUMN FOR 1 PLAYER
270 C1=1:C2=11:REM SET COLUMNS FOR 2 P
LAYERS
280 POSITION 2,0:? #6;"ENTER YOUR GUES
S"
285 REM QUESTION MARK IS INVERSE
290 POSITION C1,R:? #6;"?":GOSUB 400
300 C=C1-1:TEMP$=WORD1$:GOSUB 480
310 IF C2 THEN POSITION C2,R:? #6;"?":
GOSUB 400:C=C2-1:TEMP$=WORD2$:GOSUB 48
0
320 R=R+1:IF R<12 THEN 290:REM INCREAS
E 'R' FOR NEXT ROW - ALLOW 10 TRIES
330 GOTO 750
400 FOR X=1 TO 5:SOUND 0,20,10,10:NEXT
 X:SOUND 0,0,0,0:REM FIRST BEEP
410 FOR X=1 TO 10:NEXT X:REM SILENCE
420 FOR X=1 TO 5:SOUND 0,20,10,10:NEXT
 X:SOUND 0,0,0,0:REM SECOND BEEP
430 RETURN
480 FOR X=1 TO L1+1:POSITION C+X,R:REM
 SET CURSOR FOR PROPER POSITON ON SCRE
EN
490 OPEN #2,4,0,"K:":GET #2,K:CLOSE #2
:REM GET A LETTER FROM THE KEYBOARD
500 IF K=155 AND X=L1+1 THEN 570
510 IF K=126 AND X>1 THEN X=X-1:GUESS$
(X,X)=" ":K=32:POSITION C+X,R:GOTO 560

520 IF X=L1+1 THEN 490
530 IF K>127 THEN K=K-128:POKE 694,0:R
EM INVERSE KEY WAS PRESSED - RESET TO
NORMAL VIDEO
540 IF K>96 THEN POKE 702,64:K=K-32:RE
M CAPS/LOWR KEY PRESSED - RESET TO CAP
S
550 IF K<65 OR K>90 THEN 490:REM NOT A
 LETTER - TRY AGAIN
560 ? #6;CHR$(K);:GUESS$(X)=CHR$(K):IF
 K=32 THEN X=X-1
570 NEXT X
580 IF GUESS$=TEMP$ THEN 700
590 RIGHT=0:FOR X=1 TO L1:REM NOW CHEC
```

```
K THE LETTERS
600 IF GUESS$(X,X)=TEMP$(X,X) THEN RIG
HT=RIGHT+1:TEMP$(X,X)=" ":GUESS$(X,X)=
" ":REM REMOVE THE RIGHT LETTER
610 NEXT X:REM GET THE NUMBER FOR LETT
ERS IN THE RIGHT PLACE
620 CLSE=0:FOR X=1 TO L1:FOR Y=1 TO L1
:REM NOW CHECK FOR LETTERS NOT IN THE
RIGHT PLACE
630 IF X=Y THEN 660
640 IF GUESS$(X,X)=" " OR TEMP$(Y,Y)="
 " THEN 660
650 IF GUESS$(X,X)=TEMP$(Y,Y) THEN CLS
E=CLSE+1:TEMP$(Y,Y)=" ":GUESS$(X,X)="
":Y=L1:REM REMOVE THE LETTER
660 NEXT Y:REM CHECK ALL THE LETTERS
670 NEXT X:REM CHECK THE ENTIRE WORD
680 POSITION C+2+L1,R:? #6;CHR$(RIGHT+
48+128);" ";CLSE:REM SHOW THE NUMBERS
690 RETURN
700 GRAPHICS 18:REM CLEAR THE SCREEN
710 POSITION 2,4:? #6;"ConGRatULatIOns
!":REM CONGRATULATE WINNER-nGtUtIs! AR
E INVERSE
720 POSITION 3,6:? #6;"YOU GOT IT!!!":
POSITION 4,8:? #6;GUESS$
730 IF PEEK(53279)<>6 THEN 770:REM WAI
T FOR START TO PLAY AGAIN
740 RUN
750 GRAPHICS 18:POSITION C1,2:? #6;"YO
UR WORD WAS":POSITION C1,4:? #6;WORD1$
:REM SHOW THE WORDS
760 IF C2 THEN POSITION C2-5,8:? #6;"y
our word was":POSITION C2-5,10:? #6;WO
RD2$
770 GOTO 730
1000 DATA OGRE,DINE,ALAS,RAGE,LAZY
1001 DATA WORM,BATH,ROOT,CLAP,WEPT
1002 DATA RIPE,DULL,LOOP,SLID,THIN
1003 DATA HASH,DUTY,CAPE,NEST,CHAT
1004 DATA CORK,SPIN,SANK,DUSK,DOVE
1005 DATA KNEE,HEEL,TUCK,PLUG,POEM
1006 DATA HERD,MINE,LESS,SORT,MOON
1007 DATA HARM,SIZE,SELL,PITY,COST
1008 DATA STAR,DUST,WEST,WORE,BUCK
1009 DATA GASP,PINE,SHOP,TEAR,REAR
```

```
1010 DATA FAVOR,FIGHT,SLEPT,GUIDE,GUAR
D
1011 DATA TIGER,FETCH,CHILD,NORTH,DROO
P
1012 DATA COACH,SPLIT,CLOUD,TOWEL,WITC
H
1013 DATA SKEIN,COUNT,CANDY,HUTCH,STUM
P
1014 DATA QUILT,CREPT,TEMPT,RUSTY,CORA
L
1015 DATA ROBIN,WORST,DROWN,CRUEL,FAUL
T
1016 DATA STUDY,THUMB,BRIEF,PEACE,RINS
E
1017 DATA MARRY,ROYAL,SALAD,BREAD,PROV
E
1018 DATA TRUTH,APRON,AWARE,MAGIC,ANGE
R
1019 DATA EAGER,WORTH,GROUP,SHADE,BRAV
E
1020 DATA EMPIRE,RATHER,PROPER,STREAM,
GOBLIN
1021 DATA PICKET,HOLLOW,KIMONO,BUTLER,
BARLEY
1022 DATA BRAIDS,SELECT,COWARD,PILLOW,
PREFER
1023 DATA LINGER,BUBBLE,TICKLE,THRONE,
CASTLE
1024 DATA MANGER,NAPKIN,CRADLE,ACCEPT,
GREEDY
1025 DATA HUMBLE,SMOOTH,SADDLE,GENTLE,
INSIST
1026 DATA MANAGE,THOUGH,EXCEPT,BOTHER,
UNLESS
1027 DATA VALLEY,BREATH,CATTLE,MOMENT,
BECAME
1028 DATA DIVIDE,STUPID,BOUGHT,OUTFIT,
ADVICE
1029 DATA DOCTOR,FIGURE,CARROT,TWENTY,
COURSE
```

the Return key. The ATASCII value of the pressed key is stored in the variable L.

Line 140 checks the value of L to make sure the key pressed was a 4, 5, or 6. If any other key was pressed, the computer will repeat line 130.

Line 150 prints the number of letters the words will contain on the screen. To get the actual number, 48 is subtracted from the value of L. This is stored in variable L1. The words are stored on DATA lines beginning with line 1000. The four-letter words are on the first 10 lines, the five-letter words on the second set of 10 lines, and six-letter words on the third set of 10 lines. Since the ATASCII value of 4 is 52, we subtract 52 from L, multiply this by 10 and add the result to 1000. If the five-letter words were chosen, the computer would subtract 52 from the value of L (53). This equals 1. Multiplying 1 times 10 equals 10; 10 added to 1000 is 1010—the DATA line holding these words.

Line 160 chooses a random number from 0 to 9. By adding this number to the line number, we arrive at a random data line for the words.

Line 170 uses the RESTORE command to point to that DATA line.

Line 180 chooses a random number from 1 to 5. Each DATA line contains five words. We will use one of these five words.

Line 190 uses a FOR-NEXT loop to count from 1 to the value of W. One word will be read each time this loop is executed. When the computer reaches the value of W, the loop stops and the word the computer is using in this game is stored in the variable WORD1$.

Line 200 checks the value of P. If it is 1, there is only one player for this game and the computer is directed to line 250.

Line 210 chooses a DATA line for the second player. The first line of the words has been calculated. The DATA line the computer uses for the second player's words is stored in the variable WL.

Line 220 uses the RESTORE command to point to the new DATA line.

Line 230 chooses a number from 1 to 5. This is the position of the word in the new DATA line.

Line 240 uses a FOR-NEXT loop again to READ the words in the DATA line until it reaches the word for this player.

Line 250 clears the screen and sets it for graphics 2 without the text window.

Line 260 sets the variable R to indicate on which row the question mark and then the word should be printed. If there is only one player, variable C1 is set to 5 and variable C2 is set to 0. This is the column where the question mark will be printed. The computer is directed to line 280. The next line is used for a two-player game.

Line 270 sets the column variables to 1 and 11 for a two-player game.

Line 280 prints a message at the top of the screen.

Line 290 prints the question mark on the screen. Variable C1 indicates the column position of the question mark, variable R the row. The computer is sent to the subroutine at line 400 to make the prompt sound.

Line 300 decreases the value of C1 by 1 and stores it in variable C. This variable will be used in the next subroutine to point to the column in which the word will begin. The word the computer chose is moved into TEMP$ and the computer is sent to the subroutine at line 480 to get a word from the player.

Line 310 is used when there are two players. If there is only one player, variable C2 is 0. If it is anything but 0, the computer uses this program line. The question mark is printed at the position indicated by variables C2 and R. The computer makes a sound by using the subroutine at line 400. The position of the question mark less one is stored in variable C, the second word the computer chose is moved into TEMP$, and the computer is sent to the subroutine at line 480 to get the second player's guess.

Line 320 increases variable R by 1. The next word is printed on the row. If the value of R is less than 12, the game continues with line 290.

Line 330 sends the computer to line 750 for the end of the game.

Lines 400-430 make the prompt sound. The sound is made in line 400. The FOR-NEXT loop keeps the sound on for a few seconds, then turns it off. The next FOR-NEXT loop is a short pause before the next SOUND command. After the second beep, the computer is sent back to the main program.

Line 480 begins the routine that gets a guess from the player. The FOR-NEXT loop counts from 1 to the number of letters in the word, plus 1. One is added to the number of letters because the last key to be pressed is the Return key; if we counted up to the last letter, the loop would end before the Return key could be pressed. The cursor is positioned at the correct column and row.

Line 490 opens the keyboard and places the value of the key pressed into variable K.

Line 500 compares the value of K to 155. If it is 155 and X is equal to one more than the number of letters in the word, the computer goes to line 570. The value of the Return key is 155. By checking to see if all the letters have been entered, the computer makes sure a short word is not entered for a longer one. Line 570

can only be executed when all the letters have been entered.

Line 510 checks to see if the Delete key has been pressed. If it has been pressed and the value of X is greater than one, the line will continue. If X is 1, no letters have been entered and there is nothing to erase. If letters have been entered, the value of X is decreased by 1. The entered letter is erased from GUESS$, the value of K is set to 32 (a space) and the position is set so the letter on the screen can be erased. The computer is directed to line 560 to erase the letter on the screen.

Line 520 checks to see if any more letters can be entered. If the value of X is greater by 1 than the number of letters needed for the word, the computer is sent to line 490. It can only accept a Delete key or a Return key.

Line 530 checks to see if the value of K is greater than 127. If it is, the ATARI key has been pressed. To correct it, subtract 128 from the value of K and POKE location 694 with a 0.

Line 540 checks to see if an uppercase letter was entered. If the value of K is greater than 96, the key was lowercase. To get the uppercase value, 32 is subtracted from the value of K and the location 702 is POKEd with 64. This resets the keyboard to uppercase.

Line 550 sends the computer to line 490 if the value of K is less than 65 or greater than 90. Only letters are acceptable. Values out of this range indicate that the key pressed was not a letter.

Line 560 prints the pressed letter on the screen. It is stored in GUESS$. If the value of K is 32 (a space), the value of X is decreased by 1.

Line 570 continues the loop.

Line 580 compares the word the player entered (GUESS$) with the word the computer chose (TEMP$). If both words are the same, the computer is sent to line 700 to end the game.

Line 590 sets the variable RIGHT to 0. This counts the number of letters in the player's word that are in the same position as the computer's word. The FOR-NEXT loop starts with the first letter and ends with the last letter of the word.

Line 600 compares a letter in one position with the letter in the same position in the computer's word. If both letters are the same, variable RIGHT is increased by 1. The letter is removed from the computer's string and the player's string.

Line 610 continues the loop until all the letters have been checked.

Line 620 sets variable CLSE to 0. This variable will keep track

of how many letters are the same in both words, but in different positions. Two FOR-NEXT loops are used. The first points at the position in the player's word, while the second points to the computer's word.

Line 630 sends the computer to line 660 if both X and Y are the same value. We already compared the strings letter for this letter; we do not need to do it again.

Line 640 sends the computer to line 660 if either position is empty. No need to compare empty places, either.

Line 650 compares the letter in GUESS$ with the letter in TEMP$. If they are both the same, variable CLSE is incremented by 1. The letters are the same, but in different positions. Variable Y is set to the last position because there is no letter left to compare with the rest of the letters in this word.

Lines 660 and 670 continue the loop until all the letters and positions have been checked.

Line 680 prints the number of letters that are the same and in the same positions on the screen, then prints the number of letters that are the same but in different positions. By adding 48 and 128 to the value of RIGHT, the number in the correct position is printed in blue.

Line 690 returns the computer to the main program.

Lines 700-710 congratulate the winner. The first person to guess the word is the winner.

Lines 720-730 loop until the Start key is pressed. Then the program is run again.

Lines 740-760 show the correct word.

Line 770 sends the computer to line 720 to wait for the Start key. To end the program, press the System Reset key.

Lines 1000-1029 contain the words for this program. These words can be changed as long as four-letter words replace four-letter words, etc.

ROBOTMAN

Objective of the game: To identify correctly the word before the robot disappears.

Directions: This program is similar to the popular *Hangman* game. Children, especially, enjoy the challenge of trying to guess the correct word in the fewest number of tries.

In this game, the letters of the alphabet appear on the screen along with a robotman. The number of question marks on the screen represent the letters in the word. There is an arrow pointing to the

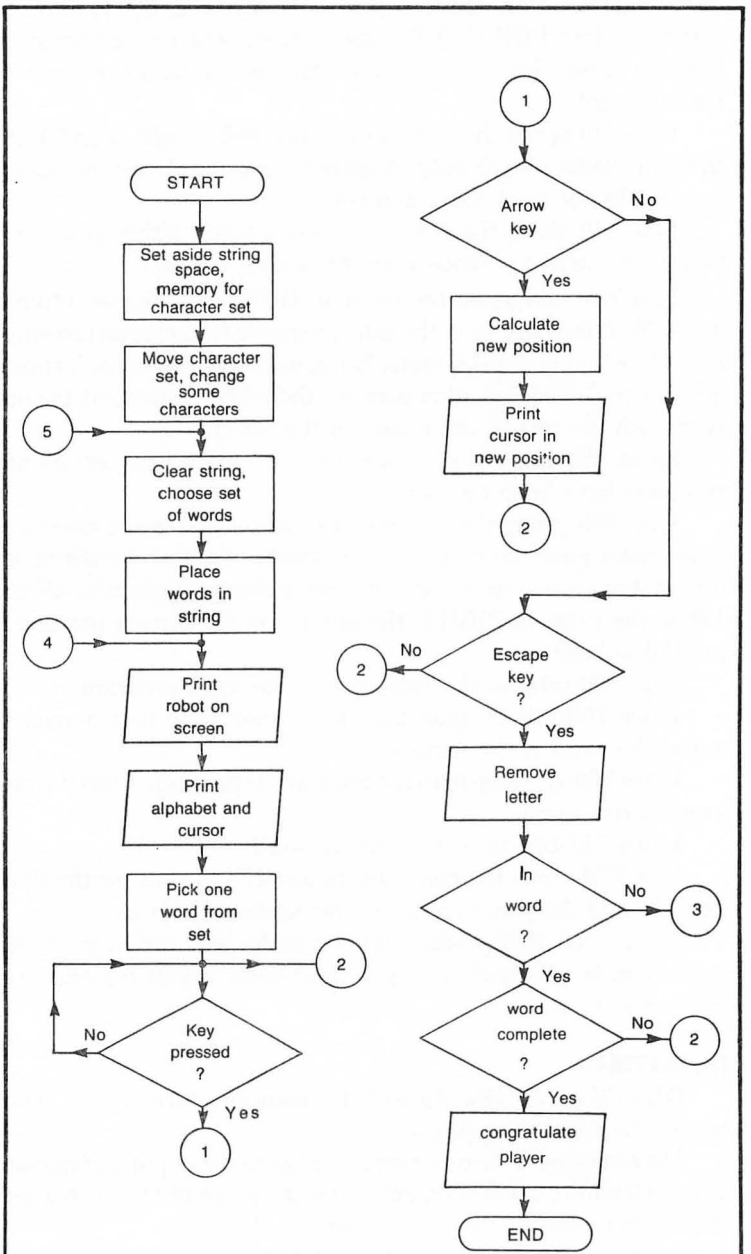
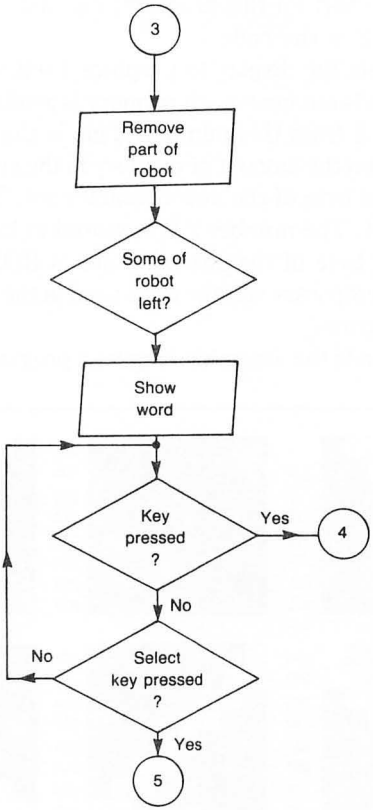


Fig. 3-2. Flowchart for Robotman.



letter A. To move the arrow, press the right arrow key or the left arrow key. When the arrow is pointing to the letter you want to enter, press the Escape key. The letter will disintegrate on the screen. If the letter belongs in the word, it will replace a question mark. If the letter is not part of that word, a portion of the robot will disappear.

If the entire robot disappears, the correct word will appear on the screen just above the question marks. If the entire word appears on the screen within 10 tries, a message appears on the screen.

To continue the game with the same set of words, press any key. To continue the game with a different set of words, press the Select key on the computer.

The number of words you enter correctly and the number of

words you miss are printed near the bottom of the screen. Figure 3-2 is the flowchart for this program; Fig. 3-3 is the character set, and Listing 3-2 is the code.

Line 60 sets the display to graphics 1 without a text window.

Line 70 finds out how much memory is available on your system and subtracts 8 from this number. This is the equivalent of subtracting 2K from the amount of memory in the system. This number will be the first byte of the new character set. This value is stored in location 204. The number 224 is stored in location 206. This is the beginning byte of the character set in ROM. These memory locations are temporary storage units used in the following assembly language program.

Line 80 reads the assembly language program from line 90 into

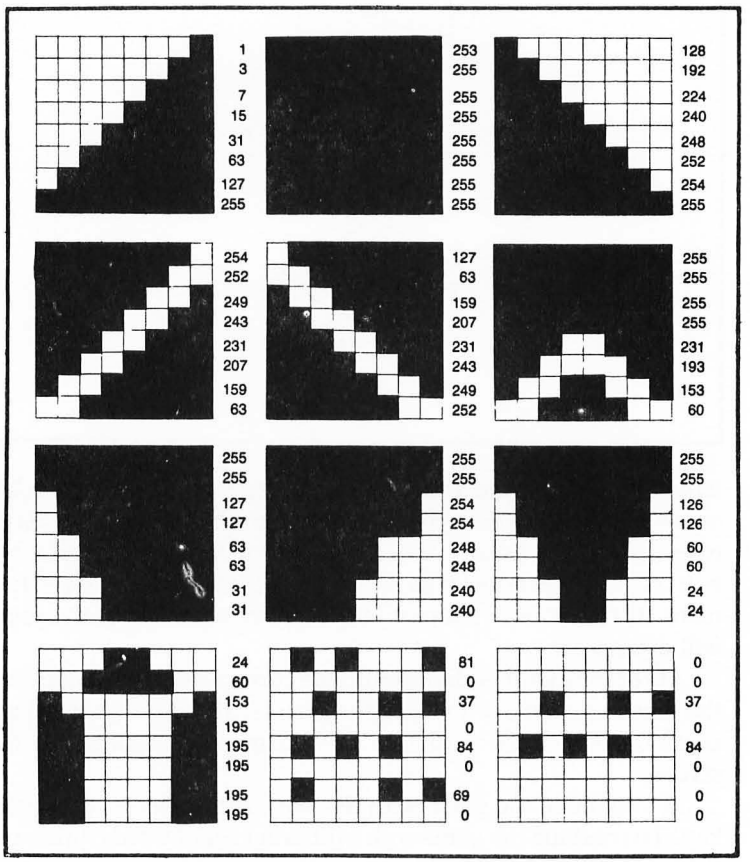


Fig. 3-3. Character set for Robotman.

Listing 3-2. Robotman.

```
10 REM ROBOTMAN - A VARIATION OF HANGM
AN
20 REM CHAPTER 3 - WORD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM WORD$(10),BASE$(100),L$(1),ANSW
ER$(10)
60 GRAPHICS 17:REM CLEAR SCREEN - GRAP
HIC 1/NO TEXT WINDOW
70 A=PEEK(106)-8:POKE 204,A:POKE 206,2
24:REM PLACE NEW CHARACTER BASE 1K ABO
VE DISPLAY LIST
80 FOR X=1536 TO 1555:READ B:POKE X,B:
NEXT X:REM MOVE THE MACHINE LANGUAGE S
UBROUTINE
90 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
100 Q=USR(1536):REM RUN THE MACHINE LA
NGUAGE SUBROUTINE
110 POKE 756,A:REM TELL THE COMPUTER W
HERE THE CHARACTER SET IS
120 CHBASE=A*256:REM DECIMAL ADDRESS O
F CHARACTER BASE
130 FOR X=24 TO 119:READ B:POKE CHBASE
+X,B:NEXT X:REM READ IN THE NEW CHARAC
TER SET
140 DATA 1,3,7,15,31,63,127,255
150 DATA 255,255,255,255,255,255,255,2
55
160 DATA 128,192,224,240,248,252,254,2
55
170 DATA 254,252,249,243,231,207,159,6
3
180 DATA 127,63,159,207,231,243,249,25
2
190 DATA 255,255,255,255,231,195,153,6
0
200 DATA 255,255,127,127,63,63,31,31
210 DATA 255,255,254,254,248,248,240,2
40
220 DATA 255,255,126,126,60,60,24,24
230 DATA 24,60,153,195,195,195,195,195
240 DATA 81,0,37,0,84,0,69,0
```

```
250 DATA 0,0,37,0,84,0,0,0
260 W=INT(RND(1)*10):BASE$(1)=" ":BASE
$(100)=" ":BASE$(2)=BASE$:WORD$=BASE$:
REM CLEAR THE OLD WORDS FROM STRINGS
270 RESTORE 1000+W*10
280 FOR X=1 TO 91 STEP 10:READ ANSWER$
:REM GET THE WORDS FROM THE LIST
290 BASE$(X)=ANSWER$:NEXT X:REM PLACE
THE WORDS IN THE STRING
300 POSITION 12,7:? #6;"$$$$%":POSITIO
N 12,8:? #6;"$&$'$":POSITION 12,9:? #6
;"$$($$"
310 POSITION 12,10:? #6;")$$$*":REM PR
INT THE HEAD
320 POSITION 13,11:? #6;"$$%":POSITION
 11,12:? #6;"$$$$$$%":POSITION 11,13:?
 #6;"$ $$$ $"
330 POSITION 11,14:? #6;"$ &&& $":POSI
TION 11,15:? #6;", $$$ ,":POSITION 13,
16:? #6;"$$$"
340 POSITION 13,17:? #6;"$ $":POSITION
 13,18:? #6;"$ $":POSITION 13,19:? #6;
"+ +":REM PRINT ARMS AND TORSO
350 REM PRINT THE ALPHABET AND SCORES.
  ROBOT IS IN INVERSE LOWER CASE, HUMA
N IN LOWER CASE
360 POSITION 2,0:FOR X=65 TO 77:? #6;C
HR$(X+128);:NEXT X:REM PRINT HALF OF A
LPHABET
370 POSITION 2,1:? #6;"^":REM PRINT AR
ROW - INVERSE VIDEO
380 POSITION 2,2:FOR X=78 TO 90:? #6;C
HR$(X+128);:NEXT X:REM PRINT THE REST
OF THE ALPHABET
390 POSITION 1,22:? #6;"robot ";RS:POS
ITION 11,22:? #6;"human ";HS:REM robot
 IS LOWER CASE INVERSE
400 W=INT(RND(1)*10)*10+1:E=9:E1=10:IF
 W=91 THEN E=LEN(BASE$)-W:E1=E:REM CHO
OSE A WORD FROM THE BASE
410 ANSWER$=BASE$(W,W+E):FOR X=1 TO E1
:IF ANSWER$(X,X)=" " THEN ANSWER$=ANSW
ER$(1,X-1):GOTO 430
420 NEXT X
430 WL=LEN(ANSWER$):FOR X=1 TO WL:POSI
TION X,15:? #6;"?":NEXT X:R=1:C=2:MI=1
```

```
0:REM PRINT THE ? - SET POINTERS
440 OPEN #2,4,0,"K:"
450 GET #2,K:CLOSE #2:IF K>127 THEN K=
K-128:POKE 694,0:REM CLEAR THE BUFFER
AND CLOSE IT
460 IF K=42 THEN 500
470 IF K=43 THEN 540
480 IF K=27 THEN 570
490 GOTO 440
500 POSITION C,R:? #6;" ":REM REMOVE T
HE POINTER
510 C=C+1:IF C=15 THEN C=2:R=R+2:IF R=
5 THEN R=1:REM SET NEW POSITION
520 POSITION C,R:? #6;"^":REM MOVE THE
 POINTER - ARROW IS INVERSE
530 GOTO 440:REM GET ANOTHER MOVE
540 POSITION C,R:? #6;" ":REM REMOVE T
HE POINTER
550 C=C-1:IF C=1 THEN C=14:R=R-2:IF R=
-1 THEN R=3:REM SET NEW POSITION
560 GOTO 520
570 LOCATE C,R-1,L:POSITION C,R-1:? #6
;CHR$(L);:IF L=32 THEN 440:REM ONLY TA
KE LETTERS THAT ARE THERE
580 POSITION C,R-1:? #6;"-":FOR V=14 T
O 7 STEP -1:FOR T=100 TO 50 STEP -5:SO
UND 0,7,V,10:NEXT T:NEXT V
590 POSITION C,R-1:? #6;".":FOR V=7 TO
 0 STEP -1:FOR T=50 TO 0 STEP -5:SOUND
 0,T,V,10:NEXT T:NEXT V
600 SOUND 0,0,0,0:POSITION C,R-1:? #6;
" ":REM REMOVE THE LETTER
610 L=L-128:M=0:FOR X=1 TO WL:REM CHEC
K THE LETTERS
620 IF L<>ASC(ANSWER$(X,X)) THEN 650:R
EM NO MATCH
630 POSITION X,15:? #6;CHR$(L):WORD$(X
,X)=CHR$(L):M=1
640 FOR S=250 TO 5 STEP -5:SOUND 0,S,1
0,8:NEXT S:SOUND 0,0,0,0
650 NEXT X:IF M AND ANSWER$=WORD$(1,WL
) THEN 840
660 IF  NOT M THEN GOSUB 700+(MI*10):M
I=MI-1
670 IF MI THEN 440
680 POSITION 1,12:? #6;ANSWER$:FOR S=1
```

```
00 TO 200:SOUND 0,S,6,8:NEXT S:SOUND 0
,0,0,0
690 RS=RS+1:GOTO 850
700 REM REMOVE THE BODY
710 FOR PX=7 TO 10:POSITION 12,PX:? #6
;"     ";:NEXT PX:RETURN
720 POSITION 13,8:? #6;"$$$":RETURN
730 POSITION 14,9:? #6;"$":RETURN
740 FOR PX=11 TO 16:POSITION 13,PX:? #
6;"   ";:NEXT PX:RETURN
750 FOR PX=12 TO 14:POSITION 16,PX:? #
6;"  ";:NEXT PX:RETURN
760 FOR PX=12 TO 14:POSITION 11,PX:? #
6;"  ";:NEXT PX:RETURN
770 FOR PX=17 TO 18:POSITION 13,PX:? #
6;" ";:NEXT PX:RETURN
780 FOR PX=17 TO 18:POSITION 15,PX:? #
6;" ";:NEXT PX:RETURN
790 POSITION 11,15:? #6;"  $$$  ":RETU
RN
800 POSITION 13,19:? #6;"   ";:RETURN
830 REM OrC! ARE INVERSE
840 POSITION 1,12:? #6;"COrrECt!":FOR
X=1 TO WL:SOUND 0,ASC(WORD$(X,X)),10,1
0:NEXT X:HS=HS+1
850 SOUND 0,0,0,0:POKE 764,255:POSITIO
N C,R:? #6;" ";:WORD$="           ":REM
 CLEAR OUT THE POINTER AND THE STRING
860 IF PEEK(764)<>255 THEN LINE=300:GO
TO 890
870 IF PEEK(53279)=5 THEN LINE=260:GOT
O 890
880 GOTO 860
890 POKE 764,255:POSITION 1,12:? #6;WO
RD$:POSITION 1,15:? #6;WORD$;:GOTO LIN
E
1000 DATA MAJESTY,DINE,ALAS,OGRE,FAVOR
,COTTAGE,EMPIRE,RATHER,SURROUNDED,SLIP
PED
1010 DATA PROPER,SCRATCH,REMARKABLE,RA
GE,BLINDED,NAP,STREAM,SAWMILL,GOBLIN,F
IFTEENTH
1020 DATA ABSOLUTELY,PLAYMATE,LAZY,PIC
KETS,JAR,FIGHT,HOLLOW,SLEPT,STOUTNESS,
GOODBYE
1030 DATA WORMS,KIMONO,BATH,GUIDE,ROOT
```

```
S,HIPS,BUTLER,BADE,GUARD,BARLEY
1040 DATA TIGER,CLAP,GRANTED,PEARLS,DA
RED,FETCH,CHILD,BRAIDS,TERRIFIED,SEIZE
D
1050 DATA WEPT,MUTTERING,HELPLESSLY,SE
LECTED,RIPE,BID,DULL,RATS,UTTERLY,SPOO
L
1060 DATA COWARDS,LOOP,PILLOW,SCUFFLIN
G,NORTH,PADDY,FUMED,SLID,DROOP,PAINFUL
1070 DATA WITS,COACH,SHIVERING,HALLO,S
PLIT,CLOUD,FORTUNATE,PREFER,TOWEL,WITC
H
1080 DATA COMFORT,HEALTHY,SWEETLY,SKEI
N,SNIP,SOBBING,ANT,LINGERED,COUNT,THIN
1090 DATA CANDY,HUTCH,BUBBLES,HASH,STU
MP,QUILT,MUGS,CARELESSLY,SLENDER,DUTY
```

a free area of RAM. Each byte is read from the data line and POKEd into a memory location. This subroutine is completely relocatable. In this program it is stored on page 6. It can be stored in any other convenient area of memory.

Line 100 executes the assembly language subroutine. This subroutine moves the character set from ROM into RAM.

Line 110 POKEs the high-order byte of the beginning address of the character set into location 756. Now the computer will use the character set in RAM instead of the ROM set.

Line 120 calculates the decimal address of the character set. Since the character set must begin on an even boundary, the high-order byte is multiplied by 256.

Line 130 reads the bytes from the following DATA lines and POKEs them into RAM. Each DATA line is one new character. Each character is made up of eight bytes. The characters from the pound sign to the period, inclusive, will be changed. See Fig. 3-4 for the new character set.

Line 260 uses the random number command to pick a number from one to ten. This number is the set of words that will be used in the program. BASE$ is cleared and WORD$ is cleared. If the strings are not cleared before they are used, data from previous programs could be stored in them.

Line 270 points to the DATA line that contains the set of words for this session.

Line 280-290 read the words from the DATA line and place them into ANSWER$. The words are then transferred into BASE$.

Each word is given 10 bytes or places in the string.

Line 300-340 print the robot on the screen. Be sure to place the spaces where indicated, or the robot will not look right on the screen.

Lines 370 and 390 print the alphabet on the screen. The alphabet will appear in two rows on the screen with a blank line between the first and second row. The number of words guessed and the robot's score also appear on the screen.

Line 380 positions the up arrow under the A.

Line 400 chooses a number from 1 to 10. This number is the word chosen from the word set that will be used in this turn.

Line 410 stores the word in ANSWER$. The line looks for the space or end of word so the string will contain only the letters for that word.

Line 430 finds the length of ANSWER$ and prints a question mark for every letter in the word.

Line 440 opens the keyboard for a read.

Line 450 gets the ATASCII value of the key pressed and closes the keyboard. If the value of K is greater than 127, the ATARI key is pressed. By subtracting 128 from the value of K, we know which key was pressed. POKEing location 694 with 0 restores the keyboard to normal key values.

Lines 460-490 compare the value of K to the two arrow keys and the Escape key. If a match is made, the program goes on to the correct line number. If a match is not made, the program will loop back and wait for another keystroke.

Line 500 erases the arrow from its position.

Line 510 increases the value of C by 1. C is the column the arrow is in. When this value reaches 15, the arrow would not be under a letter, so C is reset to 2 and variable R is increased by 2. Variable R contains the row the arrow is in. If this value reaches 5, the variable is set back to 1. There are only two rows that the arrow can be in, the first row or the second.

Line 520 prints the arrow in the new position. This gives a wrap around effect on the screen.

Line 530 sends the computer back to line 440 to wait for another keystroke.

Lines 540-560 move the arrow in the opposite direction.

Line 570 begins the routine for the Escape key. By pressing the Escape key, you are telling the computer that this is the letter you want to try. The Locate command checks the screen just above the arrow to see if, in fact, a letter is there. If the space is blank,

the program goes back to line 440 and waits for another key to be pressed.

Lines 580-590 remove the letter from the screen. The new characters for the dash and period and the sound routines make the letter look like it is disintegrating.

Line 600 turns the sound generator off and erases the letter on the screen.

Line 610 begins the FOR-NEXT loop that checks the "word" for the letter just entered.

Line 620 individually checks each letter in the word. If the entered letter does not match this letter, the computer is directed to line 650 and the loop continues.

Line 630 replaces the question mark on the screen with the entered letter and places that letter into WORD$. Variable M is used as a flag and is set to 1 when a match has been made.

Line 640 is the sound routine that makes the letter sound like it popped onto the screen.

Line 650 continues the FOR-NEXT loop for the entire word. If the flag (M) was set to 1 and the word in ANSWER$ matches the word in WORD$, the score is increased by 1, a short tune is played and the computer continues with line 850.

Line 660 directs the computer to a line between 700 and 800. This line will erase part of the robot.

Line 670 checks the value of M. If it is not 0, the program continues with line 440.

Line 680 prints the word in ANSWER$ on the screen and makes a sound. This line is executed after the entire robot is erased.

Line 690 increases the computer's score and sends the computer to line 850.

Lines 700-800 remove parts of the robot. Each incorrect letter that is entered removes one part of the robot.

Line 850 removes the letters from WORD$ string. This prepares it for the next turn.

Line 860 checks location 764. If any key has been pressed, variable LINE is set to 300 and the computer goes to line 890.

Line 870 checks the location 53279. If it is 5, the Select key has been pressed. Variable LINE is set to 260 and the computer continues with line 890.

Line 880 sends the computer back to line 860. No key has been pressed.

Line 890 resets location 764 by POKEing it with 255. If the answer was printed on the screen, it is removed, and the computer

is directed to the line number stored in LINE. If variable LINE is equal to 300, the next word will be chosen from the same set of words. If variable LINE is equal to 260, a new set of words will be chosen. Otherwise, another word from the same set will be chosen.

Lines 1000-1090 contain the words that can be used in this program. To change the words, simply type 10 new words in a line. The line number *must* be one of the lines that appear in this part of the listing. Be sure the line contains 10 words and there is a comma separating each word.

FRACTURED STORIES

Objective of the game: To create a new story.

Description: This game is a favorite with most children. Whether they play it with the computer or use one of the many popular books on the market, children love trying to come up with stories that are sillier than the last.

In this program there are three different stories from which to choose. The first is a Mother Goose rhyme, the second a space story. The third story is about a place the child can visit. To select a story, enter a story number and press the Return key. The screen clears and a description appears on the screen. This may be a part of speech, or a type of word the story needs. A slash after a word gives more information about the word that is needed. Here are a few selections:

verb/pt - verb part tense
verb/s - end with an "s"
verb/ing - end with "ing"

Each story requires between seven and 12 words. Once all the words have been entered, the screen clears and the story is printed on the screen. To return to the menu, press the Start key. Figure 3-4 is the flowchart for this program, and Listing 3- 3 is the code.

Line 50 sets aside the string space required for this program. WORD$ holds the words entered by the user, D$ is used for the description of the word that should be entered, and A$ temporarily stores the word entered by the user.

Line 60 clears the string. If a string is not cleared before it is used, it may contain garbage. The first element of the string is set to a space, then the last element of the string is set to a space, then the last element of the string is set to a space. By setting the sec-

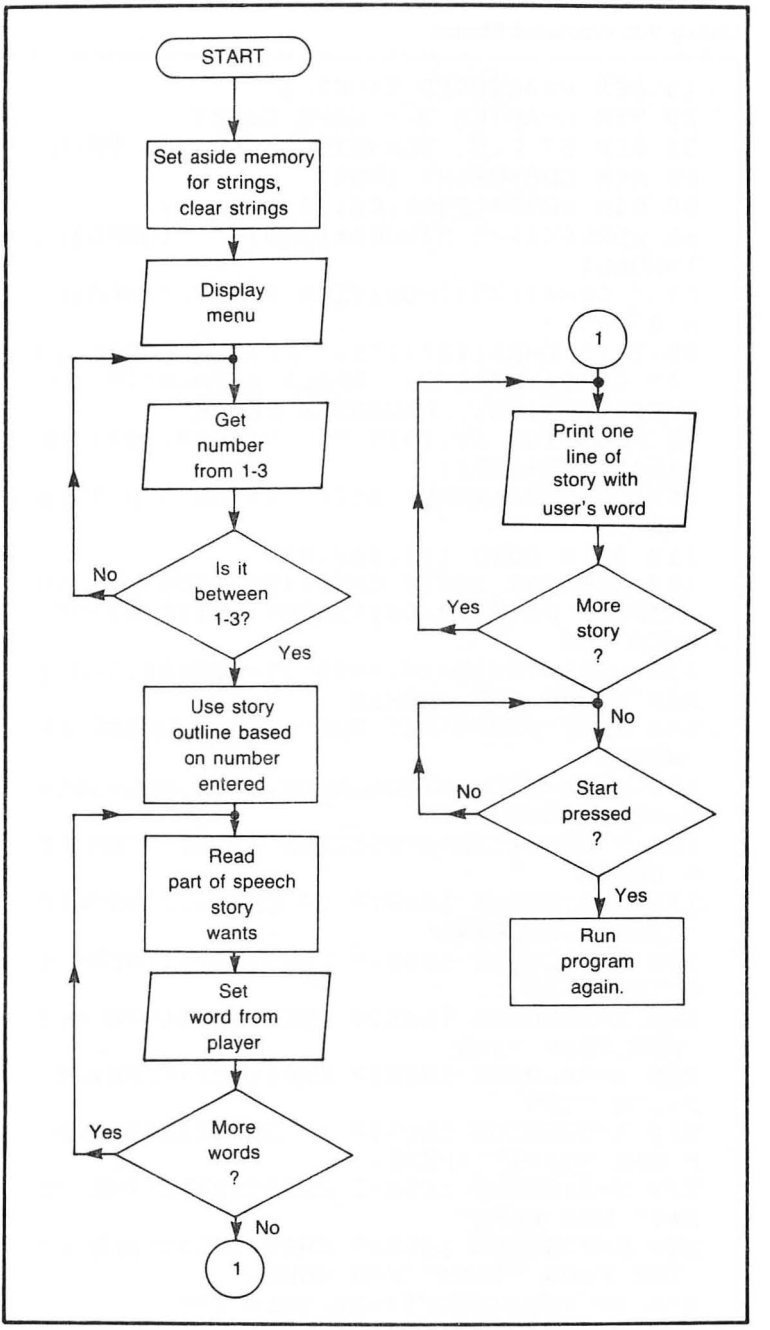


Fig. 3-4. Flowchart for Fractured Stories.

Listing 3-3. Fractured Stories.

```
10 REM FRACTURED STORIES
20 REM CHAPTER 3 - WORD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1984
50 DIM WORD$(240),D$(20),A$(15)
60 WORD$(1)=" ":WORD$(240)=" ":WORD$(2
)=WORD$
70 ? CHR$(125):POSITION 3,5:? "CHOOSE
A STORY -"
80 ? :? CHR$(127);"1.  MOTHER GOOSE":?
 :? CHR$(127);"2.  SPACE ATTACK":? :?
CHR$(127);"3.  FAVORITE PLACE"
90 POSITION 19,14:? "   ";CHR$(30);CHR
$(30);CHR$(30);
100 TRAP 90:INPUT S:IF S<1 OR S>3 THEN
 90
110 ON S GOTO 120,260,430
120 RESTORE 150:? CHR$(125):FOR X=1 TO
 7:READ D$:? :? D$;:INPUT A$:IF A$=""
THEN 120
130 WORD$(X*15-14,X*15-15+LEN(A$))=A$:
REM STORE THE ANSWER
140 NEXT X:REM GET THE RIGHT NUMBER OF
 WORDS
150 DATA OCCUPATION,NOUN,NOUN,NOUN,VER
B,NOUN,NOUN
160 ? CHR$(125):POSITION 13,2:? "MOTHE
R GOOSE"
170 X=1:GOSUB 1000:? :? CHR$(127);"OLD
 ";A$;" HUBBARD"
180 X=2:GOSUB 1000:? CHR$(127);"WENT T
O THE ";A$
190 X=3:GOSUB 1000:? CHR$(127);"TO GET
 HER POOR ";A$
200 X=4:GOSUB 1000:? CHR$(127);CHR$(12
7);"A ";A$
210 X=5:GOSUB 1000:? :? CHR$(127);"WHE
N SHE ";A$;" THERE,"
220 X=6:GOSUB 1000:? CHR$(127);"THE ";
A$;" WAS BARE"
230 X=7:GOSUB 1000:? CHR$(127);"AND SO
 THE POOR ";A$;"HAD NONE."
240 IF PEEK(53279)<>6 THEN 240
250 RUN
```

```
260 RESTORE 290:? CHR$(125):FOR X=1 TO
 11:READ D$:? :? D$;:INPUT A$:IF A$=""
 THEN 260
270 WORD$(X*15-14,X*15-15+LEN(A$))=A$:
REM STORE THE ANSWER
280 NEXT X:REM GET THE RIGHT NUMBER OF
 WORDS
290 DATA VERB/PT,NOUN,VERB/PT,COLOR,VE
RB/ING,NOUN,NUMBER,ADVERB,NOUN,VERB/PT
,NOUN
300 ? CHR$(125):POSITION 13,2:? "SPACE
 ATTACK"
310 X=1:GOSUB 1000:? :? CHR$(127);"ONE
 NIGHT I ";A$
320 X=2:GOSUB 1000:? "INTO THE DARK ";
A$;".  I DID"
330 X=3:GOSUB 1000:? "NOT BELIEVE WHAT
 I ";A$;"."
340 X=4:GOSUB 1000:? :? A$;" MARTIANS
WERE ";:X=5:GOSUB 1000:? A$
350 X=6:GOSUB 1000:? "FROM THE ";A$;".
"
360 X=7:GOSUB 1000:? "THERE MUST HAVE
BEEN ";A$;" OF THEM."
370 X=8:GOSUB 1000:? :? "I WAS ";A$;".
  I WANTED"
380 X=9:GOSUB 1000:? "TO LEAVE MY ";A$
;"."
390 X=10:GOSUB 1000:? :? "THEN I ";A$;
"."
400 X=11:GOSUB 1000:? :? CHR$(127);"IT
 WAS ONLY A ";A$;"!"
410 IF PEEK(53279)<>6 THEN 410
420 RUN
430 RESTORE 460:? CHR$(125):FOR X=1 TO
 12:READ D$:? :? D$;:INPUT A$:IF A$=""
 THEN 430
440 WORD$(X*15-14,X*15-15+LEN(A$))=A$:
REM STORE THE ANSWER
450 NEXT X:REM GET THE RIGHT NUMBER OF
 WORDS
460 DATA PLACE,SEASON,NOUN,NOUN,VERB,N
OUN,ADJECTIVE,NOUN,VERB,NOUN,VERB/S,VE
RB
470 X=1:GOSUB 1000:? CHR$(125);"I LIKE
 TO GO TO THE ";A$;" IN ";:X=2:GOSUB 1
```

```
000:? A$;","
480 X=3:GOSUB 1000:? "THE ";A$;" MAKES
 WAVES ON"
490 ? "THE ";:X=4:GOSUB 1000:? A$;","
500 X=5:GOSUB 1000:? :? "THE FISH LIKE
 TO ";A$;","
510 X=6:GOSUB 1000:? "THE ";A$;:X=7:GO
SUB 1000:? " IS ";A$;","
520 X=8:GOSUB 1000:? :? "SOMETIMES I S
IT ON THE ";A$;","
530 X=9:GOSUB 1000:? "OTHERTIMES I ";A
$;" IN A ROWBOAT,"
540 X=10:GOSUB 1000:? "THE ";A$;" ";:X
=11:GOSUB 1000:? A$;" MY BODY,"
550 X=12:GOSUB 1000:? "BUT I DON'T ";A
$;","
560 ? :? CHR$(127);"IT'S FUN!!"
570 IF PEEK(53279)<>6 THEN 570
580 RUN
1000 A$=WORD$(X*15-14,X*15)
1010 FOR V=1 TO 15:IF A$(V,V)<>" " THE
N NEXT V:REM FIND THE END OF THE WORD
1020 V=V-1:A$=WORD$(X*15-14,X*15-15+V)
:RETURN
```

ond element of the string to the string, the entire string becomes spaces.

Lines 70-90 print the menu on the screen. The screen is cleared, the menu is printed, and the position for the question mark is erased. Be sure to include all the semicolons on line 90: these semicolons keep the cursor on the same line. The question mark will be printed on this line.

Line 100 places the question mark on the screen and waits for a number. The TRAP command keeps the program from crashing if a letter is entered instead of a number. The computer repeats line 90. When a number is entered, it is stored in variable S. This variable is tested to see if it contains a number between one and three. If it does not, the computer remains on this line until a correct number is entered.

Line 110 sends the computer to the correct program line for the chosen story.

Line 120 begins the Mother Goose rhyme. The RESTORE command tells the computer to use the DATA on line 150. The screen is cleared and the computer begins the FOR-NEXT loop.

This story asks for seven different words. The description of the needed word is read from line 150 and stored in variable D$. This description is printed on the screen. There are two PRINT commands (in the ? shorthand), separated by a colon, before the second D$. This skips a row on the screen between the words, making it easier to read. The entered word is stored in A$. If the Return key is pressed and no word is entered the computer is directed to the beginning of this line, and an entire set of new words can be entered.

Line 130 places the word stored in A$ into the correct position in WORD$. There are 15 positions or spaces available for every word entered. One is subtracted from the value of X. This new value is multiplied by 15. Then 1 is added to the product. This is the first element for the word entered. We need to subtract 1 from the value of X before it is multiplied by 15, and add it back after the multiplication because the word ends with a value that is a multiple of 15, but begins one space after the last character of the previous word.

Line 140 continues the loop until all seven words have been entered.

Line 150 contains the descriptions of the words to be entered.

Lines 160-230 print the story on the screen. Variable X indicates which word in the string is used in the sentence. The computer uses the subroutine at line 1000 to get the correct word. Be sure to use a semicolon before and after A$. Otherwise, the rest of that line will be printed in the next row on the screen.

Line 240 waits until the Select key is pressed.

Line 250 runs the program again.

Line 260 begins the second story. The RESTORE command tells the computer that the information for the descriptions of the words are found on line 290. The screen clears and the program asks for the words in the same manner as it did for the first story.

Line 270 places the word entered into the correct position of WORD$.

Line 280 continues the loop until all 11 words are entered.

Line 290 contains the data for the descriptions of the words to be entered.

Lines 300-400 print the story on the screen. Variable X indicates which word is used in the line. Be sure to include the spaces and semicolons.

Line 410 loops until the Start key is pressed.

Line 420 runs the program again.

Line 430 begins the third story. The data for this story is on line 460. The same routine is used to read the data and enter the words.

Line 440 places the word entered into WORD$.

Line 450 continues the loop until all 12 words are entered.

Line 460 contains the word descriptions read by line 430.

Lines 470-560 print the new story on the screen.

Line 570 loops until the Start key is pressed.

Line 580 runs the program again.

Lines 1000-1020 use the value of variable X to find the correct word in WORD$. One is subtracted from X because the new word will begin in the next space after the area set aside for the previous word. The word will end at the value of X times 15. If the entire 15 characters placed in A$ were used, unnecessary spaces would be printed on the screen. The FOR-NEXT loop looks for the first space. This position minus one is the end of the word. The word without the extra spaces is stored in A$. The routine returns to the line that called it.

DECODER

Objective of the game: Test your ability to decipher the code created by the computer.

Directions: The computer takes a well-known expression and codes it. Every letter has another letter substituted for it. At the more difficult level, every letter will have a graphics character substituted. Talk about hieroglyphics! The program can be played by two players, each taking turns trying to decipher his message.

To play the game, enter whether one or two persons will be playing. The computer chooses a quotation and ciphers it. The encoded message is printed on the screen. On each player's turn, an up arrow is printed under the first letter or character of the encrypted quotation. Press the right arrow or left arrow key to move the up arrow under a different letter or character. When the up arrow is pointing to the letter or character you want to change, press the space bar. A question mark appears above that letter. Press the letter key you want substituted. If you press the space bar, then change your mind, press the Return key and the up arrow will appear under the letter. If you want to erase all the letters and start again, press the Escape key. A letter can be changed more than once, but the same letter should not be substituted for two different letters or characters. Figure 3-5 is the flowchart for this program, and Listing 3-4 shows the code.

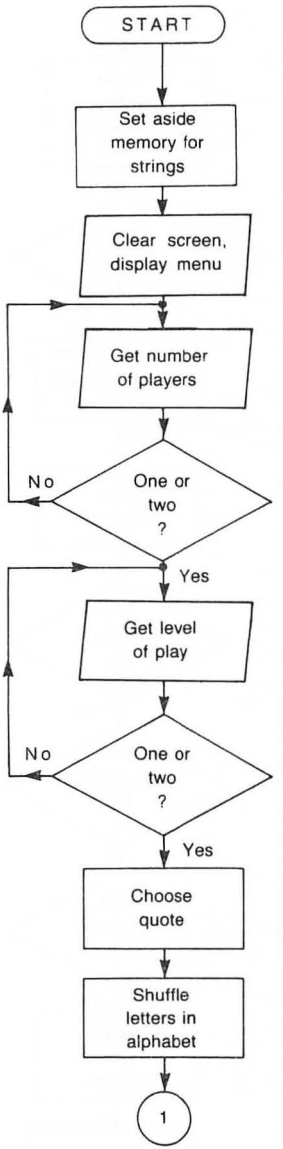
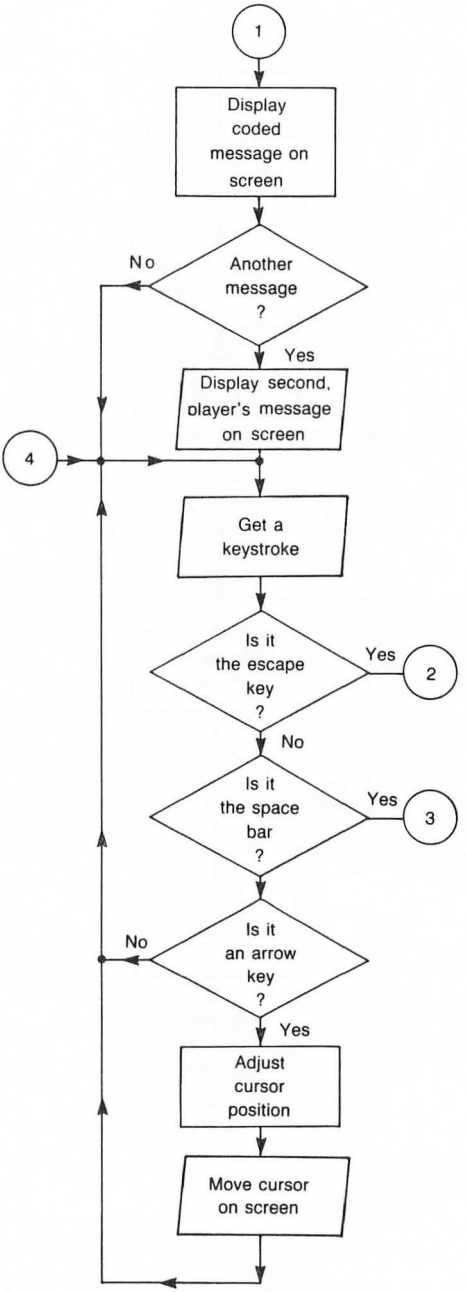
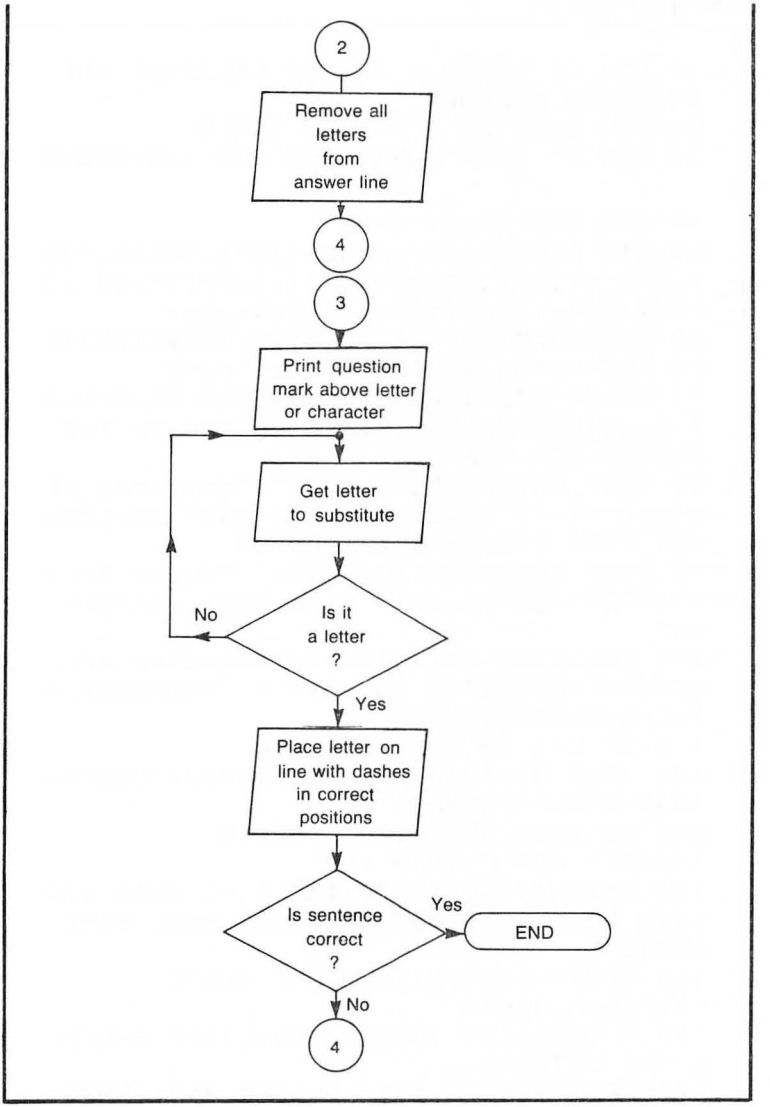


Fig. 3-5. Flowchart for Decoder. (Continued through page 165.)


1
Display coded message on screen
No
Another message ?
Yes
Display second, player's message on screen
4
Get a keystroke
Is it the escape key ?
Yes
2
No
Is it the space bar ?
Yes
3
No
Is it an arrow key ?
Yes
Adjust cursor position
Move cursor on screen



Line 50 sets aside string space for the program. MES1$ and MES2$ store the messages that will be encrypted. ANS1$ and ANS2$ store your answer to the message. SCRM1$ and SCRM2$ hold the scrambled message, ALPH$ contains the alphabet, and TEMP$ and TEMP2$ are temporary strings used to move the message from one string to another.

Line 60 sets the computer for graphics 0, removes the cursor,

Listing 3-4. Decoder.

```
10 REM DE CODER - TRY TO DECIPHER THE
ENCRYTED MESSAGE
20 REM CHAPTER 3 - WORD GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS

40 REM COPYRIGHT 1983
50 DIM MES1$(100),MES2$(100),ANS1$(100
),ANS2$(100),SCRM1$(100),SCRM2$(100),A
LPH$(26),TEMP$(100),TEMP2$(100)
60 GRAPHICS 0:POKE 752,1:? CHR$(125):R
EM CLEAR THE SCREEN - TEXT MODE
70 COLOR (42):PLOT 5,5:DRAWTO 35,5:PLO
T 6,18:DRAWTO 35,18:REM DESIGN ON THE
SCREEN FOR MENU
80 TRAP 80:POSITION 7,7:? "How many pl
ayers (1-2)  ";CHR$(126);CHR$(126);:IN
PUT P:IF P<1 OR P>2 THEN 80
90 TRAP 100:POSITION 7,9:? "Which leve
l":POSITION 10,11:? "1-Letter encrypti
on"
100 POSITION 10,13:? "2-Character encr
yption":POSITION 15,15:? "  ";CHR$(126
);CHR$(126);:INPUT L
110 IF L<1 OR L>2 THEN 100
120 TRAP 40000:Q=INT(RND(1)*20):QUOTE=
1000+Q:P=P-1:L=L-1
130 RESTORE QUOTE:READ MES1$
140 IF  NOT P THEN 180
150 Q1=INT(RND(1)*20):IF Q1=Q THEN 150
:REM DON'T LET BOTH PLAYERS HAVE SAME
QUOTE
160 QUOTE=1000+Q1:RESTORE QUOTE
170 READ MES2$
180 ? CHR$(125):GOSUB 400+L:REM SHUFFL
E THE LETTERS
190 POSITION 25,0:? "PLAYER #1":TEMP$=
MES1$:REM PLAYER #1 IS INVERSE
200 C=1:R=1:C1=C:R1=R:D=1
210 GOSUB 900:R3=R:REM FIRST MESSAGE
230 SCRM1$=TEMP$:REM STORE THEN MESSAG
E AND THE ENCRYPTION
240 TEMP$=MES1$:GOSUB 870:REM CHANGE L
ETTERS TO DASHES
250 C=1:R=7:D=0:GOSUB 900:ANS1$=TEMP$
```

```
260 IF  NOT P THEN 290
265 REM PLAYER #2 IS INVERSE
270 GOSUB 400+L:TEMP$=MES2$:C=1:R=13:C
2=C:R2=R:POSITION 25,12:? "PLAYER #2":
D=1:GOSUB 900:R4=R
280 SCRM2$=TEMP$:TEMP$=MES2$:GOSUB 870
:C=1:R=19:D=0:GOSUB 900:ANS2$=TEMP$
290 C=C1:R=R1+1:RE=R3+1:RB=R:TEMP$=SCR
M1$:TEMP2$=ANS1$:GOSUB 830:GOSUB 740
300 ANS1$=TEMP2$:IF ANS1$=MES1$ THEN 9
80
310 TEMP$=ANS1$:D=1:C=1:R=7:GOSUB 950
320 IF  NOT P THEN 290
330 C=C2:R=R2+1:RE=R4+1:RB=R:TEMP$=SCR
M2$:TEMP2$=ANS2$:GOSUB 830:GOSUB 740
340 ANS2$=TEMP2$:IF ANS2$=MES2$ THEN 9
80
350 TEMP$=ANS2$:D=1:C=1:R=19:GOSUB 950
360 GOTO 290
400 FOR X=1 TO 26:ALPH$(X,X)=CHR$(X+64
):NEXT X:GOTO 410
401 FOR X=1 TO 26:ALPH$(X,X)=CHR$(X):N
EXT X
410 FOR T=1 TO 3:FOR X=26 TO 1 STEP -1
:Q=INT(RND(1)*X)+1
420 TEMP$(1,1)=ALPH$(Q,Q):ALPH$(Q,Q)=A
LPH$(X,X):ALPH$(X,X)=TEMP$(1,1):REM MO
VE LETTER TO LAST POSITION AVAILABLE
430 NEXT X:NEXT T
440 FOR X=1 TO 26:IF ALPH$(X,X)=CHR$(X
+64) THEN GOTO 400+L
450 NEXT X:RETURN
600 POSITION C,R:? "^";:REM PRINT THE
UP-ARROW
610 OPEN #2,4,0,"K:":REM OPEN KEYBOARD
 FOR A READ
620 GET #2,K:CLOSE #2:POSITION C,R:? "
 ";:IF K=27 THEN 810:REM CHANGE ALL TH
E CHARACTERS BACK
630 IF K>127 THEN K=K-128:POKE 694,0:R
EM TOGGLE THE INVERSE FLAG
640 IF K=32 THEN GOSUB 760
650 IF K=43 THEN 700
660 IF K=42 THEN 730
670 GOTO 600
700 C=C-1:IF C=0 THEN C=38:R=R-2:IF R<
```

```
RB THEN R=RE
710 LOCATE C,R-1,L1:POSITION C,R-1:? C
HR$(L1):IF  NOT L THEN IF L1<65 OR L1>
90 THEN 700
715 IF L THEN IF L1>26 THEN 700
720 GOTO 600
730 C=C+1:IF C=39 THEN C=1:R=R+2:IF R>
RE THEN R=RB
740 LOCATE C,R-1,L1:POSITION C,R-1:? C
HR$(L1):IF  NOT L THEN IF L1<65 OR L1>
90 THEN 730
745 IF L THEN IF L1>26 THEN 730
750 GOTO 600
760 POSITION 1,RB-2:? "?";:GOSUB 840:O
PEN #2,4,0,"K:":REM ? IS INVERSE
770 GET #2,K:CLOSE #2:IF K=155 THEN RE
TURN
775 IF K>127 THEN K=K-128:POKE 694,0:R
EM RESET INVERSE FLAG
777 IF K>95 THEN K=K-32:POKE 702,64:RE
M RESET TO UPPER CASE
780 IF K<65 OR K>90 THEN 760
790 POSITION 3,RB-2:? CHR$(K);:FOR X=1
 TO LEN(TEMP$):IF ASC(TEMP$(X,X))=L1 T
HEN TEMP2$(X,X)=CHR$(K)
800 NEXT X:POSITION 1,RB-2:? "   ";:PO
P :RETURN
810 TEMP$=TEMP2$:GOSUB 870:TEMP2$=TEMP
$:RETURN
830 FOR X=1 TO 10:SOUND 0,20,10,10:NEX
T X:SOUND 0,0,0,0:RETURN :REM CURSOR R
EADY SOUND
840 FOR X=1 TO 10:SOUND 0,80,10,10:NEX
T X:SOUND 0,0,0,0:RETURN :REM INPUT LE
TTER SOUND
870 FOR X=1 TO LEN(TEMP$)
880 IF ASC(TEMP$(X,X))<27 OR (ASC(TEMP
$(X,X))>64 AND ASC(TEMP$(X,X))<91) THE
N TEMP$(X,X)="-"
890 NEXT X:RETURN
900 FOR X=1 TO LEN(TEMP$):IF ASC(TEMP$
(X,X))=92 THEN TEMP$(X,X)=CHR$(44):REM
 PRINT COMMA FOR SLASH
910 IF ASC(TEMP$(X,X))<65 THEN 930:REM
 KEEP NON-LETTERS
920 IF D THEN TEMP$(X,X)=ALPH$(ASC(TEM
```

```
P$(X,X))-64):REM CRYPT FOR LETTERS
930 NEXT X
950 FOR X=1 TO LEN(TEMP$):POSITION C,R
:? TEMP$(X,X);
960 C=C+1:IF C>=30 AND TEMP$(X,X)=" "
THEN C=1:R=R+2:REM NEXT ROW
970 NEXT X:RETURN
980 FOR X=1 TO 10:POKE 710,150-PEEK(71
0):FOR T=1 TO 100:NEXT T:NEXT X
990 GOTO 990:REM LOOP HERE UNTIL RESET
 IS PRESSED
1000 DATA "WHEN IDEAS FAIL\ WORDS COME
 IN VERY HANDY." GOETHE
1001 DATA "I ONLY LIKE TWO KINDS OF ME
N: DOMESTIC AND FOREIGN." MAE WEST
1002 DATA "THE GREATEST PRAYER IS PATI
ENCE." THE BUDDHA
1003 DATA "WHEN THE CAT AND MOUSE AGRE
E\ THE GROCER IS RUINED." PERSIAN PROV
ERB
1004 DATA "IF YOU DRINK\ DON'T DRIVE.
DON'T EVEN PUTT." DEAN MARTIN
1005 DATA "IT IS BETTER TO WEAR OUT TH
AN TO RUST OUT." BISHOP CUMBERLAND
1006 DATA "A SMILE IS THE SHORTEST DIS
TANCE BETWEEN TWO PEOPLE." VICTOR BORG
E
1007 DATA "THE REWARD OF A THING WELL
DONE\ IS TO HAVE DONE IT." RALPH WALDO

1008 DATA "BLESSED ARE THE PURE OF HEA
RT; FOR THEY SHALL SEE GOD." MATTHEW 5
:8
1009 DATA "A SMALL LEAK WILL SINK A GR
EAT SHIP." THOMAS FULLER
1010 DATA "MUSIC IS TO THE MIND AS AIR
 IS TO THE BODY." PLATO
1011 DATA "HE THAT PLANTS TREES LOVES
OTHERS BESIDES HIMSELF." ENGLISH PROVE
RB
1012 DATA "JUSTICE IS TRUTH IN ACTION.
" BENJAMIN DISRAELI
1013 DATA "NEVER CLAIM AS A RIGHT WHAT
 YOU CAN ASK AS A FAVOR." JOHN CHURTON
 COLLINS
1014 DATA "MIGHT DOES NOT MAKE RIGHT;
```

```
IT ONLY MAKES HISTORY." JIM FIEBIG
1015 DATA "IF YOU WANT A THING WELL DO
NE\ DO IT YOURSELF." CHARLES HADDON SP
URGEON
1016 DATA "NO GOOD DEED GOES UNPUNISHE
D." CLARE BOOTHE LUCE
1017 DATA "MEN WILL EITHER BE GOVERNED
 BY GOD OR RULED BY TYRANTS." WILLIAM
PENN
1018 DATA "IF ALL THE WORLD WERE JUST\
 THERE WOULD BE NO NEED OF VALOUR." PL
UTARCH
1019 DATA "THE ONLY REAL PEOPLE ARE TH
E PEOPLE WHO NEVER EXISTED." OSCAR WIL
DE
```

and clears the screen. Although the computer is normally set up in the text or graphics 0 mode, we must use the command in this program so we can use the PLOT and DRAWTO commands. If the screen is not cleared after the POKE command, the cursor will stay on the screen.

Line 70 uses the PLOT and DRAWTO commands to draw two rows of asterisks on the screen.

Line 80 asks you to enter the number of players. The two CHR$(126)s backspace over the two spaces that follow the parenthesis. This erases an erroneous answer. The TRAP prevents the program from crashing if a letter or character is entered instead of a number. The computer remains on this line until a 1 or 2 is entered.

Lines 90-110 ask for the level of play. The TRAP is used again to prevent the program from crashing. The two spaces and two CHR$(126)s erase an erroneous answer. Variable L is tested for the value 1 or 2. If it contains any other number, line 100 is repeated.

Line 120 disables the TRAP. A number from 1 to 20 is chosen. This number will be the quotation the computer will encrypt. By adding 1000 to the value of Q, the computer knows the line number of the selected quotation. One is subtracted from the values of P and L. If there is only one player, or level one was chosen, the value of the variable is now 0. This value is used later in the program.

Line 130 points to the line that contains the quotation. The quotation is read into MES1$.

Line 140 uses the NOT operator to determine whether there are one or two players. Two lines before, 1 was subtracted from the value of P. In this line, if P is 0 the computer is sent to line 180. If P is 1, the computer continues with the next line.

Line 150 selects a number from 1 to 20 for the second player. If the number chosen is the same number as that chosen for the first player, the line is repeated.

Line 160 adds 1000 to the value of Q1 and uses the RESTORE command to point to the quotation.

Line 170 reads the new quotation into MES2$. This is the quotation that will be encrypted for the second player.

Line 180 clears the screen. The computer is sent to the correct line of the subroutine that shuffles the letters. The value of L is added to 400 to arrive at the correct line.

Line 190 prints the number of the player whose turn it is. The letters, character and number are in inverse video. The quotation is placed into TEMP$.

Line 200—the variables C, R, C1, R1, and D are all set to 1. These values are used to print the encrypted code on the screen.

Line 210 sends the computer to the subroutine at line 900. This subroutine substitutes a new letter or character for the letters in the quotation. The value of R is stored in R3. This is the last row used by the encryption.

Line 230 stores the encrypted quotation in SCRM1$.

Line 240 places the unencrypted quotation back in TEMP$. The subroutine at line 870 changes all the letters in the quotation into dashes.

Line 250 resets the variable C to 1, the variable R to 7. This is the row and column at which the dashes will begin. The variable D is set to 0. The same subroutine at line 900 is used to print the dashes on the screen. The string of dashes is then transferred to ANS1$.

Line 260 checks the value of P. If it is NOT, or is 0, the computer is directed to line 290. There is only one player.

Line 270 sends the computer to the correct line to shuffle the letters or characters for the second player. The second message is stored in the temporary string TEMP$. Variable C is set to 1 and R is set to 13. This encryption is printed on the 13th row, beginning with column 1. The column and row values for the second player are stored in variables C2 and R2. These values are used later in the program to determine where the up arrow should be placed. The player number is printed on the screen in inverse video.

Variable D is set to 1, indicating that letters or characters rather than dashes should be printed. The computer is sent to the subroutine at line 900. The second encrypted quotation is printed on the screen. The last row the message uses is stored in variable R4.

Line 280 places the encrypted quotation in SCRM2$. The quotation stored in MES2$ is moved into TEMP$ so the computer can use the subroutine at line 870 to convert the letters into dashes. Column variable C is set to 1 and row variable R is set to 19. The dashes begin on the 19th row. Variable D is set to 0, indicating that dashes will be printed on the screen. The computer uses the subroutine at line 900 to print the dashes, then stores this string in ANS2$.

Line 290 moves the column value of the first quotation from C1 to C. The row number is increased by 1 and stored in variable R. Variable RE contains the last row that letters are printed on, plus 1. The first row the up arrow will be printed on is stored in variable RB. When the computer reaches the last row the up arrow can be printed on, it needs to know where the first row is, to complete the wrap-around. The encrypted quotation is placed into TEMP$ and the current answer is stored in TEMP2$. The computer uses the subroutine at line 830 to make a sound, then uses the subroutine at line 740 to move the up arrow and substitute a letter for another letter or character.

Line 300 places the contents of TEMP2$ into ANS1$. This is the string with the dashes. The letters that have been changed are also placed in this string. If the contents of ANS1$ are the same as MES1$, the code has been broken and the computer is sent to line 980.

Line 320 checks the value of P. If there is only one player, the computer proceeds to line 290. There is no need to use the remaining lines. These lines are used by the second player.

Line 330 moves the values stored in variables C2 and R2 to indicate the column and row of the second player's quotation. The value of the row is one more than the row the letters are on, so the arrow is under the letters. Variable RE is the last row on which the up arrow can be. The first row of the quotation is stored in variable RB. The encrypted quotation is moved to TEMP$, and the dashes and letters that have been changed are moved to TEMP2$. The subroutine at line 830 sounds the tone. The computer uses the subroutine at line 740 to move the up arrow or change a letter.

Line 340 places the new string of dashes and changed letters into ANS2$. This string is compared to MES2$. If the quotation has been deciphered, the computer moves to line 980 to end the game.

Line 350 places ANS2$ into the temporary string TEMP$, sets the variables to print dashes beginning with the correct row and column, and uses the subroutine at line 950 to print the string on the screen.

Line 360 sends the computer to line 290 to continue the program until one player solves the code.

Lines 400-450 contain the subroutine that shuffles the letters or characters.

Line 400 is used if level 1—letter encryption—is chosen. The variable X counts from 1 to 26: there are 26 letters in the alphabet. The ASCII code for the letters is not 1 to 26. To get the correct ASCII code for uppercase letters, 64 must be added to the value of X. The character of this value, which is an uppercase letter, is stored in ALPH$.

Line 401 is used if level 2—character encryption—was chosen. The values 1-26 contain the ATARI characters. When either routine is finished, ALPH$ will contain the entire alphabet or character set in the correct order.

Line 410 begins the shuffling routine. The letters or characters are shuffled three times. The second FOR-NEXT loop begins with the last element of ALPH$ and steps backwards. One element of the string is chosen. Each time this loop repeats, there is one less element to choose from.

Line 420 moves the letter or character from ALPH$ and places it in the first element of TEMP$. It then moves the last possible element of ALPH$ and places it into the chosen element. The letter of character placed into TEMP$ is moved into the last possible element of ALPH$.

Line 430 continues the loop. Variable X is decremented by one. The letter of character moved into the last possible element of ALPH$ cannot be moved because its position is greater than the value of X. This routine continues until the letters or characters have been shuffled three times. The computer returns to the main part of the program. This routine can take several seconds. The screen remains blank while the letters are shuffled and the quotation is encrypted.

Lines 440-450 check each element of ALPH$ to make sure every letter or character has been moved. If one has not, the routine

is repeated. Otherwise, the computer returns to the main program.

Lines 600-810 move the up arrow and change the letters or characters. The up arrow is printed at the position set by variables C and R. These variables contain the column and row values. These values change depending on which player is trying to break the code.

Line 610 opens the keyboard so the key that is pressed can be read without having to press the Return key.

Line 620 uses the GET command to place the value of the key pressed into variable K. The computer waits at this line until a key is pressed. Once a key is pressed, the keyboard is closed and the up arrow is erased from the screen. If the Escape key is pressed, the computer is directed to line 810. All the characters are changed back to the original encryption, and the letters on the dashes are removed. The dashes are placed back on the screen.

Line 630 checks to see if the variable K is greater than 127. If it is, the ATARI key has been accidentally pressed. To get the correct value of the pressed key, 128 is subtracted from the value of K and the location 694 is POKEd with zero. This resets the keyboard to normal video.

Line 640 checks to see if the space bar was pressed. If it has, the computer goes to line 760 to change a letter.

Line 650 compares K to the value of the back arrow. The computer moves to line 700 if the up arrow should move backwards.

Line 660 directs the computer to line 730 when the right arrow key is pressed. To move the up arrow, simply press the arrow key. The program look for the value of the key without pressing the shift or Control key.

Lines 700-720 move the up arrow to the left. The variable C is the column in which the up arrow is located. It is decreased by 1. If it becomes zero, it is reset to 38—the last column in which letters can be printed. If the variable is reset to 38, the row value is decreased by 2. The empty rows for the up arrow are two rows apart. If the value of R is less than the value of RB, the up arrow would be printed above the letters for this quotation, so variable R is reset to RE—the last row the up arrow can be in.

Line 710 uses the Locate command to get the value of the letter or character to which the up arrow is pointing. The letter or character is printed in that location because, occasionally, the computer erases a character when the LOCATE command is used. The value of L is checked for zero. If the player chose level 1, variable L1 is checked to see if it is a letter. If it is not, the computer is

sent back to line 700 to move it to the left again. This way, the up arrow will not point to spaces or punctuation marks.

Line 715 checks to see if the up arrow is pointing to anything other than a character. If it is, line 700 is repeated.

Line 720 sends the computer back to line 600 so the player can move the up arrow or change a letter.

Lines 730-750 move the up arrow to the right. Variable C is increased by 1. It is then checked to see if it is equal to 39. This is the right-most edge of the screen. If it is, the variable is reset to 1. This is the first column. Variable R is now increased by 2 so the up arrow can move down one row. The value of R is compared to RE to make sure the up arrow will not be printed below the dashes. RE is the last row the up arrow can be in. If R is greater than RE, the variable R is reset to RB—the first row the up arrow can be in.

Line 740 uses the LOCATE command to get the value of the letter or character to which the up arrow is pointing. The letter or character is printed on the screen. The computer then checks to see if this is level 1. If it is, L equals 0, and the computer checks the value of L1—the letter to which it is pointing. If the up arrow is not pointing to a letter, the value of L1 will be less than 65 or greater than 90. The computer repeats line 730 and moves the up arrow one more position to the right.

Line 745 compares the value of L1 to 26. The computer uses this line when L is 1, i.e., level 2 was chosen. If the value of L1 is greater than 26, the up arrow is not pointing to a character and the computer repeats line 730, moving the up arrow one more column to the right.

Line 750 sends the computer to line 600 to wait for the player to move the up arrow again or change a letter.

Lines 760-800 get the letter the player wants to use. A question mark is printed above the row the up arrow was pointing to and the computer makes a sound to prompt the player to press a key. The keyboard is opened.

Line 770 gets the value of the pressed key and places it in variable K. The keyboard is closed. Variable K is checked to see if it contains 155. This is the code for the Return key. If it is, the computer returns the routine that called it. This is an escape in case the player changes his mind about entering a letter.

Line 775 checks the value of K to see if the ATARI key has been pressed. If it has, the keyboard is reset for normal video and variable K is adjusted.

Line 777 checks to see if the CAPS/LOWER key had been pressed. If it had, the keyboard is reset for uppercase and the value of K is adjusted.

Line 780 now checks to see if a letter has been pressed. Only letters can be substituted for the letters or characters in the encrypted code. If a letter has not been pressed, the computer repeats line 760 and waits for a letter key.

Line 790 prints the letter that has been pressed. Every character in TEMP$ is checked for the code of the letter or character being changed. TEMP$ contains the encrypted code for the quotation. When the matching character is found, the letter that is being substituted for that letter or character is placed in TEMP2$. This is the string that contains the dashes. This way, only the string with the answer is changed. The encrypted message stays intact.

Line 800 continues the loop until all the characters or letters have been changed. The letter that has been substituted is erased from the screen. The POP command is used to remove one return address off the stack and the RETURN command is used to send the computer back to the main program. If POP was not used, the computer would return to the calling routine, and the second player would not get a turn.

Line 810 removes the letters from TEMP2$ and replaces them with dashes. This line is used when the Escape key is pressed. The answer string stored in TEMP2$ is moved to TEMP$. The subroutine at line 870 is used to change any letters to dashes. This string is moved back to TEMP2$. The computer returns to the main program because a GOTO rather than a GOSUB is used to send the computer to this line.

Line 830 is the sound subroutine used to tell the player when the up arrow is on the screen.

Line 840 is the sound used to prompt the player to press a letter key.

Lines 870-890 change the letters in a string to dashes. The values of the characters in TEMP$ are checked to make sure the character is a graphics character or a letter. Only the spaces and punctuation marks remain.

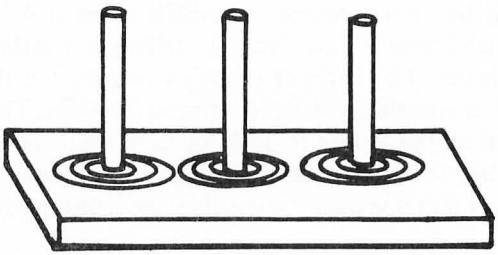
Lines 900-970 encrypt the quotation and print the encryption on the screen. First, every slash in the quotation is changed to a comma. The slashes are used in the DATA lines because a comma is used to indicate the end of the information in the DATA line. The slashes are changed to commas in the string so the quotation is printed correctly on the screen.

If the character of the string is a space or punctuation mark the routine skips the character completely, since it is only interested in changing the letters. If variable D is set to 1, the letter is changed to another letter or a graphics character. The ASCII value of the letter is decreased by 64. This number is the position of the letter in the alphabet. For example, the ASCII value of A is 65. So by subtracting 64 from it we arrive at 1—telling us it is the first letter of the alphabet. The letter or graphics character in this position in ALPH$ is substituted for the letter in TEMP$. The loop continues until all the letters in TEMP$ have been changed.

Lines 980-990 end the program. The screen changes colors 10 times. The computer then loops at line 990 until the Reset key is pressed.

Lines 1000-1019 contain the quotations used in this program. The quotation marks must be inserted as well as the slashes. The slashes are changed to commas within the program, but if commas were used in these DATA lines, the computer would not read the entire line. It stops whenever it reaches a comma. These lines can be changed to any quotation or sentences as long as slashes are used instead of commas, and the line does not exceed 100 letters and characters.

Chapter 4



Logic Games

The programs in this chapter require a thorough analysis of each problem to win the game. Much more is involved than luck or skill. Each game follows a logical sequence of steps. In some of the games the winner is determined by the first move; others require careful thinking and evaluation of the moves made. Like chess, you should try to be one step ahead of the computer. Good luck!

PEBBLES

Objective of the game: To force the computer to take the last pebble.

Directions: There are 23 pebbles scattered on the beach. You can take one, two, or three pebbles in each turn. The computer can take one to three pebbles in its turn. The player who takes the last pebble is the loser.

The screen clears and the title of the program is displayed. The pebbles are scattered on the screen. The computer asks if you want to go first. If you do, press the Y key, otherwise, press N. You are asked to enter a number from one to three: this is the number of pebbles you will take. After you have entered a number, that number of pebbles disappears from the screen. The computer will also take away some pebbles. This number is placed on the screen so you will know how many pebbles the computer took. These pebbles are also erased from the screen.

The program continues until one player is forced to take the last pebble. When all the pebbles are gone, the computer displays the winner. The winning message will remain on the screen until the Start key is pressed. If you do not want to play again, press the System Reset key. Figure 4-1 is the flowchart, Fig. 4-2 the character set for this program, and Listing 4-1 is the code.

Line 60 sets aside memory for the S array. This array keeps track of the locations of the pebbles.

Line 70 PEEKs at location 106 to see how much memory is in the computer. Eight is subtracted from this amount, and the value is stored in location 204. This is the new address of the character set that will be transferred to RAM. The address of the character set in ROM is placed into location 206. This information is used in the assembly language subroutine that moves the character set from ROM into RAM.

Line 80 places the code for the assembly language subroutine in memory addresses 1536-1555. The computer uses the USR command to execute the assembly language subroutine in RAM. Line 90 contains the decimal codes for the routine that moves the character set in ROM into RAM.

Line 100 changes the screen to graphics 18. This is graphics 2 with no text window.

Line 110 calculates the beginning address of the character set in RAM. The value of A is multiplied by 256. The FOR-NEXT loop reads the code for the new characters. The first character that will be changed is the fourth character of the character set. Its first byte is 24 bytes after the first byte of the character set. Two characters in the character set will be changed.

Lines 120-130 contain the new codes for the two new characters.

Line 140 POKEs location 756 with the value of A. The new character set in RAM will be used by the computer.

Line 150 places the name of the program on the screen.

Line 160 begins the FOR-NEXT loop that places the pebbles on the screen. The loop counts from 1 to 23 because there are 23 pebbles to be placed.

Line 170 selects two random numbers. One is the column, the other the row. The column can be any number from 1 to 17. The row can be any number from 3 through 9. This keeps the pebbles in the middle of the screen.

Line 180 uses the LOCATE command to check the location the computer picked. If this location already contains a pebble, the

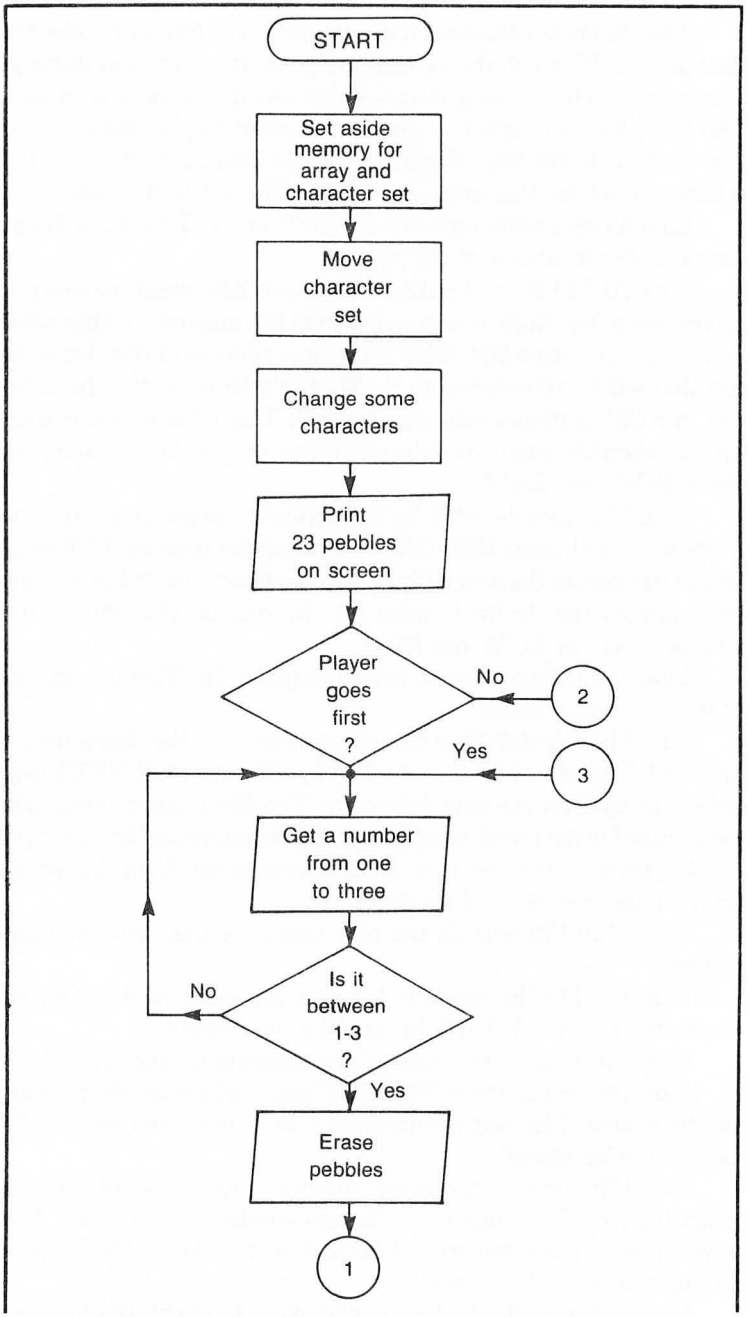
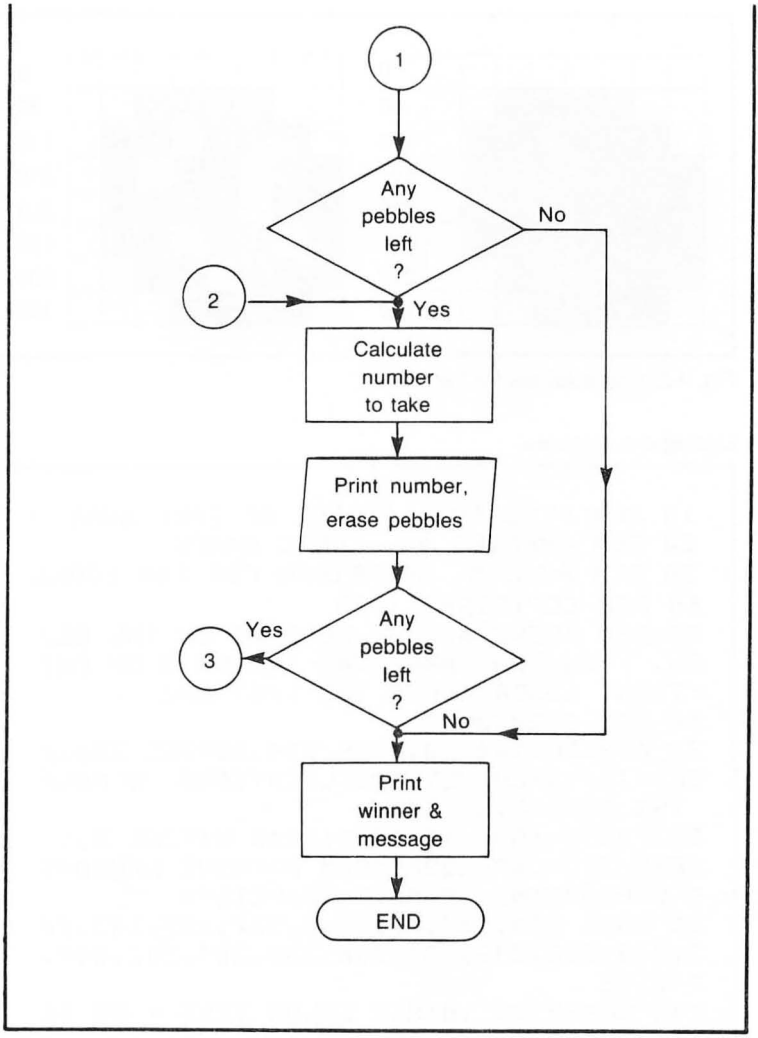


Fig. 4-1. Flowchart for Pebbles.



computer is sent back to line 170 to pick another location. The value of P will be 32 if the location is empty.

Line 190 picks a number, either 1 or 2. This determines which character the computer will print for the pebble. The next random number is the color value. The offset that will be added to the character value depends on which number was chosen. Either pebble can be printed in any of the four colors. The computer starts by setting variable CL to 0. It then checks the value of C1. Each possible value changes the value of CL.

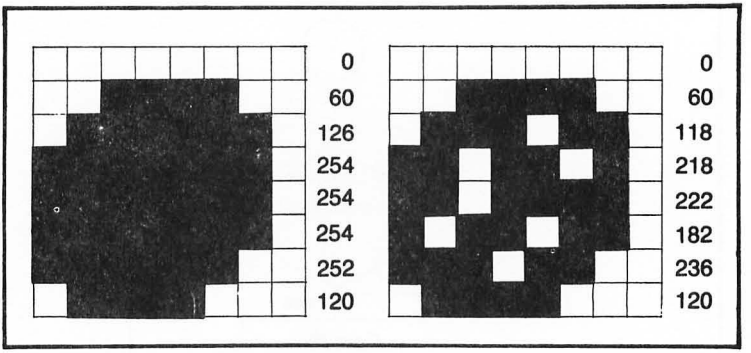


Fig. 4-2. Character set for Pebbles.

Listing 4-1. Pebbles.

```
10 REM PEBBLES - A GAME OF TAKE-AWAY
20 REM CHAPTER 4 - LOGIC GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 REM THERE ARE 23 PEBBLES ON THE BEA
CH.  YOU CAN TAKE AWAY 1,2,OR 3 ON ONE
 TURN. LOSER TAKES THE LAST ONE
60 DIM S(23,2)
70 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756):REM SET THE LOCATIONS TO MOVE
 THE CHARACTER SET
80 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:Q=USR(1536):REM MACHINE LANGUAG
E SUBROUTINE TO MOVE CHARCTERS
90 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
100 GRAPHICS 18:REM COLOR TEXT - NO TE
XT WINDOW
110 A1=A*256:FOR X=24 TO 39:READ V:POK
E X+A1,V:NEXT X:REM ADD NEW CHARACTERS
120 DATA 0,60,126,254,254,254,252,120
130 DATA 0,60,118,218,222,182,236,120
140 POKE 756,A:REM USE THE NEW CHARACT
ER SET
150 POSITION 6,1:? #6;"PebBLes!":REM b
Bs! ARE INVERSE
160 FOR X=1 TO 23:REM PRINT 23 PEBBLES
170 C=INT(RND(1)*17)+1:R=INT(RND(1)*6)
```

```
+3:REM PICK A ROW AND COLUMN
180 LOCATE C,R,P:POSITION C,R:? #6;CHR
$(P);:IF P<>32 THEN 170:REM CHECK THAT
 LOCATION
190 W=INT(RND(1)*2):C1=INT(RND(1)*4):C
L=0:IF C1>0 THEN CL=32:IF C1>1 THEN CL
=128:IF C1>2 THEN CL=160
200 POSITION C,R:? #6;CHR$(3+CL+W);:RE
M PUT IT ON THE SCREEN
210 S(X,1)=C:S(X,2)=R:NEXT X:X=X-1:REM
 REMEMBER WHERE IT IS & KEEP GOING UNT
IL ALL ARE ON SCREEN
220 POSITION 1,11:? #6;"go first (y/n)
?";:REM SEE IF PLAYER WANTS TO GO FIRS
T
230 OPEN #2,4,0,"K:":GET #2,B:CLOSE #2
:IF B<>89 AND B<>78 THEN 230:REM GET A
 KEY & CHECK IT
240 IF B=78 THEN 300
250 GOSUB 500:POSITION 1,11:? #6;"how
many (1,2,3)?":REM PRINT MESSAGE IN IN
VERSE
260 OPEN #2,4,0,"K:":GET #2,P:CLOSE #2
:IF P<49 OR P>51 OR P-48>X THEN 250:RE
M CHECK THE NUMBER
270 P=P-48:FOR M=X TO X-P+1 STEP -1:PO
SITION S(M,1),S(M,2):? #6;" ";:NEXT M:
REM ERASE THE PEBBLES
280 FOR TIME=1 TO 200:NEXT TIME:X=X-P:
IF X=0 THEN W=1:GOTO 400:REM SUBTRACT
THE NUMBER TAKEN
290 REM COMPUTER'S TURN
300 POSITION 1,11:? #6;"
   ":REM ERASE THE MESSAGE
310 P=X-(INT((X-1)/4)*4+1):IF P=0 THEN
 P=INT(RND(1)*3)+1:IF P>X AND X=1 THEN
 P=1:REM FIGURE OUT HOW MANY TO TAKE A
320 POSITION 1,11:? #6;"I TOOK ";P:FOR
 TIME=1 TO 100:NEXT TIME:REM TELL HOW
MANY ARE TAKEN AWAY
330 FOR M=X TO X-P+1 STEP -1:POSITION
S(M,1),S(M,2):? #6;" ";:NEXT M:REM REM
OVE THE PEBBLES
340 GOSUB 510:X=X-P:IF X=0 THEN W=0:GO
TO 400
350 GOTO 250
```

```
400 GRAPHICS 18:POSITION 5,5:? #6;"THE
 WINNER IS "
410 POSITION 5,7:IF W THEN ? #6;"THE C
OMPUTER":GOTO 430:REM UPPER CASE INVER
S
420 ? #6;"the human":REM LOWER CASE IN
VERSE


 430 IF PEEK(53279)<>6 THEN 430:REM STA
 Y HERE UNTIL START KEY IS PRESSED
 440 RUN
 500 SOUND 0,100,14,8:FOR TIME=1 TO 50:
 NEXT TIME:SOUND 0,0,0,0:RETURN
 510 SOUND 0,200,10,8:FOR TIME=1 TO 50:
 NEXT TIME:SOUND 0,0,0,0:FOR TIME=1 TO
 500:NEXT TIME:RETURN
```

Line 200 prints the pebble on the screen. The value of variables CL and W are added to 3 to get the right pebble in the right color.

Line 210 stores the values of R and C in array S. Variable X is the number of the pebble. The column is stored in the first part of that element and the row in the second. The loop continues until all the pebbles are placed. The X variable is decreased by 1 so it contains the actual number of pebbles.

Line 220 asks if the player wants to go first.

Line 230 opens the keyboard for a read, then waits until a key is pressed. Once a key is pressed, the keyboard is closed. If the value of the pressed key is not 89 or 78, the computer loops back to the beginning of this line. The computer loops at this line until a Y or N is entered.

Line 240 sends the computer to line 300 if the pressed key was an N. The computer will go first.

Line 250 uses the subroutine in line 500 to make a sound. This is the prompt. The computer then prints a message on the screen. You are asked to enter the number of pebbles you would like to take.

Line 260 opens the keyboard for a read. The computer waits until a key is pressed. If the value of the key is less than 49 or greater than 51, or greater than the number of pebbles left on the screen, the computer goes back to line 250 to make a sound and wait for another entry. The computer loops between these two lines until the 1, 2, or 3 key is pressed.

Line 270 subtracts 48 from the value of P. Variable P holds the value of the pressed key. To get the number the key represents, the computer must subtract 48 from the value of P. The FOR-NEXT loop begins with the last pebble printed in the screen and erases the number of pebbles entered. The computer uses the values stored in the S array to find the pebbles. The space is printed at these locations. The loop continues until the correct number of pebbles are erased.

Line 280 is a timing loop that slows the program down. The number of pebbles taken away is subtracted from variable X. The next time pebbles are erased, the pebble pointed to by X is erased. Variable X also keeps track of how many pebbles are on the screen. If the value of X is 0, variable W is set to line number 400 and the computer is sent to line 400 to declare the winner.

Line 300 begins the computer's turn. The message on the screen is erased.

Line 310 decides how many pebbles to take away. This is not a random number selection. To win the game, the computer makes sure the number of pebbles left is one more than a multiple of four. This way, no matter how many pebbles you take, on your second from the last turn you can only take away enough pebbles to leave two, three, or four for the computer. The computer, of course, will then take away one less than the number of pebbles left and win the game. Only if the value of P is 0 will the computer choose a random number. This means that you, the player, know the winning combination, and are using it in the game. If there is only one pebble left, the computer will take it.

Line 320 prints the number of pebbles the computer will take. The timing loop slows the program down.

Line 330 contains the FOR-NEXT loop that removes the pebbles the computer took. Again, the loop begins with the last pebble left on the screen and uses the values from the S array to find and erase the pebbles.

Line 340 uses the subroutine at line 510 to make a sound. The number of pebbles the computer took is subtracted from variable X. If there are no pebbles left, the computer sets variable W to zero and goes to line 400 to end the program.

Line 350 sends the computer to line 250 to continue the game.

Line 400 begins the routine that ends the game. The screen is cleared by using the graphics command. The winning message is printed on the screen.

Line 410 declares the computer the winner if variable W is set

to 1. The computer goes to line 430 to play another game.

Line 420 is used when the value of W is 0. This line declares the user the winner.

Line 430 loops until the Start key is pressed. Then the program runs again. To end the program press the System Reset key.

Line 500 is the sound prompt for the player's turn.

Line 510 makes the sound when the computer has removed some pebbles.

NIM

Objective of the game: To take the last character.

Directions: This game is a little more involved than Pebbles. You can play against the computer or another person. You may want to start by playing against another person. The computer is programmed to win!

You are asked to enter the number of rows you want in this game. You can play with any number from three to seven. The top row always has one character; each row under it has two more characters than the previous row.

You have the option of going first. You are always Player One. Your opponent, whether it is the computer or another human, is Player Two.

The screen clears and the rows of characters are displayed. Your character is on the left side of the screen, your opponent's is on the right. Use the up and down arrows to move your character up or down until you are in front of the row in which you want to remove characters. There is no limit to the number of characters that can be taken in one turn as long as all the characters taken are in the same row. To remove the characters, press the right arrow key. Your character will move right over the characters in that row. Press the space bar when you have taken all the characters you want in that row.

Your opponent will do the same. The second player moves the up and down arrows to move his character up and down on the screen. The left arrow moves this character to the left. The space bar ends this player's turn.

If you are playing with the computer, the computer will automatically move its character and remove characters from the row.

When all the characters are removed, the player who took the last character is declared the winner. Figure 4-3 is the flowchart

for this program, Fig. 4-4 is the character set, and Listing 4-2 is the code.

Line 50 sets aside the memory used for the arrays and strings. The T array is used to keep track of the characters in the rows. CH$ is one character, CH1$ is the other. The R array keeps track of the number of characters in each row.

Line 60 PEEKs at location 106 to see how much memory is in the computer. This amount is decreased by 8, and the new value is stored in location 204. This is where the character set will begin in RAM. The location of the character set in ROM is stored in location 206. This information is used by the assembly language subroutine that moves the character set from ROM into RAM.

Lines 70-80 read the code for the assembly language subroutine and place it into memory locations 1536-1555.

Line 90 uses the USR command to send the computer to the assembly language subroutine at memory location 1536. Line 100 contains the decimal codes for the routine.

Line 110 finds the first byte of the character set in RAM and stores it in variable CHBAS. The computer will change eight characters in the new character set.

Line 120 continues the loop that changes the characters in the character set.

Lines 130-195 contain the code for the new characters.

Line 200 changes the screen to graphics 18. This is graphics 2 with no next window. The computer POKEs location 756 with the value of A to change the character set used by the computer from the set in ROM to the new set in RAM.

Lines 210-230 print the program options on the screen. The game can be played against the computer or another human.

Line 240 opens the keyboard for a read. The computer waits until a key is pressed. The value of the key that has been pressed is stored in variable K. After a key is pressed, the keyboard is closed.

Line 250 checks to see if the value of K is greater than 127. If it is, the computer subtracts 128 from this value. Location 694 is POKEd with 0 to reset the flag for normal video.

Line 260 checks to see if the value of K is greater than 95. If it is, the value of K is decreased by 32. The key is now uppercase instead of lowercase. Location 702 is POKEd with 64 to reset it to uppercase letters.

Line 270 looks at the value of K. If it is 72 or 67, the C or H

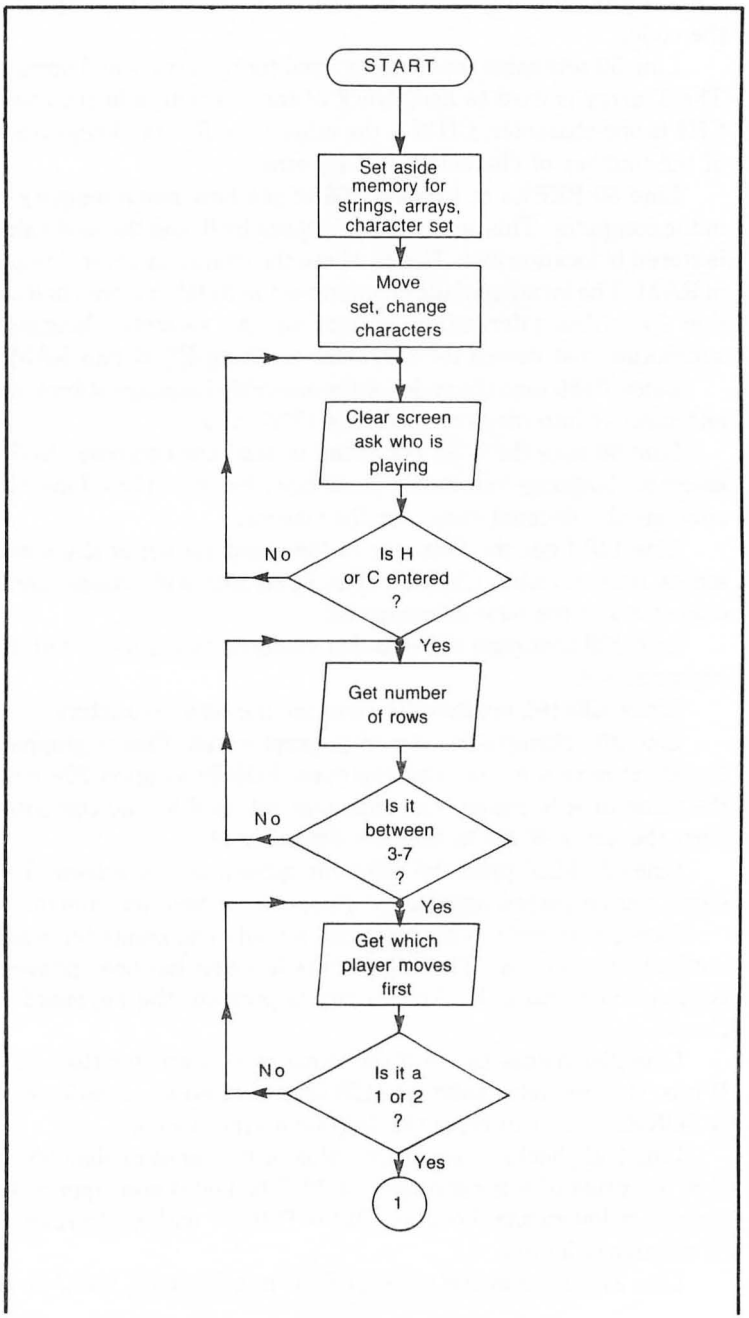
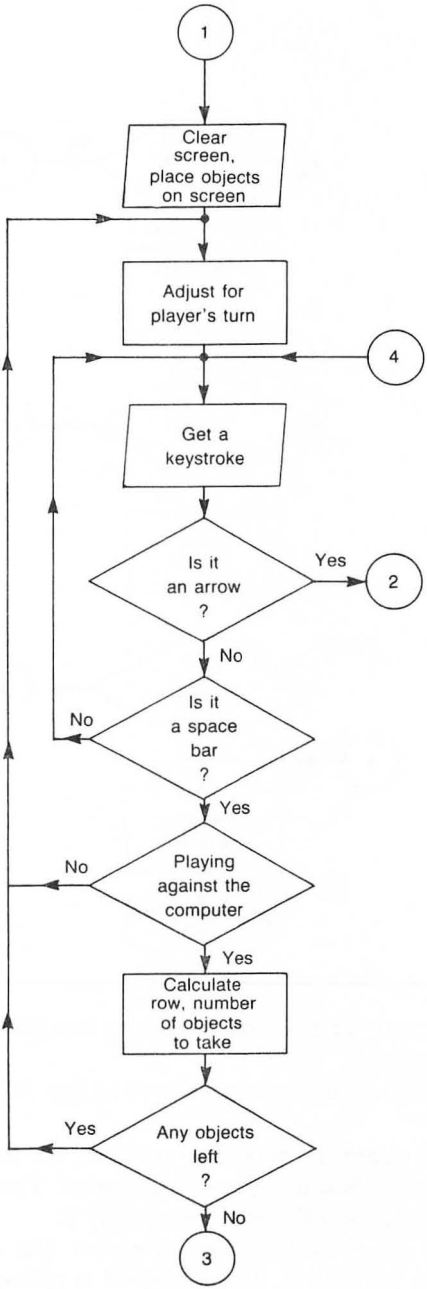
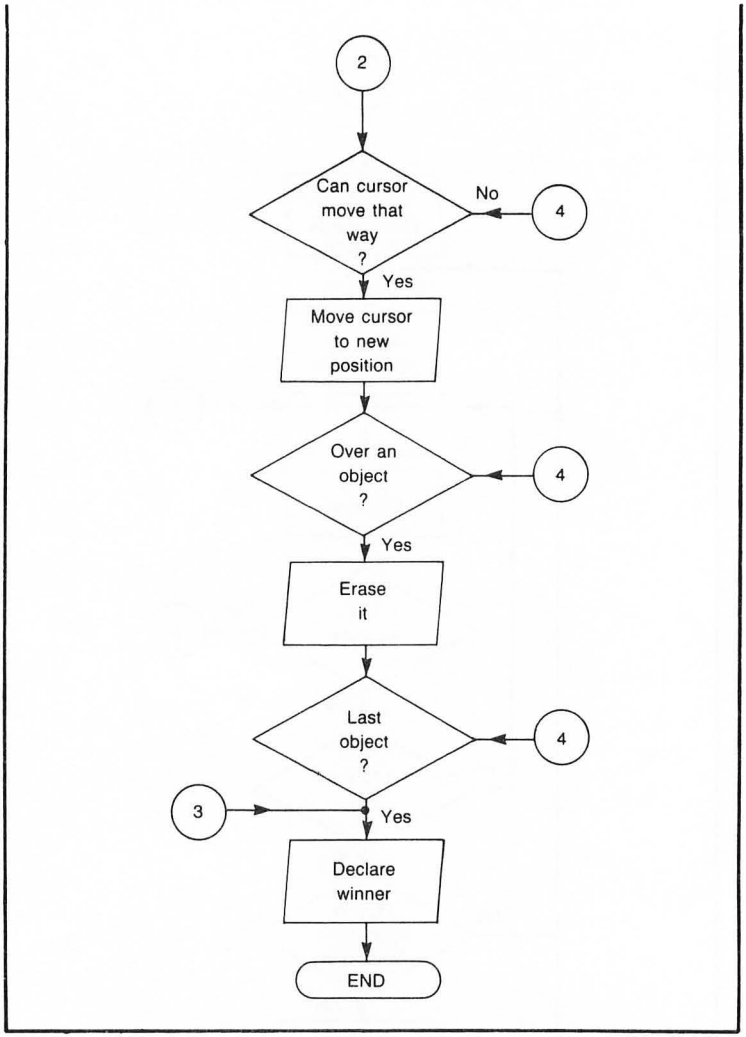


Fig. 4-3. Flowchart for Nim. (Continued through page 190.)


1
Clear screen, place objects on screen
Adjust for player's turn
4
Get a keystroke
Is it an arrow ?
Yes
2
No
Is it a space bar ?
No
Yes
Playing against the computer
No
Yes
Calculate row, number of objects to take
Any objects left ?
Yes
No
3



key was pressed. The computer goes to line 290 to wait to continue the program.

Line 280 sends the computer back to line 240 to get another key value.

Line 290 clears the screen to continue the program.

Line 300 asks how many rows are needed. There are 11 spaces between the S in ROWS and the number 3.

Line 310 opens the keyboard for a read. The number entered is placed into variable R. The keyboard is closed.

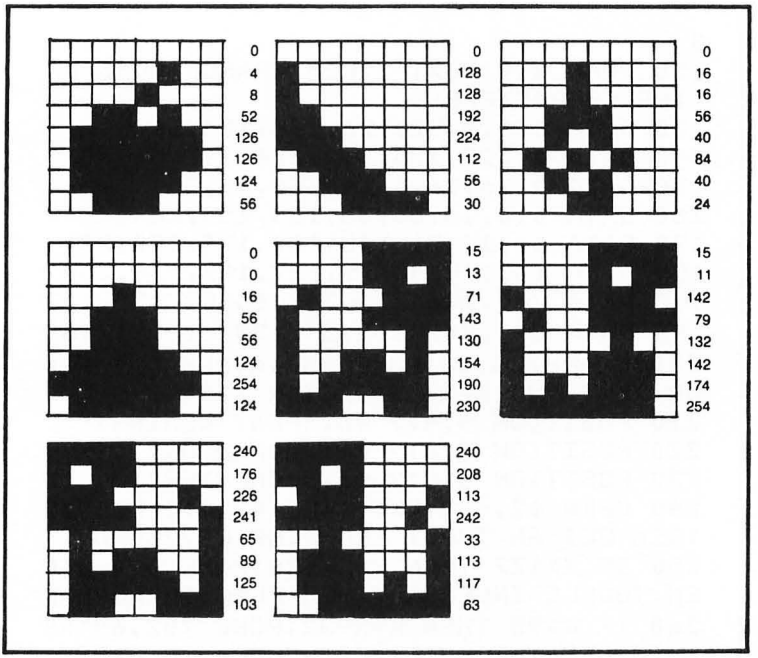


Fig. 4-4. Character set for Nim.

Listing 4-2. Nim.

```
10 REM NIM - A CHINESE GAME OF TAKE-AW
AY
20 REM CHAPTER 4 - LOGIC GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM T(13,7),CH$(1),CH1$(1),R(7)
60 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756)
70 FOR X=1536 TO 1555:READ V:REM GET T
HE MACHINE CODE TO MOVE THE CHARACTER
SET
80 POKE X,V:NEXT X:REM PUT IT IN MEMOR
Y
90 Q=USR(1536):REM RUN IT
100 DATA 104,162,4,160,0,177,205,145,2
03,200,208,249,230,206,230,204,202,208
,242,96
110 CHBAS=A*256:FOR X=CHBAS+24 TO CHBA
S+87:READ V:REM READ THE NEW CHARACTER
```

```
S
120 POKE X,V:NEXT X:REM CHANGE SOME OF
 THE CHARACTERS
130 DATA 0,4,8,52,126,126,124,56
140 DATA 0,128,128,192,224,112,56,30
150 DATA 0,16,16,56,40,84,40,24
160 DATA 0,0,16,56,56,124,254,124
170 DATA 15,13,71,143,130,154,190,230
180 DATA 15,11,142,79,132,142,174,254
190 DATA 240,176,226,241,65,89,125,103
195 DATA 240,208,113,242,33,113,117,63
200 GRAPHICS 18:POKE 756,A:REM USE COL
OR TEXT AND NEW CHARACTER SET
210 POSITION 4,4:? #6;"PLAY AGAINST"
220 POSITION 6,6:? #6;"hUMAN OR"
230 POSITION 6,8:? #6;"cOMPUTER"
240 OPEN #2,4,0,"K:":GET #2,K:CLOSE #2
:REM GET AN INPUT FROM THE KEYBOARD
250 IF K>127 THEN K=K-128:POKE 694,0:R
EM TOGGLE INVERSE VIDEO FLAG TO NORMAL
260 IF K>95 THEN K=K-32:POKE 702,64:RE
M TOGGLE THE SHIFT/LOCK BACK TO CAPS
270 IF K=72 OR K=67 THEN 290:REM PLAYE
R PRESSED 'H' OR 'C'
280 GOTO 240
290 PRINT #6;CHR$(125):REM CLEAR THE S
CREEN FOR NEXT MESSAGE
300 POSITION 3,2:? #6;"HOW MANY ROWS
         3-7?";
310 OPEN #2,4,0,"K:":GET #2,R:CLOSE #2
320 IF R>127 THEN R=R-128:POKE 694,0:R
EM TOGGLE INVERSE FLAG
330 IF R<51 OR R>55 THEN 310:REM INCOR
RECT NUMBER
340 POSITION 12,3:? #6;CHR$(R):R=R-48
350 POSITION 2,6:? #6;"WHO GOES FIRST?
"
360 POSITION 6,8:? #6;"PLAYER 1":REM 1
 IS INVERSE
370 POSITION 6,10:? #6;"PLAYER 2":REM
2 IS INVERSE
380 OPEN #2,4,0,"K:":GET #2,P:CLOSE #2
:REM GET AN INPUT FROM THE KEYBOARD
390 IF P>127 THEN P=P-128:POKE 694,0:R
EM TOGGLE INVERSE VIDEO FLAG TO NORMAL
400 IF P=49 OR P=50 THEN 420:REM PLAYE
```

```
R PRESSED '1' OR '2'
410 GOTO 380
420 P=P-48:REM GET THE VALUE OF P
430 FOR X=1 TO 7:FOR Y=1 TO 13:T(Y,X)=
0:NEXT Y:NEXT X:REM CLEAR THE MATRIX
440 V=1:FOR X=1 TO R:FOR Y=1 TO V:T(Y,
X)=1:NEXT Y:V=V+2:NEXT X:V=V-2:REM FIL
L THE MATRIX FOR THE OBJECTS
450 GRAPHICS 18:POKE 756,A:REM USE COL
OR TEXT AND NEW CHARACTER SET
460 POSITION 8,0:? #6;"nIm":REM TITLE
THE PAGE - Im ARE INVERSE
470 FOR X=1 TO R:FOR Y=1 TO V:Z=INT(RN
D(1)*4):REM PICK A CHARACTER
480 IF T(Y,X) THEN POSITION Y+3,X+2:?
#6;CHR$(Z+3):REM PRINT THE CHARACTER I
F IT'S IN THE MATRIX
490 NEXT Y:NEXT X
500 POSITION 1,1:? #6;"1":POSITION 18,
1:? #6;"2";:REM 2 IS INVERSE
510 C1=1:R1=3:C2=18:R2=3:REM ROW AND C
OLUMN FOR BOTH PLAYER'S TOKENS
520 POSITION C1,R1:? #6;"'":POSITION C
2,R2:? #6;")":REM ) IS INVERSE
530 ON P GOTO 540,550
540 TA=0:C=C1:M=R1:CH$="'":CH1$="(":PO
SITION 1,1:? #6;"1":POSITION 18,1:? #6
;" ":GOSUB 590:P=3-P
545 REM ), *, & 2 ARE INVERSE
550 TA=0:C=C2:M=R2:CH$=")":CH1$="*":PO
SITION 1,1:? #6;" ":POSITION 18,1:? #6
;"2"
560 IF K=72 THEN GOSUB 590:P=3-P:GOTO
540
570 TT=0:TR=0:GOSUB 900:P=3-P:GOTO 540
590 POSITION C,M:? #6;CH$
600 OPEN #2,4,0,"K:":GET #2,A:CLOSE #2
:IF A>127 THEN POKE 694,0:GOTO 600
610 IF A=45 OR A=61 THEN 650
620 IF A=43 OR A=42 THEN 700
630 IF A=32 AND TA THEN POSITION C,M:?
 #6;" ":RETURN
640 GOTO 600:REM NOT AN ARROW KEY
650 IF C<>1 AND C<>18 THEN 600:REM NOT
 IN CORRECT COLUMN
660 POSITION C,M:? #6;" ";:REM ERASE T
```

```
HE CHARACTER
670 IF A=45 THEN M=M-1:IF M=2 THEN M=R
+2:GOTO 690:REM REPOSITION THE CHARACT
ER
680 IF A=61 THEN M=M+1:IF M=R+3 THEN M
=3
690 GOTO 590:REM GET ANOTHER DIRECTION
700 Y=0:FOR X=1 TO 13:Y=Y+T(X,M-2):NEX
T X:REM CHECK IF THERE'S ANYTHING IN T
HIS ROW
710 IF Y=0 THEN 600:REM CAN'T TAKE-AWA
Y FROM AN EMPTY ROW
720 POSITION C,M:? #6;" ";:REM ERASE T
HE CHARACTER
730 IF A=43 THEN C=C-1:IF C<2 THEN C=C
+1:REM CAN'T WRAP AROUND
740 IF A=42 THEN C=C+1:IF C>17 THEN C=
C-1:REM CAN'T WRAP AROUND THIS WAY EIT
HER
750 POSITION C,M:IF C/2=INT(C/2) THEN
? #6;CH$;:GOTO 760:REM MOVE THE CHARAC
TER
755 ? #6;CH1$
760 GOSUB 2000:IF P=1 AND C>=4 THEN IF
 T(C-3,M-2) THEN T(C-3,M-2)=0:TA=1:GOT
O 780
770 IF P=2 AND C<=16 THEN IF T(C-3,M-2
) THEN T(C-3,M-2)=0:TA=1
780 Y=0:FOR X=1 TO 13:Y=Y+T(X,M-2):NEX
T X:REM CHECK FOR EMPTY ROW
790 IF Y THEN 600:REM CAN STILL TAKE-A
WAY
800 FOR X=1 TO 13:FOR Y=1 TO 7:IF T(X,
Y)=0 THEN NEXT Y:NEXT X:GOTO 850:REM C
HECK FOR ALL EMPTY
810 POSITION C,M:? #6;" ";:RETURN
850 POSITION 5,M+1:? #6;"THE WINNER"
860 POSITION 5,M+2:? #6;"PLAYER  ";P
870 POKE 708,PEEK(708)+1:IF PEEK(708)=
255 THEN POKE 708,0:REM CHANGE THE COL
ORS
880 IF PEEK(53279)<>6 THEN 870:REM CHA
NGE COLORS UNTIL START KEY IS PRESSED
890 RUN
900 RO=0:TT=0:E=0:F=0:T=0:O=0:REM CLEA
R THE COUNT VARIABLES
```

```
910 FOR Y=1 TO R:FOR X=1 TO 13:TT=TT+T
(X,Y):NEXT X:R(Y)=TT:REM COUNT THE NUM
BER OF CHAR/ROW
920 IF TT<>0 THEN RO=RO+1:REM SEE HOW
MANY ROWS
930 TT=0:NEXT Y:IF RO=1 THEN 1750
940 FOR Y=1 TO R:IF R(Y)>=8 THEN E=E+1
:R(Y)=R(Y)-8:REM GET THE 8'S
950 IF R(Y)>=4 THEN F=F+1:R(Y)=R(Y)-4:
REM GET THE 4'S
960 IF R(Y)>=2 THEN T=T+1:R(Y)=R(Y)-2:
REM GET THE 2'S
970 IF R(Y)=1 THEN O=O+1:R(Y)=0:REM GE
T THE 1'S
980 NEXT Y:FOR Y=1 TO R:FOR X=1 TO 13:
TT=TT+T(X,Y):NEXT X:R(Y)=TT:TT=0:NEXT
Y:REM COUNT THE NUMBERS AGAIN
990 OD=0:IF E/2<>INT(E/2) THEN OD=8
1000 IF F/2<>INT(F/2) THEN OD=OD+4
1010 IF T/2<>INT(T/2) THEN OD=OD+2
1020 IF O/2<>INT(O/2) THEN OD=OD+1
1030 IF  NOT OD THEN 1800
1040 IF OD=8 OR OD=4 OR OD=2 OR OD=1 T
HEN 1190
1050 OD=0:IF E<>1 THEN 1100
1060 IF F/2<>INT(F/2) THEN OD=4
1070 IF T/2<>INT(T/2) THEN OD=OD+2
1080 IF O/2<>INT(O/2) THEN OD=OD+1
1090 GOTO 1190
1100 IF F/2=INT(F/2) THEN 1140
1110 IF T/2<>INT(T/2) THEN OD=OD+2
1120 IF O/2<>INT(O/2) THEN OD=OD+1
1130 GOTO 1190
1140 IF T/2=INT(T/2) THEN 1170
1150 IF O/2<>INT(O/2) THEN OD=OD+1
1160 GOTO 1190
1170 IF O/2=INT(O/2) AND OD<>0 THEN 18
00
1180 OD=OD+1
1190 FOR Y=R TO 1 STEP -1:IF R(Y)>OD T
HEN 1840
1200 NEXT Y:GOTO 1810:REM NON LARGER
1750 FOR Y=1 TO R:IF R(Y)=0 THEN NEXT
Y:END :REM BBBBBBBBUUUGGGG!!!!
1760 OD=R(Y):GOTO 1840:REM ERASE THE W
HOLE ROW
```

```
1800 OD=3:REM REMOVE 3 IF IT'S EVEN
1810 FOR Y=R TO 1 STEP -1:IF R(Y)>=OD
THEN 1840
1820 NEXT Y:REM KEEP LOOKING
1830 OD=1:GOTO 1810:REM NO ROW HAS 3 O
R MORE
1840 FOR X=M TO Y+2:POSITION C,X-1:? #
6;" ":REM ERASE THE LAST CHARACTER
1850 POSITION C,X:? #6;CH$:REM MOVE TH
E CHARACTER
1860 FOR TIME=1 TO 100:NEXT TIME:NEXT
X:X=X-1
1870 POSITION C,X:? #6;" ":REM ERASE T
HE CHARACTER
1880 C=C-1:POSITION C,X:IF C/2=INT(C/2
) THEN ? #6;CH$:GOTO 1885
1881 ? #6;CH1$
1885 GOSUB 2000:FOR TIME=1 TO 50:NEXT
TIME
1890 IF C<=16 THEN IF T(C-3,X-2) THEN
T(C-3,X-2)=0:OD=OD-1:IF OD=0 THEN 1910

1900 GOTO 1870
1910 FOR Z=1 TO 13:FOR A=1 TO 7:IF T(Z
,A)=0 THEN NEXT A:NEXT Z:M=X:POP :GOTO
 850:REM CHECK FOR ALL EMPTY
1920 POSITION C,X:? #6;" ":RETURN
2000 SOUND 0,10,12,6:SOUND 1,2,2,5:FOR
 TIME=1 TO 5:NEXT TIME:SOUND 0,0,0,0:S
OUND 1,0,0,0:RETURN
2010 SOUND 0,SO,C/2,8:FOR TIME=1 TO 50
:NEXT TIME:SOUND 0,0,0,0:FOR TIME=1 TO
 50:NEXT TIME:RETURN
```

Line 320 checks to see if the value of R is greater than 127. If it is, the inverse key was pressed. The computer subtracts 128 from this value and resets the flag for normal video by POKEing location 694 with 0.

Line 330 checks to see if the value of R is between 51 and 55. If it isn't, the pressed key was not between 3 and 7. The computer goes to line 310 to get another key.

Line 340 prints this number on the screen. To get the actual number, the computer subtracts 48 from the value or R.

Lines 350-370 ask who will be the first player. Enter a 1 for Player One, a 2 for Player Two.

Line 380 opens the keyboard for a read. The computer waits until a key has been pressed. The value of this key is stored in variable P. The keyboard is closed.

Line 390 checks to see if the value of P is greater than 127. If it is, the computer subtracts 128 from the value of P. The location 694 is POKEd with 0. This resets the flag for normal video.

Line 400 checks to see if the value of the key is 49 or 50. The one key has a value of 49, the two key the value of 50. If the value of P is either, the computer goes to line 420 to begin the program.

Line 410 sends the computer to line 380 to get another entry.

Line 420 subtracts 48 from the value of P. This is the actual number of the pressed key.

Line 430 begins a FOR-NEXT loop that clears the array. If the array is not cleared before it is used, it may contain erroneous information.

Line 440 fills the array. Variable V indicates how many elements are used in that row. There is one character in the first row and two additional characters in each subsequent row.

Line 450 clears the screen with the graphics command. Every time this command is used, the address of the new character set must be placed in location 756.

Line 460 prints the title of the program near the top of the screen.

Line 470 begins the FOR-NEXT loop that places the characters into the array. The computer chooses a random number from 0 to 4. There are four characters that can be used.

Line 480 checks to see if there is a value in the element of the array to which X and Y are pointing. If there is, the character is printed on the screen. The value of Z is increased by 3.

Line 490 continues the loop.

Line 500 prints the number near the top of the screen.

Line 510 sets variables C1, R1, C2, and R2. These variables indicate the row and column in which are the characters for Players One and Two.

Line 520 prints the characters for both players on the screen in their positions.

Line 530 sends the computer to the correct line depending on the value of P. Variable P indicates whose turn it is.

Line 540 sets the value of variable TA to 0. This variable will change to 1 when a character is taken. The column and row of the character are transferred to variables C and M. The characters that make up this player are placed in strings CH$ and CH1$. This

player's number is printed on the screen and the other player's number is removed. The computer uses the subroutine in line 590 to continue this player's turn. When the computer returns to this line, the value of P will be subtracted from 3 so P will contain the number of the next player.

Line 550 clears variable TA and sets variables C and M to the values of C2 and R2. This is the row and column of the second player's character. The characters that make up this character are placed in CH$ and CH1$. The number for this player is placed on the screen and the opponent's number is erased.

Line 560 checks to see if variable K is 72. If it is, another human is playing and the computer goes to the subroutine at line 590 to get this player's move. When the computer returns, the value of P is reset for the other player and the computer goes to line 540 for the first player's move.

Line 570 is used when the computer is playing. Variables TT and TR are cleared. The computer uses the subroutine in line 900 to figure out how many characters to take, and in which row. When the computer returns to this line it is sent to line 540 for the first player's turn.

Line 590 begins the subroutine used by the players to move their character. The character is printed on the screen in the correct position.

Line 600 opens the keyboard for a read. The computer waits until a key is pressed. The value of this key is stored in variable A. The keyboard is closed. If the value of A is greater than 127, the inverse key was pressed and the computer POKEs location 694 with 0 and sends the computer back to the beginning of the line to get another key. Location 694 is the inverse flag. POKEing 0 to this location clears the flag.

Line 610 checks to see if the value of A is 45 or 61. If it is, the up or down arrow key was pressed and the computer is directed to line 650 to move the character.

Line 620 checks the value of A for 43 or 42. If it is either, the computer goes to line 700 to move the character up.

Line 630 checks to see if the space bar was pressed. If it was and the value of TA is not 0, the computer will erase the character and return to the main program.

Line 640 sends the computer to line 600. The pressed key was not one of the four arrow keys.

Line 650 checks to see if the character is in the first or last column. If it is, it can be moved up or down. If it isn't, the com-

puter goes to line 600 for another entry.

Line 660 erases the character in its current position.

Line 670 checks to see if the value of A is 45. If it is the value of M is decreased by 1. The character will move up the screen. If the character has reached the top of the screen, it will wrap around and be repositioned on the bottom of the screen. The computer is sent to line 690 to print the character on the screen.

Line 680 checks to see if the value of A is 61. If it is, the character will move down the screen. Once the character reaches the bottom of the screen, it is repositioned at the top of the screen.

Line 690 sends the computer to line 590 to print the character on the screen and get another direction.

Line 700 is used when the right or left arrow key is pressed. Variable Y is cleared. The computer checks the elements in the array for this row. If any are not 0, the value of Y will be changed. The loop continues until the entire row is checked.

Line 710 checks the value of Y. If it is still zero, the row is empty and the computer goes back to line 600 to get another entry.

Line 720 erases the character at its current position.

Line 730 checks the value of A. If it is 43, the character should move to the left. The value of C is decreased by 1. If it is less than 2, it would be off the screen. The characters do not wrap-around the screen. The value of C is reset.

Line 740 checks to see if the right arrow was pressed. If it was, the value of C is increased by 1. Again, this value is checked to see if it would place the character off the screen. If it would, the variable C is reset.

Line 750 prints the character on the screen. There are two characters that can be used. If variable C is even, the character in CH$ is used and the computer goes to line 760 after the character is printed.

Line 760 prints the alternate character.

Line 770 uses the subroutine in line 2000 to make a sound. If variable P is 1 and the value of C is equal to or greater than 4, the computer checks the array to see if the character took a character away from the row. If there is a value in this element, it is erased and variable TA is set to 1. The computer goes to line 780 to continue the game.

Line 770 checks to see if this is the second player's turn. If it is and this player is in a position that could be held by a row character, the computer checks the element of the array to see if it is occupied. If it is, the character in that position is erased and

variable TA is set to 1.

Line 780 clears variable Y. Each element of the array at this location is checked to see if it is occupied. If any are, the value of Y changes.

Line 790 sends the computer back to line 600 for another entry if there are any characters left in this row.

Line 800 checks the entire array. If the entire array is empty, the game is over and the computer goes to line 850 to end the game.

Line 810 erases the character from the screen and returns to the main program. The most characters that can be taken in one turn is the number of characters the row held. Once the row is empty, that player's turn is over.

Lines 850-890 end the game. The computer declares the winner based on the value of P. The computer loops, changing the color of the screen until the Start key is pressed. To end the program, press the System Reset key.

Line 900 is used by the computer when it is playing the game. Variables RO, TT, E, F, T, and O are set to 0. These variables are used to count in this subroutine.

Line 910 counts the number of characters left in each row, and places that number in the correct element of the R array. Variable TT counts how many characters are in a particular row.

Line 920 checks variable TT; if it is not 0, there are characters in that row and variable RO is increased by one. This variable counts how many rows have characters in them.

Line 930 clears variable TT and continues the loop. If variable RO is one, the computer goes to line 1750 to find that row.

Lines 940-980 look at all the rows to see how many characters are in each row. If the row has eight or more characters in it, variable E is increased by 1 and 8 is subtracted from this row's value. If the row has four or more characters in it, variable F is increased by 1 and 4 is subtracted from the number of characters in this row. If the row has two or more characters in it, variable T is increased by 1 and 2 is subtracted from the number of characters in this row. If there is only one character in this row, variable O is increased by 1 and the row is set to 0. A row with 13 characters in it would increment the E, F, and O variables. The loop continues until all the rows have been checked. The number of elements in all the rows are counted again.

Lines 990-1030 check to see if any of the values are odd. If so, this is the number of characters that could have been removed from a row. If none of the rows are odd the computer will remove one

character from a row.

Line 1040 checks to see if the value of OD is one particular value. If it is, the computer is sent to line 1190 to find the row that has this number of characters in it.

Lines 1050-1180 determine how many characters the computer will take. First the computer clears variable OD. If variable E is not 1, there are two or more rows with eight or more characters in it. The computer is sent to line 1100. The computer continues through the lines checking the values of E, F, T, and O. When the computer finally figures out how many characters it will remove, it goes to line 1190. The key to winning this program is to remove only enough characters to leave one set of eight, four, two, and one characters. This game is binary in nature.

Line 1190 looks for the row that contains the number of characters the computer wants to remove. Once this row is found, the computer is sent to line 1840.

Line 1200 continues the loop. If no row is found, the computer goes to line 1810.

Line 1750 is a trap. The computer comes to this line from line 930. Variable RO is set. If the computer goes through this entire loop and cannot find the row there is a bug in the program. The only time this line is used is when there is only one row on the screen that has characters in it. The computer wants to erase the entire row so it can win the game.

Line 1760 erases the entire row of characters by setting variable OD to the value of that row. The computer is sent to line 1840 to erase the row of characters.

Line 1800 sets the value of OD to 3 if there is an even number of ones.

Lines 1810-1830 looks for a row that has that number or more of the characters indicated by variable OD. If the computer cannot find this number of characters, the value of OD is changed to 1 and the loop is tried again.

Lines 1840-1900 remove the characters from the row by moving the computer's character over the existing character on the screen. These lines continue until all the characters that should be removed are removed.

Line 1910 checks to see if there are any characters left on the screen. If all the characters have been removed, the computer is sent to line 850 to end the game.

Line 1920 erases the computer's character and returns to the main part of the program.

Lines 2000-2010 are the two sound routines used by the computer.

SYMBOLIZE

Objective of the game: To figure out the symbol and color sequence the computer is thinking of.

Directions: This program is written for one or two players. In the two-player mode, each player has a different color and symbol sequence to work out. The first person to guess the correct sequence wins. If the players cannot guess the correct sequence within ten tries the computer prints the answers.

The computer asks for the number of players. Enter numeral 1 or 2. Then the computer displays the menu for three levels. In the first level, the computer chooses from two different symbols. These symbols can be any of four colors. The second level uses four symbols, and the third level uses six. Press 1, 2, or 3 for the level number.

The screen will clear. Near the bottom of the screen, the computer prints the symbols and four colors this level uses. To enter your guess, press the letter key that represents the first letter of the color you want. To change the color, press another color key. Press the Return key to keep the color. Now press the number key for the symbol. Do not press the Shift key, just the number key. The symbol appears on the screen in the color indicated. To change the symbol, press a different number. To enter symbol, press the Return key.

The computer displays two numbers. The first number indicates how many symbols are correct in color, and in correct position. The second number tells you how many symbols in the correct color are in the wrong position. If, after 10 tries you cannot enter the correct symbol sequence in the correct colors, the computer will tell you what the code was.

To play again, press the Start key. To quit, press System Reset. Figure 4-5 is the flowchart for this program, and Listing 4-3 is the code.

Line 50 sets aside memory space for the strings used in this program. SYM$ contains the code for the first player, SYM2$ the code for the second player. GUESS$ holds the symbols the player entered. TEMP$ is a temporary storage string.

Line 60 changes the screen to graphics 18. This is graphics 2 with no text window.

Line 70 prints a message on the screen. The computer needs the number of persons playing this game.

Line 80 opens the keyboard to read a keystroke. The value of the pressed key is stored in variable P. The keyboard is closed after a key is pressed.

Line 90 checks to see if the 1 or 2 key was pressed. If it was not, the computer goes back to line 80 to get another entry. The computer loops through these two lines until the proper key is pressed.

Line 100 prints the entered number on the screen immediately following the question.

Line 110 subtracts 48 from the value of P to get the actual number. The ATASCII value of 1 is 49, the value of 2 is 50.

Lines 120-140 print the menu on the screen. The three levels of play and the combinations the computer uses at each level are displayed on the screen.

Line 150 opens the keyboard to read a keystroke. The value of the pressed key is stored in variable L. The keyboard is closed after a key is pressed.

Line 160 checks to see if the value of the pressed key is between 49 and 51. If it is not, the computer goes back to line 150 to get another keystroke.

Line 170 subtracts 48 from the value of L. The result is multiplied by two. This is the number of symbols the computer will use for the code.

Line 180 begins the loop that chooses the code for the computer. The first loop counts from 1 to the value of P. This is the number of players for which the computer needs code. The second loop counts from 1 to 4. There are four codes in one sequence. Variable C holds the color of the symbol. The computer has four colors to choose from. Variable S is the symbol for this position. The number of symbols the computer can choose from is determined by variable L. The same symbol and the same color can be repeated in the line.

Line 190 sends the computer to the correct program line, depending on which color was chosen.

Line 200 uses the code for the symbol without adding a number to it. This symbol is colored light green.

Line 210 adds 32 to the value of S. This symbol is yellow.

Line 220 adds 128 to the value of S. The symbol is red.

Line 230 adds 160 to the value of S. This symbol is colored blue.

Line 240 continues the loop for the next position in GUESS$.

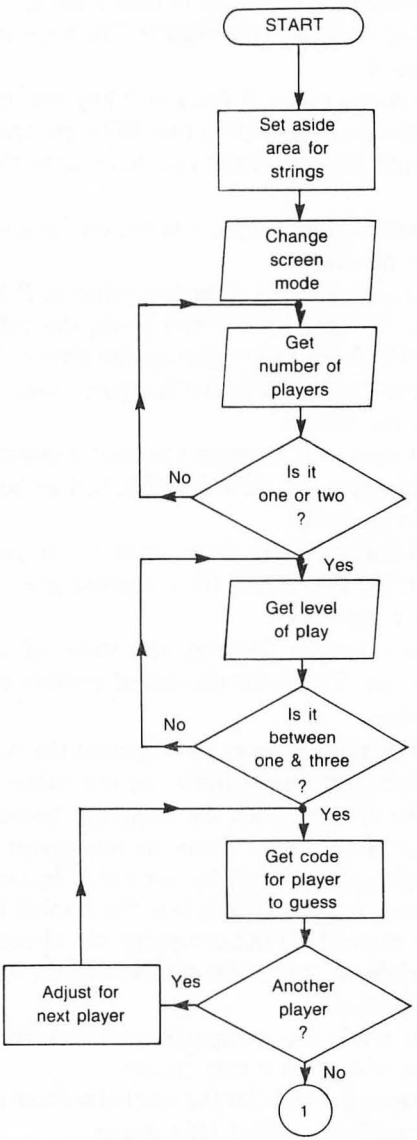
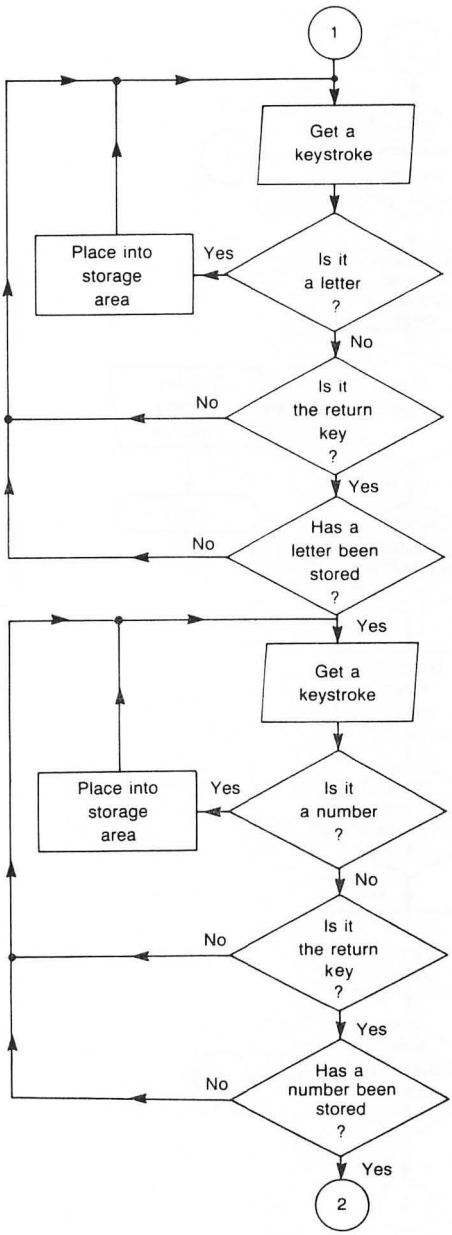
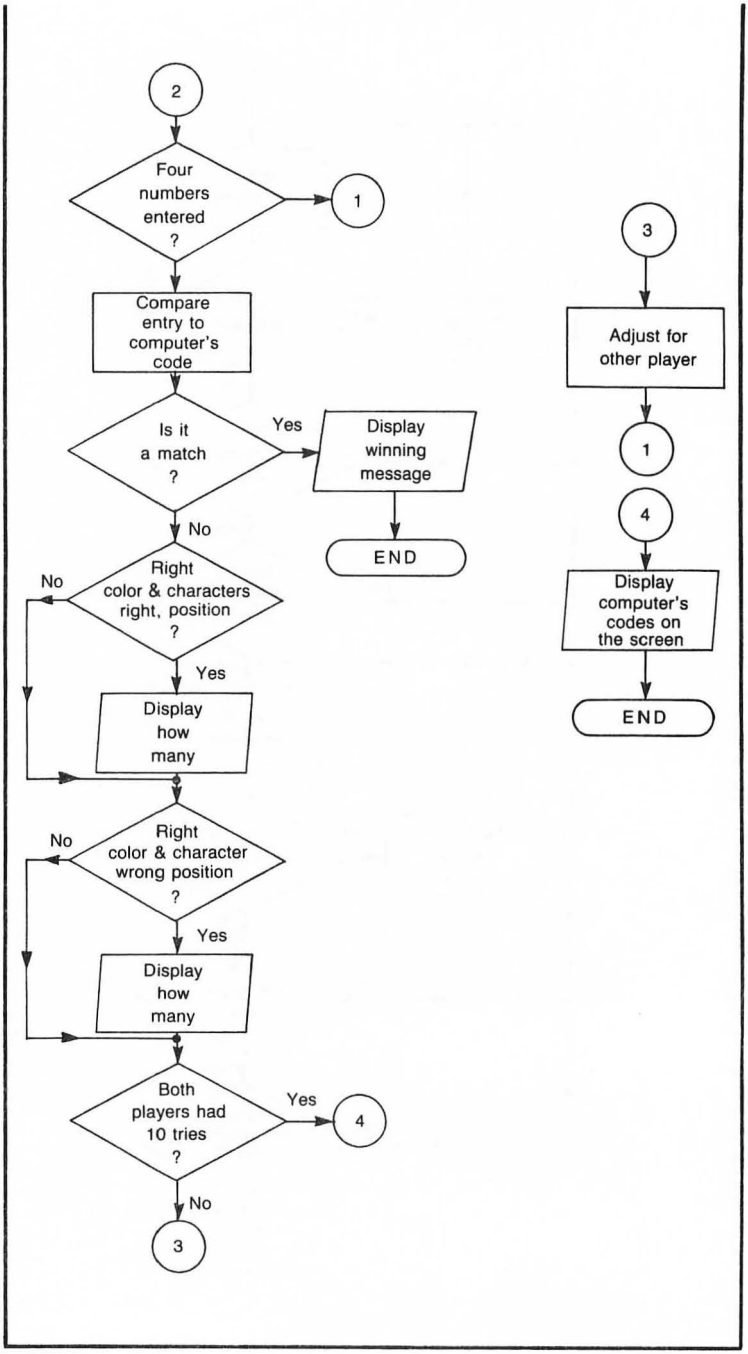


Fig. 4-5. Flowchart for Symbolize. (Continued through page 206.)


1
Get a keystroke
Is it a letter ?
Yes
Place into storage area
No
Is it the return key ?
No
Yes
Has a letter been stored ?
No
Yes
Get a keystroke
Is it a number ?
Yes
Place into storage area
No
Is it the return key ?
No
Yes
Has a number been stored ?
No
Yes
2


2
Four numbers entered ?
1
Compare entry to computer's code
Is it a match ?
Yes
Display winning message
END
No
Right color & characters right, position ?
No
Yes
Display how many
Right color & character wrong position ?
No
Yes
Display how many
Both players had 10 tries ?
Yes
4
No
3
3
Adjust for other player
1
4
Display computer's codes on the screen
END

Listing 4-3. Symbolize.

```
10 REM SYMBOLIZE - A 'MASTERMIND' GAME
 USING COLORS AND SYMBOLS
20 REM CHAPTER 4 - LOGIC GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1984
50 DIM SYM$(4),SYM2$(4),GUESS$(4),TEMP
$(4)
60 GRAPHICS 18:REM GRAPHICS 2 WITH NO
TEXT WINDOW
70 POSITION 1,1:? #6;"HOW MANY PLAYERS
?";
80 OPEN #2,4,0,"K:":GET #2,P:CLOSE #2:
REM GET THE NUMBER OF PLAYERS
90 IF P<49 OR P>50 THEN 80:REM CAN ONL
Y BE ONE OR TWO PLAYERS
100 POSITION 19,1:? #6;CHR$(P):REM SHO
W THE NUMBER
110 P=P-48:REM GET THE ACTUAL NUMBER
115 REM PRINT MENU IN INVERSE
120 POSITION 1,3:? #6;"WHICH LEVEL (1-
3)?":POSITION 1,4:? #6;"LEVEL 1 - 2 SY
MBOLS             4 COLORS"
130 POSITION 1,6:? #6;"LEVEL 2 - 4 SYM
BOLS             4 COLORS"
140 POSITION 1,8:? #6;"LEVEL 3 - 6 SYM
BOLS             4 COLORS"
150 OPEN #2,4,0,"K:":GET #2,L:CLOSE #2
:REM GET THE LEVEL
160 IF L<49 OR L>51 THEN 150:REM ONLY
THREE LEVELS
170 L=(L-48)*2:REM GET THE RIGHT NUMBE
R OF SYMBOLS
180 FOR X=1 TO P:FOR PL=1 TO 4:C=INT(R
ND(1)*4)+1:S=INT(RND(1)*L)+1:REM PICK
A COLOR AND SYMBOL
190 ON C GOTO 200,210,220,230
200 GUESS$(PL,PL)=CHR$(S):GOTO 240
210 GUESS$(PL,PL)=CHR$(S+32):GOTO 240
220 GUESS$(PL,PL)=CHR$(S+128):GOTO 240
230 GUESS$(PL,PL)=CHR$(S+160)
240 NEXT PL:SYM2$=GUESS$:IF X=1 THEN S
YM$=GUESS$
250 NEXT X
260 ? #6;CHR$(125):POSITION 0,11:FOR X
```

```
=1 TO L:? #6;CHR$(X+32);" ";:NEXT X
270 ? #6;"O g B r";:REM 'B' AND 'r' AR
E INVERSE
280 FOR X=1 TO P:POSITION (X-1)*10+5,0
:? #6;X:NEXT X:TRY=0
290 TRY=TRY+1:FOR PP=1 TO P:FOR X=1 TO
 4
300 POSITION X-1+((PP-1)*10),TRY:? #6;
"?":GOSUB 1000
310 GUESS$(X,X)=CHR$(C):NEXT X:REM GET
 THE GUESS
320 POSITION 0,9:IF PP=1 THEN TEMP$=SY
M$:IF GUESS$=SYM$ THEN ? #6;"  ";PP;"
GOT IT!!":GOTO 380
330 IF PP=2 THEN TEMP$=SYM2$:IF GUESS$
=SYM2$ THEN ? #6;"  ";PP;" GOT IT!!":G
OTO 380
340 GOSUB 500
350 NEXT PP:IF TRY<>10 THEN 290
360 POSITION 0,11:? #6;"
    ";:REM CLEAR THE BOTTOM ROW
370 POSITION 0,10:? #6;"1  ";SYM$;:IF
PL=2 THEN POSITION 10,10:? #6;"2  ";SY
M2$;
380 IF PEEK(53279)<>6 THEN 380
390 RUN
500 RP=0:RS=0:FOR V=1 TO 4
510 IF TEMP$(V,V)=GUESS$(V,V) THEN RP=
RP+1:TEMP$(V,V)=" ":GUESS$(V,V)=" "
520 NEXT V
530 FOR V=1 TO 4:IF TEMP$(V,V)=" " THE
N NEXT V:GOTO 570
540 FOR V1=1 TO 4:IF TEMP$(V,V)=GUESS$
(V1,V1) THEN TEMP$(V,V)=" ":GUESS$(V1,
V1)=" ":RS=RS+1:GOTO 560
550 NEXT V1
560 NEXT V
570 POSITION (PP-1)*10+5,TRY:? #6;RP;"
 ";CHR$(RS+48+128):RETURN
1000 FOR ZZ=1 TO 2:SOUND 0,50,10,10:FO
R TI=1 TO 50:NEXT TI
1010 SOUND 0,0,0,0:FOR TI=1 TO 50:NEXT
 TI:NEXT ZZ
1020 C=0:S=0:FOR PL=1 TO 2:GOSUB 1100:
NEXT PL:RETURN
1100 OPEN #2,4,0,"K:"
1110 GET #2,K:IF K=155 THEN CLOSE #2:R
```

```
ETURN
1120 IF K>127 THEN K=K-128:POKE 694,0:
REM RESET TO NORMAL VIDEO
1130 IF K>95 THEN K=K-96:POKE 702,64:R
EM SET FOR UPPER CASE
1140 IF PL=1 THEN IF K=66 OR K=82 OR K
=79 OR K=71 THEN 1170
1150 IF PL=2 THEN IF K>48 AND K<49+L T
HEN K=K+S-16:GOTO 1200
1160 GOTO 1110
1170 S=0:IF K=66 THEN K=194:S=128
1180 IF K=82 THEN K=242:S=96
1190 IF K=71 THEN K=103:S=-32
1200 POSITION X-1+((PP-1)*10),TRY:? #6
;CHR$(K):C=K:GOTO 1110
```

The code in GUESS$ is placed into SYM2$. If the value of X is 1, the contents of GUESS$ are placed in SYM$.

Line 250 continues the loop until both players have their own code to guess.

Line 260 clears the screen. The characters the player can choose from are printed near the bottom of the screen. The number of characters are determined by the value of L.

Line 270 prints the four colors on the screen. Be sure to enter the uppercase "B" and the lowercase "r" in inverse. Otherwise, the letter will not appear in four colors.

Line 280 prints the number of players on the screen. The players will enter their guesses under their number. Variable TRY is set to 0. This is the number of tries the players have had.

Line 290 increases variable TRY. This is the current turn number for the players. The FOR-NEXT loop accepts one guess from each player. The X is the position of the entry both on the screen and in GUESS$.

Line 300 prints the question mark on the screen. The column of the question mark is determined by the value of X and the player number. The row is set by the value of TRY. The computer is sent to the subroutine in line 1000 to get an entry. This subroutine also prints the entry on the screen.

Line 310 records in GUESS$ the entered character. The element of GUESS$ that this character will occupy depends on the value of X. The loop continues until all four characters are entered.

Line 320 places the code chosen by the computer in TEMP$. If the code entered by the player is the same as the code the com-

puter has in SYM$, the player guessed the code. The computer goes to line 380 to end the game.

Line 330 places the code the computer has chosen for the second player in TEMP$. Then the computer checks the code the player entered against its code. If both are the same, this player has guessed the code.

Line 340 sends the computer to the subroutine at line 500 where the computer checks the characters the player entered against the characters the computer has in its string.

Line 350 continues the loop. If the value of TRY is not 10, the computer goes back to line 290 for another turn. If the value of TRY is 10, the computer prints the answer on the screen.

Line 360 clears the bottom row on the screen so the answer can be printed there.

Line 370 prints the code the computer chose for each player. If there was only one player, the computer will print the code for that player.

Line 380 is the end of the program. The computer loops at this line until the Start key is pressed.

Line 390 runs the program after the Start key is pressed. To end the game completely, press the System Reset key.

Line 500 is the subroutine that checks the player's answer against the code the computer chose for that player. The first number indicates the number of symbols that are the right color and in the right position. The second number is the number of symbols that are the right color but in the wrong position. Variables RP and RS are cleared. These are the variables that count the number of characters in the right position, or are right but in the wrong position. The FOR-NEXT loop looks at each character in the string.

Line 510 checks the character in one string against the character in the same position in the other string. If both characters are the same, variable RP is increased by 1 and the characters are removed from both strings.

Line 520 continues the loop until every character in one string is compared against the character in the same position in the other string.

Lines 530-560 begin the second FOR-NEXT loop. This time the computer compares the characters in the other positions of the other string. If any of these characters are the same, the RS variable is increased by 1 and the characters are removed from both strings. Before the computer compares the characters it checks to see if

there is a character in that position. If there isn't, the loop continues.

Line 570 prints the values of RP and RS on the screen. The computer then returns to the main program.

Lines 1000-1200 contain the subroutine that gets the character entry from the player. The computer makes a sound to prompt the player. Then the computer is sent to the subroutine at line 1100. Here the keyboard is opened for a read. The value of the pressed key is stored in variable K. If the value of K is 155, the keyboard is closed and the computer returns to the line that sent it. If the value of K is greater than 127, the computer subtracts 128 from it and POKES a 0 into location 694 to reset the inverse flag. If the value of K is greater than 95, the key is in lowercase. The computer subtracts 96 from this value and POKES location 702 with 64. This resets the keyboard for uppercase. If this is the first entry from the player, the computer accepts only the letters O, B, Y, and R. If it is the second entry, the computer accepts only the numbers that correspond with the symbols the computer can use at this level. On the first entry, the computer sets the value of S to the entered color. This value is added to the value of the symbol so the computer will print the symbol in the correct color on the screen. This routine loops until the Return key is pressed twice, once to accept the color, and once for the symbol.

TOWERS OF HANOI

Objective of the game: To move the entire column of disks from the first spindle to the last. The disks must be moved one at a time, without placing a larger disk on a smaller one.

Directions: At first, this program appears to be a very simple, logical puzzle. However, if you lose your concentration, you may lose the puzzle. On the screen you will see three spindles or poles. The pole on the left side of the screen contains nine disks. The disk on the bottom is the largest, the disk on the top is the smallest. Above the pole is a small magnet. A joystick is needed to move the magnet. The joystick uses the first port. To move the magnet, push the joystick to the right or left. To raise or lower the disk, press the fire button on the joystick. When the disk is attached to the magnet, it will move across the screen when the magnet is moved.

The computer keeps track of how many moves you made. Compete with your friends to see who can transfer all the disks from one spindle to the other in the shortest number of moves. Figure 4-6 is the flowchart, Fig. 4-7 is the character set for this program,

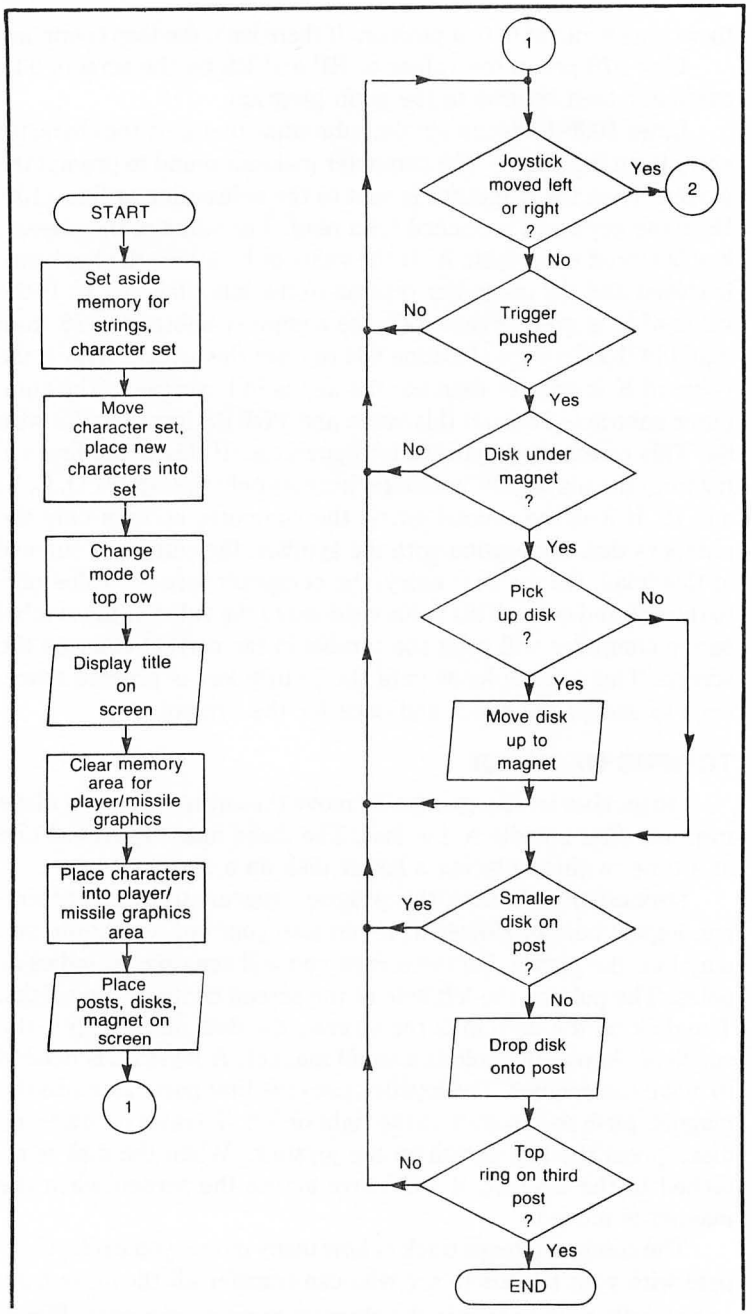
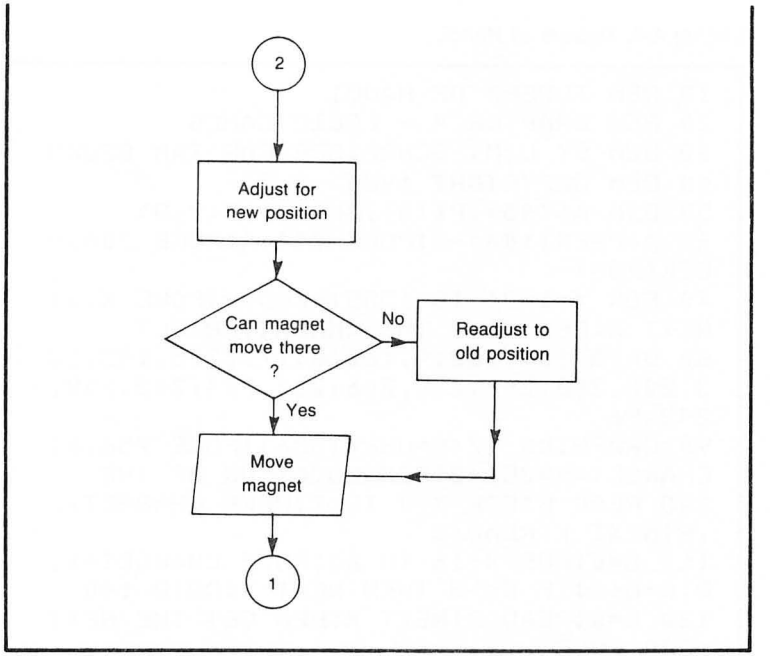


Fig. 4-6. Flowchart for Towers of Hanoi.



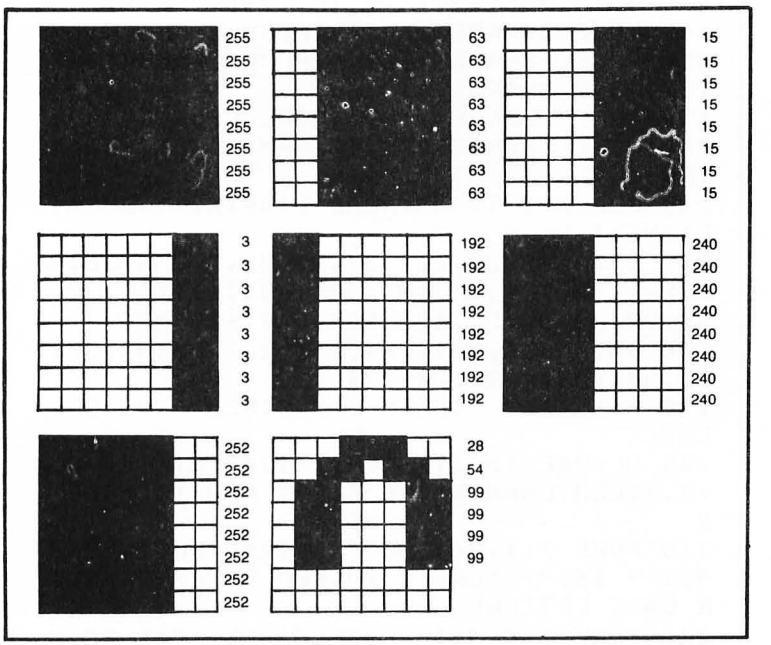


Fig. 4-7. Character set for Towers of Hanoi.

Listing 4-4. Towers of Hanoi.

```
10 REM TOWERS OF HANOI
20 REM CHAPTER 4 - LOGIC GAMES
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1984
50 DIM A$(45),P$(5),BL$(5),P(9,3)
60 A=PEEK(106)-8:POKE 204,A:POKE 206,P
EEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256+8:REM LOCATION OF THE !
100 READ B:FOR X=0 TO 7:POKE CHARSET+X
,B:NEXT X:READ B
110 C=0:FOR X=16 TO 63:POKE CHARSET+X,
B:C=C+1:IF C<>8 THEN NEXT X:GOTO 140
120 C=0:READ B:NEXT X:REM GET THE NEXT
 BAR CODE
130 DATA 255,63,15,3,192,240,252,0
140 POKE CHARSET+64,60:POKE CHARSET+65
,192:POKE CHARSET+66,6:POKE CHARSET+67
,27:POKE CHARSET+68,28
150 POKE CHARSET+69,5:POKE CHARSET+70,
204:POKE CHARSET+71,14:REM MAKE CHARAC
TER
160 FOR X=1 TO 45:READ B:A$(X,X)=CHR$(
B):NEXT X
170 DATA 32,32,161,32,32,32,37,33,38,3
2,32,132,129,135,32,32,163,161,168,32,
32,33,33,33,32,133,129,129,129,134
180 DATA 164,161,161,161,167,35,33,33,
33,40,129,129,129,129,129
190 P$="     ":BL$=P$:M=0:REM FIVE SPA
CES
200 DL=PEEK(560)+PEEK(561)*256:POKE DL
+6,7:REM CHANGE THE FIRST ROW TO MODE
2
210 POKE 711,86:POKE 709,244:POSITION
4,1:? #6;") towers )":REM INVERSE LOWE
R CASE LETTERS
220 POKE 559,46:FOR X=CHARSET+512 TO C
HARSET+1024:POKE X,0:NEXT X:REM CLEAR
```

```
PLAYER/GRAPHICS AREA
230 POKE 53277,3:POKE 623,4:POKE 54279
,A:REM ENABLE P/M GRAPHICS & SET PRIOR
ITIES
240 FOR S=512 TO 768 STEP 128:FOR X=60
 TO 105:POKE CHARSET+S+X,30:NEXT X:NEX
T S:REM DRAW THE POSTS
250 FOR S=571 TO 827 STEP 128:POKE CHA
RSET+S,12:NEXT S
260 FOR X=CHARSET+927 TO CHARSET+932:R
EAD S:POKE X,S:NEXT X
270 DATA 28,54,99,99,99,99
280 FOR X=1 TO 19:POSITION X,23:? #6;C
HR$(1);:NEXT X:REM DRAWS THE BASE
290 POKE 704,202:POKE 705,44:POKE 706,
108:POKE 707,9:REM COLOR THE POSTS
300 POKE 53248,71:POKE 53249,127:POKE
53250,183:POKE 53251,71:PS=71:REM PUT
POSTS & MAGNET ON SCREEN
310 FOR X=1 TO 9:P(X,1)=X:P(X,2)=0:P(X
,3)=0:NEXT X:REM PUT DISK VALUES INTO
STORAGE AREA
320 FOR X=22 TO 14 STEP -1:POSITION 1,
X:P=(X-14)*5+1:? #6;A$(P,P+4):NEXT X:P
L=1:R=0:REM PUT DISKS ON POST
330 POSITION 2,3:? #6;"MOVES  ";M
340 IF STICK(0)=7 THEN GOSUB 400:POKE
77,0:REM MOVE TO RIGHT
350 IF STICK(0)=11 THEN GOSUB 450:POKE
 77,0:REM MOVE TO LEFT
360 IF STRIG(0)=0 THEN GOSUB 500:POKE
77,0:REM DROP OR PICK UP DISK
370 FOR X=1 TO 50:NEXT X:SOUND 0,0,0,0
:IF P(1,3)=0 THEN 330:REM TIMING LOOP
& START AGAIN
380 FOR X=1 TO 15:FOR B=0 TO 8:POKE 70
4+B,X+B:SOUND 0,20*B,10,10:NEXT B:FOR
T1=1 TO 10:NEXT TI:NEXT X
390 GOTO 380
400 IF PS<183 THEN PS=PS+56:POKE 53251
,PS:POSITION PL,R:? #6;BL$:PL=PL+7
410 POSITION PL,R:? #6;P$
420 RETURN
450 IF PS>71 THEN PS=PS-56:POKE 53251,
PS:POSITION PL,R:? #6;BL$:PL=PL-7:GOTO
 410
```

```
460 RETURN
500 C=0:IF PS=71 THEN C=1:PL=1:REM FIN
D WHICH POST THE MAGNET'S OVER
510 IF PS=127 THEN C=2:PL=8
520 IF PS=183 THEN C=3:PL=15
530 IF C=0 THEN RETURN
540 IF R=6 THEN 600:REM HOLDING A DISK
550 FOR X=1 TO 9:IF P(X,C)=0 THEN NEXT
 X:RETURN :REM FIND THE DISK
560 S=P(X,C):P=(S-1)*5+1:P$=A$(P,P+4):
R=X+13:P(X,C)=0
570 FOR X=R TO 7 STEP -1:POSITION PL,X
:? #6;BL$:POSITION PL,X-1:? #6;P$:NEXT
 X:R=X:SOUND 0,10,10,10
580 RETURN
600 FOR X=1 TO 9:IF P(X,C)<S AND P(X,C
)<>0 THEN RETURN :REM DON'T DROP ON SM
ALLER DISK
610 IF P(X,C)<>0 THEN P=X-1:GOTO 630
620 NEXT X:P=X-1
630 FOR X=R TO R+6+P:POSITION PL,X:? #
6;BL$:POSITION PL,X+1:? #6;P$:NEXT X:R
=0:P$=BL$:SOUND 0,250,14,8:M=M+1
640 P(P,C)=S:RETURN
```

and Listing 4-4 shows the code.

Line 50 sets aside the memory needed for strings and arrays in this program. A$ holds all the disk characters. P$ holds the characters for one disk. BL$ is entirely blank and erases the disk. The P array contains the values for the disks on each spindle.

Line 60 finds out how much memory is in the computer. The new character set will occupy the memory just above the screen memory. The computer subtracts 8 from the amount of memory in the computer. This value is stored in location 204. The computer stores the beginning address of the ROM-based character set in location 206. This information will be used by the computer when it moves the character set from ROM into RAM.

Line 70 reads the code for the assembly language subroutine and places it in locations 1536-1555. Line 80 contains the decimal codes for the routine that moves the character set from ROM into RAM.

Line 90 changes the screen to graphics 17. This is graphics 1 with no text window. The computer uses the USR command to ex-

ecute the assembly language subroutine that begins at location 1536. When the computer returns to this line, the value of A is POKEd into location 756. Now the computer uses the character set in RAM. The first byte of the character in the character set that will be changed is stored in variable CHARSET.

Line 100 reads a value from the DATA line. This value is POKEd into the next eight bytes of the character set. The new character set is series of bars that make up the disks. Each bar is slightly shorter than the bar before it. This way, several bars can be put together. The beginning and end bar determine its length. Instead of reading eight codes for each character, the computer reads each code once and POKEs it into the next eight locations. The first character to be changed is the exclamation point. When the computer finishes this loop, it reads another value. It is ready to change the next character in the character set.

Line 110 clears variable C. This variable counts the number of codes that have been placed in the character set for a particular character. The FOR-NEXT loop begins with 16 and counts to 63. The next six characters, beginning with the pound sign, are changed. The computer POKEs the new value of B into the character set. Variable C is increased by 1. If the value of C is not 8, the loop continues, POKEing this value into the character set until it POKEs it eight times. When all the characters have been changed, the computer goes to line 140 to continue the program.

Line 120 is used when variable C is equal to 8. The variable is cleared and the next value is read. The loop continues until all the characters have been changed.

Line 130 contains the code for the seven characters. The last value, 0, is included but never used. The computer reads the next value before it continues the loop. After the seventh character is changed, the computer reads a value from the data line, then continues the loop. While the value of X is larger than 63, the program continues. If the zero wasn't there, the computer would try to read the next value in the DATA line. If there were no more values, the program would crash.

Lines 140-150 continue the program. One more character is changed in the character set. This character is the character printed on both sides of the word "towers."

Line 160 begins another FOR-NEXT loop. This time the computer reads the ATASCII values of the characters that make up the disks. All nine disks are placed in A$.

Lines 170-180 contain the codes for the disks.

Line 190 clears the contents of P$ and BL$. Variable M is set to 0. This is the variable that counts the number of moves the player makes.

Line 200 find the first byte of the display list. The address of the display list is found in locations 560 and 561. The address of the display list is stored in variable DL. The sixth byte after the first byte of the list is changed to a graphics 2 row. This row is twice as high as a graphics 1 row.

Line 210 changes some of the colors used in the program. The title of the program is printed on the screen.

Line 220 POKEs location 559 with 46. This tells the computer that the player/missile graphics will be displayed in double-line resolution. The computer clears all the memory used for the player/missile graphics.

Line 230 enables the player/missile graphics by POKEing location 53277 with 3. The priorities are set by POKEing location 623 with 4. The computer is told where the player/missile graphics begin by POKEing location 54279 with the value of A. This is the same value used for the character set. The character set occupies the first 512 bytes of this memory location, the player/missile graphics the next 512 bytes.

Line 240 places the code for the three poles into the area set aside for the player/missile graphics. The computer steps by 128 because the players are 128 bytes apart. The same code is used for all three poles.

Line 250 places the caps on the poles.

Line 260 places the code for the magnet into the fourth player's area.

Line 270 contains the code for the magnet.

Line 280 draws the base for the poles on the screen.

Line 290 sets the colors for the poles and the magnet.

Line 300 POKEs the position of the three poles and the magnet into locations 53248-53251. This places the players on the screen. Variable PS is set to 71. This is the position of the magnet.

Line 310 places the values 1 through 9 in the first part of P array. The other two columns are set to 0. The numbers 1 through 9 are the weights of the disks; the lightest disk is 1 and the heaviest is 9.

Line 320 places the disks on the first pole. The characters that make up the disk are taken from A$. The value of variable P is the position of the code in A$. The loop continues until all the disks are printed on the screen. Variable PL is set to 1 and the value

of R is set to 0. PL is the column in which the first disk is printed; R will contain the ROW.

Line 330 prints the number of moves on the screen. This variable is increased every time a disk is released. If a disk is raised and released over the same pole, the move is still counted.

Line 340 checks to see if the joystick has moved. If the value is 7, the stick was moved to the right and the computer uses the subroutine at line 400 to move the magnet. Location 77 is cleared to disable the ATTRACT mode. This way the screen will not change colors while you are trying to move the disks.

Line 350 checks to see if the joystick has moved to the left. If it has, the computer uses the subroutine at line 450 to move the magnet. The ATTRACT mode is disabled when the computer returns to this line.

Line 360 sends the computer to the subroutine at line 500 to pick up or drop a disk.

Line 370 contains a short timing loop. This gives the player a chance to release the fire button or the joystick. The sound is turned off and the computer checks to see if the smallest disk is in the top position of the third pole. If it is not, the computer goes to line 330.

Lines 380-390 end the program. The computer loops here, flashing the screen and making sounds until the System Reset key is pressed. This program does not run again unless the player types the word RUN after pressing the System Reset key.

Line 400 checks the value of PS. If it is less than 183 the magnet can be moved to the right. The value of PS is increased by 56 and this value is POKEd into location 53251. The disk under the magnet is moved along with the magnet.

Line 410 prints the disk under the magnet.

Line 420 sends the computer to the main program.

Line 450 checks the value of PS to see if the magnet can move to the left. If it can, the value of PS is decreased by 56 and this value is POKEd into location 53251. The disk under the magnet is erased and the computer is sent to line 410 to print the disk under the magnet.

Line 460 sends the computer back to the main program.

Line 500 is used to release or pick up a disk. Variable C is set to 0. The computer checks the value of PS to see which pole the magnet is over. If the value of PS is 71, the magnet is over the first pole. Variables C and PL are set to 1.

Line 510 checks to see if the magnet is over the second pole.

If it is, the value of PS is 127. Variable C will be set to 2 and the value of PL to 8.

Line 520 sets variable C to 3 and the variable PL to 15 if the magnet is over the third pole.

Line 530 checks the value of C. If it is 0, the magnet is not over a pole. The computer returns to the main program.

Line 540 checks the value of R. If it is 6, the magnet is holding a disk. The computer is sent to line 600 to drop the disk.

Line 550 checks the value of the P array in the position of the magnet. Variable C indicates which pole the magnet is over. The loop will continue as long as the values in this array are 0. If the loop finishes, there are no disks on this pole and the computer returns to the main program.

Line 560 places the value of the disk into variable S. The value of P is set to the position of this disk in A$. The disk is transferred to P$. The row this disk is in is the value of X plus 13. The disk is removed from the array by setting its position in the array to 0.

Line 570 moves the disk up the pole up to the magnet. The FOR-NEXT loop begins with the row the disk is currently in and ends just below the magnet. The disk is erased, then printed one row above. This loop continues until the disk is just below the magnet. The computer makes a sound.

Line 580 returns the computer to the main program.

Line 600 drops the disk from the magnet. The computer checks the top disk on the pole. If this disk has a value less than the value of the disk under the magnet, the disk on the pole is smaller and the computer will not drop the disk on the pole. The computer returns to the main program.

Line 610 checks to see if there are any disks on this pole. If there are, the value of P is one less than the number of disks on the pole. This places the disk just above the top disk. The computer is sent to line 630 to drop the disk.

Line 620 continues the loop. The value of P is set to one less than the value of X if the entire pole is empty.

Line 630 drops the disk onto the pole. The computer begins with the value of R and continues until it is at the location of P plus R plus 6. The disk is erased, then printed one row lower. The loop continues until the disk is resting on the platform or another disk. The value of M is increased by one every time a disk is released.

Line 640 places the value of the disk into the P array and returns to the main program.

Chapter 5

Simulations

The computer is the ideal tool to simulate a situation that may be too dangerous or not available at a particular time or location. The games in this chapter are traditional games that require some equipment. Your computer will display the items needed on the screen. You control them with the keyboard or joystick. Most of these games require good hand-eye coordination when played with the proper equipment; the same coordination is needed for the computer version.

DUCK DARTS

Objective of the game: To eliminate all the ducks on the screen.

Description: This game is similar to the skill game hawked at every local carnival: hit the duck and win a prize.

In the computer version, you have a choice of three levels of play. Each level places an additional row of ducks on the screen. To select the level of play, press the Select key. Each time the Select key is pressed another row of ducks is printed on the screen. When the correct level is displayed on the screen, press the Start key and the game begins. Use the joystick to move the dart back and forth across the screen. Press the fire button to release the dart. You have three darts. Every duck you hit is worth 10 points. The blue ducks are worth 50 points. The game continues until a score of 10,000 is reached or all the darts are used. To play again, press

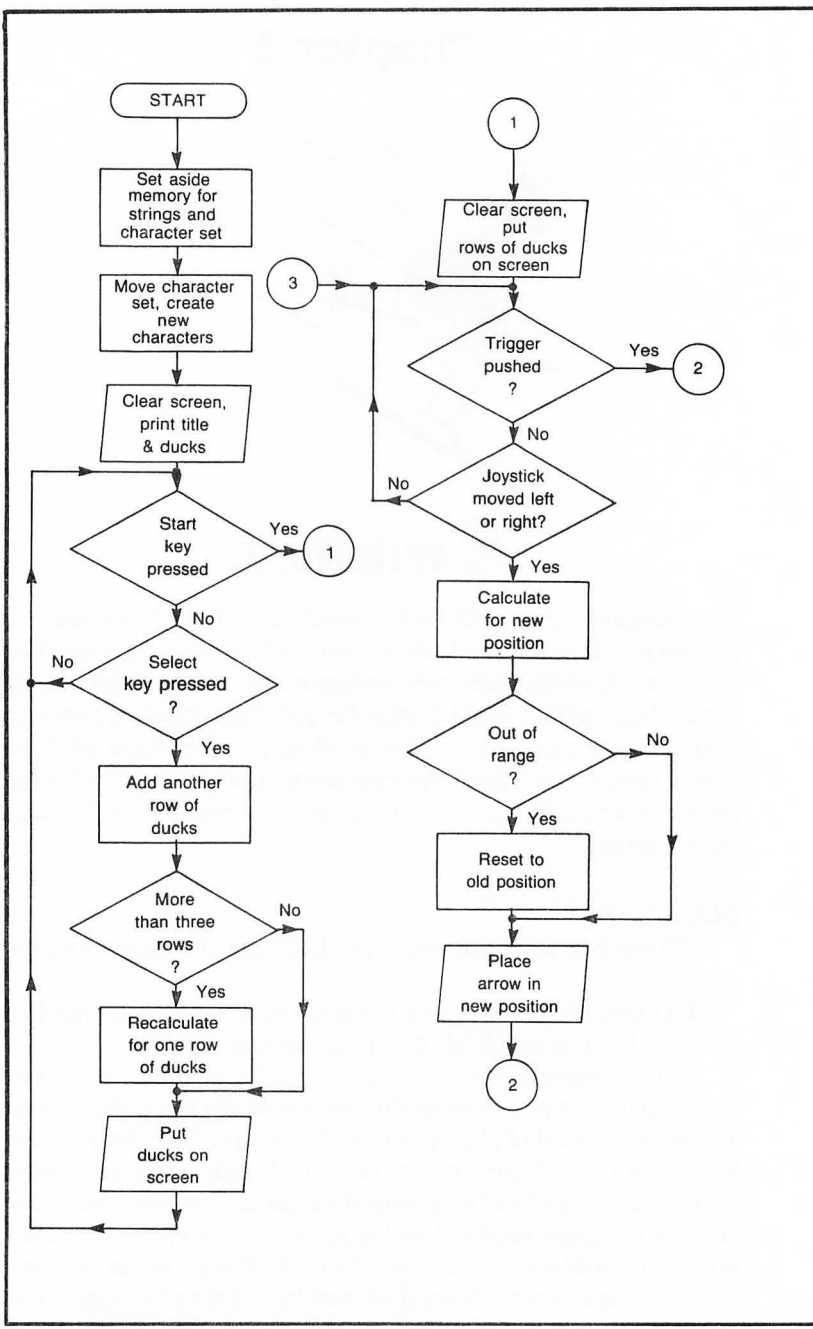
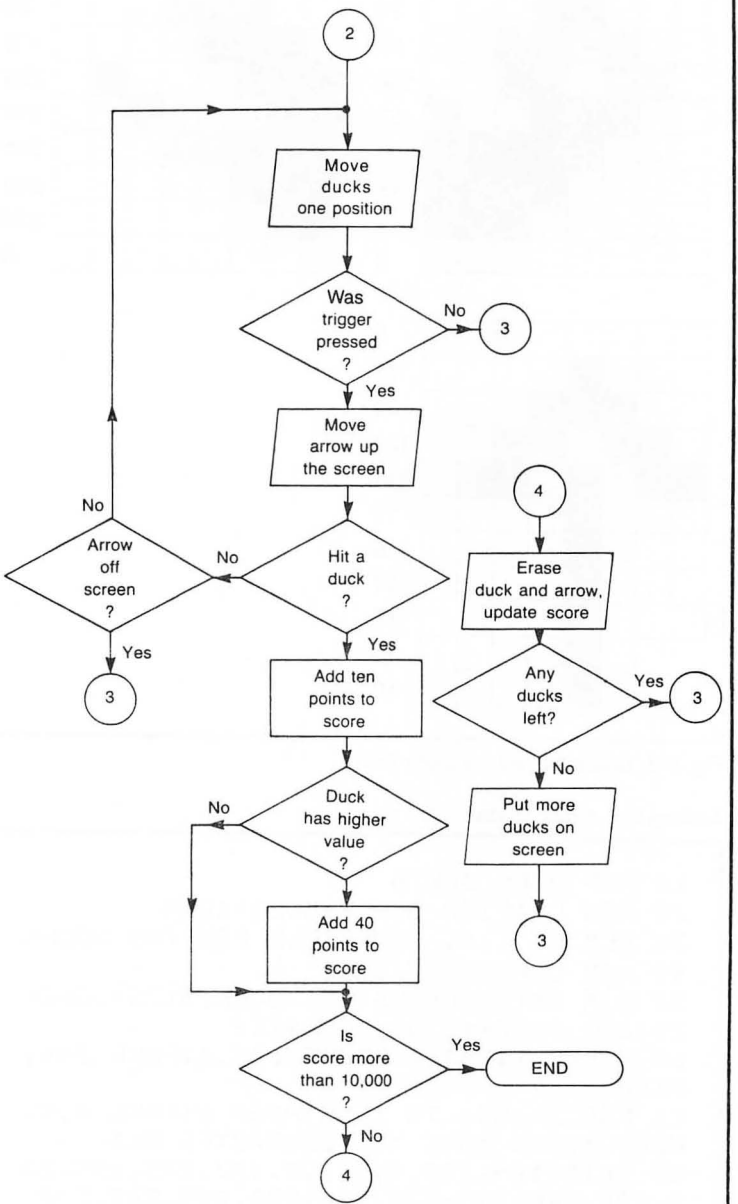


Fig. 5-1. Flowchart for Duck Darts.


2
Move ducks one position
Was trigger pressed ?
No
3
Yes
Move arrow up the screen
No
Arrow off screen ?
No
Hit a duck ?
Yes
3
Yes
Add ten points to score
Duck has higher value ?
No
Add 40 points to score
Is score more than 10,000 ?
Yes
END
No
4
4
Erase duck and arrow, update score
Any ducks left?
Yes
3
No
Put more ducks on screen
3

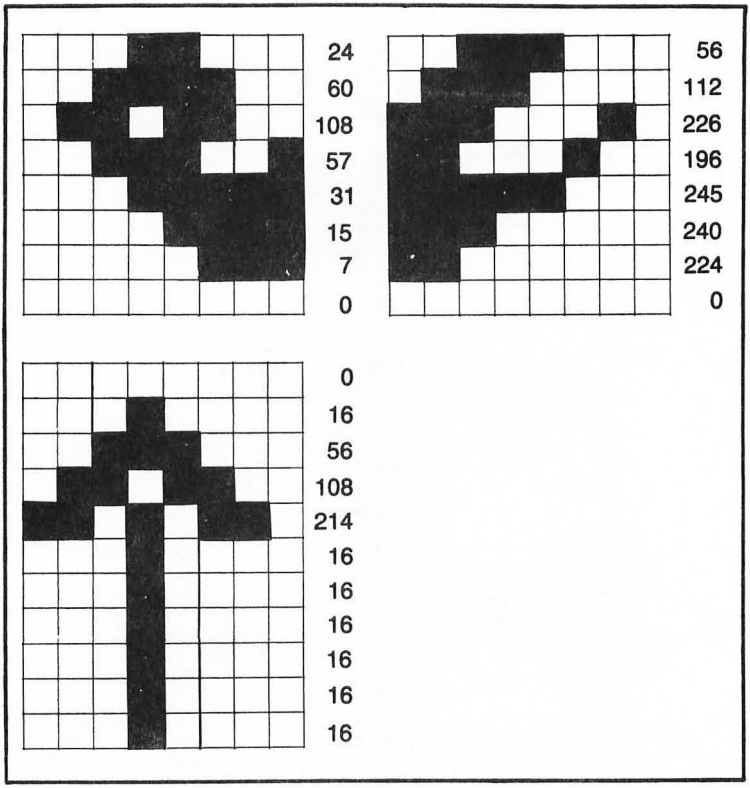


Fig. 5-2. Character set for Duck Darts.

Listing 5-1. Duck Darts.

```
10 REM DUCK DARTS
20 REM CHAPTER 5 - SIMULATIONS
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM A$(1),DUCK$(25),DUCK1$(25),DUCK
2$(25),ARROW$(20),TEMP$(1)
60 A=PEEK(106)-16:POKE 204,A:POKE 206,
PEEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 18:Q=USR(1536):POKE 756,A:
CHARSET=A*256+24:FOR X=CHARSET TO CHAR
```

```
SET+15:READ V:POKE X,V:NEXT X
95 FOR X=1 TO 20:READ V:ARROW$(X,X)=CH
R$(V):NEXT X
100 POKE 54279,A:POKE 559,62:POKE 5327
7,3:POKE 53248,0:VT=PEEK(134)+PEEK(135
)*256:STTB=PEEK(140)+PEEK(141)*256
110 STROFF=A*256+1024-STTB:ST3=INT(STR
OFF/256):ST4=STROFF-256*ST3:POKE VT+2,
ST4:POKE VT+3,ST3:POKE VT+4,255
120 POKE VT+5,0:POKE VT+6,255:POKE VT+
7,0:A$(1)=CHR$(0):A$(255)=CHR$(0):A$(2
)=A$
130 DATA 24,60,108,57,31,15,7,0,56,112
,226,196,248,240,224,0,0,16,56,108,214
,16,16,16,16,16,16,0,0,0,0,0,0,0,0,0
140 ? #6;CHR$(125):POSITION 5,5:? #6;"
DUCK":POSITION 11,7:? #6;"DARTS":REM D
ARTS IS INVERSE
150 POSITION 3,7:? #6;"##":POSITION 14
,10:? #6;CHR$(3);CHR$(4):POSITION 5,11
:? #6;CHR$(163);CHR$(164)
160 POSITION 11,6:? #6;CHR$(131);CHR$(
132)
170 AL=3:SC=0:FOR X=1 TO 24 STEP 3:C=I
NT(RND(0)*4)+1:ON C GOTO 180,190,200,2
10
180 DUCK$(X,X+2)="## ":GOTO 220
190 DUCK$(X)=CHR$(3):DUCK$(X+1)=CHR$(4
):DUCK$(X+2)=" ":GOTO 220
200 DUCK$(X)=CHR$(163):DUCK$(X+1)=CHR$
(164):DUCK$(X+2)=" ":GOTO 220
210 DUCK$(X)=CHR$(131):DUCK$(X+1)=CHR$
(132):DUCK$(X+2)=" "
220 NEXT X:X=0
230 POSITION 0,X:? #6;DUCK$(X*2+1,X*2+
20):POSITION 0,X+1:? #6;A$(1,40)
240 IF PEEK(53279)=7 THEN 240
250 IF PEEK(53279)=6 THEN 310
260 IF PEEK(53279)<>5 THEN 240
270 X=X+1:IF X>2 THEN X=0
280 FOR R=1 TO 100:NEXT R
290 GOTO 230
300 REM SET UP THE DUCKS
310 FOR R=1 TO 24 STEP 3:C=INT(RND(0)*
4)+1:ON C GOTO 320,330,340,350
320 DUCK1$(R,R+2)="## ":GOTO 360
330 DUCK1$(R)=CHR$(3):DUCK1$(R+1)=CHR$
```

```
(4):DUCK1$(R+2)=" ":GOTO 360
340 DUCK1$(R)=CHR$(163):DUCK1$(R+1)=CH
R$(164):DUCK1$(R+2)=" ":GOTO 360
350 DUCK1$(R)=CHR$(131):DUCK1$(R+1)=CH
R$(132):DUCK1$(R+2)=" "
360 NEXT R
370 FOR R=1 TO 24 STEP 3:C=INT(RND(0)*
4)+1:ON C GOTO 380,390,400,410
380 DUCK2$(R,R+2)="$$ ":GOTO 420
390 DUCK2$(R)=CHR$(3):DUCK2$(R+1)=CHR$
(4):DUCK2$(R+2)=" ":GOTO 420
400 DUCK2$(R)=CHR$(163):DUCK2$(R+1)=CH
R$(164):DUCK2$(R+2)=" ":GOTO 420
410 DUCK2$(R)=CHR$(131):DUCK2$(R+1)=CH
R$(132):DUCK2$(R+2)=" "
420 NEXT R
430 ? #6;CHR$(125):POSITION 1,0:? #6;D
UCK$(1,18):IF X=0 THEN 460
440 POSITION 1,2:? #6;DUCK1$(2,19):IF
X=1 THEN 460
450 POSITION 1,4:? #6;DUCK2$(3,20)
460 A=196:AP=121:POKE 704,60:A$(A)=ARR
OW$:POKE 53248,AP:POSITION 10,11:? #6;
SC;:TR=0:POSITION 0,11:? #6;AL-1
470 POKE 77,0:IF STRIG(0)=0 THEN TR=1:
GOTO 510
480 IF STICK(0)=7 THEN AP=AP+8:IF AP>2
01 THEN AP=201
490 IF STICK(0)=11 THEN AP=AP-8:IF AP<
49 THEN AP=49
500 POKE 53248,AP
510 TEMP$=DUCK$(1,1):DUCK$(1,23)=DUCK$
(2,24):DUCK$(24)=TEMP$:POSITION 1,0:?
#6;DUCK$(1,18)
520 TEMP$=DUCK1$(1,1):DUCK1$(1,23)=DUC
K1$(2,24):DUCK1$(24)=TEMP$:IF X<>0 THE
N POSITION 1,2:? #6;DUCK1$(2,19)
530 TEMP$=DUCK2$(1,1):DUCK2$(1,23)=DUC
K2$(2,24):DUCK2$(24)=TEMP$:IF X=2 THEN
 POSITION 1,4:? #6;DUCK2$(3,20)
540 IF  NOT TR THEN 470
550 POKE 53278,0:A$(A-1,A+18)=ARROW$:A
=A-4:IF A<20 THEN AL=AL-1:GOSUB 900:IF
 AL<1 THEN 800
555 IF A<20 THEN 630
560 IF PEEK(53252)=0 THEN 510
```

```
570 GOSUB 910:POKE 53248,0:R=0:IF A>47
 THEN R=2:IF A>79 THEN R=4
580 C=(AP-49)/8:LOCATE C,R,Q:POSITION
C-1,R:? #6;"   ":REM THREE SPACES WIPE
S OUT THE DUCK
590 SC=SC+10:IF Q<5 THEN SC=SC+40:REM
ADD TEN POINTS FOR A HIT - 50 FOR A BL
UE DUCK
595 IF SC>10000 THEN 800:REM END GAME
AT TEN THOUSAND POINTS
600 IF R=0 THEN DUCK$(C-1,C+1)="   "
610 IF R=2 THEN DUCK1$(C,C+2)="   "
620 IF R=4 THEN DUCK2$(C+1,C+3)="   "
630 A$(1)=CHR$(0):A$(255)=CHR$(0):A$(2
)=A$
640 ON X+1 GOTO 650,670,700
650 IF DUCK$<>"
 " THEN 460:REM 24 SPACES
660 GOTO 730
670 IF DUCK$<>"
 " THEN 460:REM 24 SPACES
680 IF DUCK1$<>"
  " THEN 460:REM 24 SPACES
690 GOTO 730
700 IF DUCK$<>"
 " THEN 460:REM 24 SPACES
710 IF DUCK1$<>"
  " THEN 460:REM 24 SPACES
720 IF DUCK2$<>"
  " THEN 460:REM 24 SPACES
730 FOR R=1 TO 24 STEP 3:C=INT(RND(0)*
4)+1:ON C GOTO 740,750,760,770
740 DUCK$(R,R+2)="$$ ":GOTO 780
750 DUCK$(R)=CHR$(3):DUCK$(R+1)=CHR$(4
):DUCK$(R+2)=" ":GOTO 780
760 DUCK$(R)=CHR$(163):DUCK$(R+1)=CHR$
(164):DUCK$(R+2)=" ":GOTO 780
770 DUCK$(R)=CHR$(131):DUCK$(R+1)=CHR$
(132):DUCK$(R+2)=" "
780 NEXT R:GOTO 310
800 POKE 53248,0:POSITION 5,0:? #6;"GA
ME OVER":A$(1)=CHR$(0):A$(255)=CHR$(0)
:A$(2)=A$
810 IF PEEK(53279)<>6 THEN 810
820 GOTO 140
900 SOUND 0,10,6,10:FOR XX=1 TO 50:NEX
```

```
T XX:SOUND 0,0,0,0:RETURN
910 SOUND 0,10,4,10:FOR XX=1 TO 10:NEX
T XX:SOUND 0,0,0,0:RETURN
```

the Start key. Figure 5-1 is the flowchart, and Fig. 5-2 the character set, for this program. See Listing 5-1 for the code.

Line 50 sets aside the string space needed in this program. The first string in this line is A$. For this program to work correctly, this string *must* be listed first in the line. We will change the length of this string and use it as a player/missile, so we have to know where it is in the variable table. The next three strings contain the ducks that move on the screen. ARROW$ contains the characters that make up the dart, and TEMP$ is used to hold the first character of the string when the ducks are moved on the screen.

Line 60 looks at location 106 to find out how much memory is in the computer. This program will use part of the memory in the computer for the character set and for player/missile graphics. The beginning of the new character set is placed in location 204. The beginning of the old character set is placed in location 206. The assembly language subroutine in line 90 uses these two locations to move the character set from ROM into RAM.

Line 70 reads the assembly language subroutine from line 80 and places it in memory locations 1536-1555. Line 80 is the decimal code for the routine that moves the character set from ROM into RAM.

Line 90 changes the screen to a graphics 18—graphics 2 with no text window. The USR command tells the computer to use the subroutine beginning at location 1536. The computer returns to this line after executing the assembly language subroutine. The value of A is POKEd into location 756. The A variable contains the beginning location of the new character set. If the wrong value is POKEd into this location, the screen will have strange characters on it.

The CHARSET variable is set to the first byte of the character set, plus 24. We add 24 to the first byte because the first character in the character set is a space. The second is the exclamation point, and the third the quotation marks. We do not want to change the character for the quotation marks, so we begin with the third character or 24th byte—the pound sign. The FOR-NEXT loop begins with the 24th byte of the character set. The character codes in line 130 are read and placed into the character set. Two characters, the pound sign and the dollar sign, are changed. The pound sign is the front of the duck and the dollar sign is the back.

Line 95 reads the next 20 values and places them into ARROW$. This time the values are not placed in the string, but the character the value represents is. This string will be moved into the area set aside for player/graphics. Each byte is one code. The code is very conveniently stored within this string as a series of characters.

Line 100 POKEs location 54279 with the value of A. This tells the computer where the player/missile area is. By POKEing 559 with 62, we tell the computer that the players are single-resolution. When location 53277 is POKEd with 3, the player/missile graphics are enabled. The VT variable contains the beginning address of the variable value table. This table keeps track of all the variables used in a program. The variables are stored in this table in the order they are entered, which is why A$ must be the first variable in line 50, and no other variables are placed on any program line before line 50. The STTB variable contains the beginning address of the string array area. This area stores the information in the strings. Each element of a string takes up one memory location in this area.

Line 110 determines the offset for A$. We want A$ to be the first player in the player/missile graphics area. The computer designated one byte for A$, the first byte of the string array area. To change the location of this string in memory, we have to figure out where we want the string to begin. The first player in the player/missile graphics area begins 1K after the address POKEd into location 54279. To arrive at the correct offset, we multiply the value of A by 256 and add 1024 (1K). This is where we want the first byte of A$ to be.

If we placed this value in the offset location in the variable value table, we would *not* tell the computer where A$ should begin, because the computer adds the offset value to the value of locations 140 and 141. We subtract the value of STTB, which contains the value of locations 140 and 141. When the computer adds this value to the offset value in the variable value table, it places the information in A$ in the player/missile graphics area.

The value of STROFF is larger than 255 and cannot be POKEd into the variable value table as one number. To divide it into two numbers the computer can understand, take the integer of STROFF and store it in variable ST3. This number is referred to as the *high-order byte*. By multiplying this number by 256 and subtracting the product from the offset value, we arrive at the *low-order byte*. These two values are POKEd into the third and fourth locations of variable

value table. The value 255 is POKEd into the fifth location, because we want the computer to think that string is 255 bytes long.

Line 120 POKEs a 0 into the sixth location of the variable value table. This is the high-order byte of the length of A$. The seventh and eighth locations are POKEd with a 255 and 0. The number of bytes set aside for A$ is 255. By setting the first and last elements of A$ to the character 0, then setting the second element to the contents of A$, we clear the entire string. If A$ were printed, it would be a string of hearts, but the ATASCII value of that character is 0. This value is what will be displayed as a player on the screen.

Line 130 contains the codes to create the duck and the dart.

Lines 140-160 print the title of the program and four ducks, each in a different color, on the screen.

Lines 170-220 place a string of ducks in the first string, DUCK$. A random number from 1 to 4 is chosen. This number determines the color of the duck. The computer is sent to the correct line. The two parts of the duck, followed by one space, are placed into DUCK$. The loop continues until the entire string is filled. Variable X is set to 0. It is used in the next routine as the row number.

Lines 230-290 set the level of play. One row of ducks is printed on the screen. The first 40 characters of A$ is printed on the next line. Since A$ is hearts, the computer will print two rows of spaces on the screen. This will erase any ducks from the screen should the Select key be pressed three or more times. The computer loops at line 240 until a Start, Option, or Select key is pressed. If the Start key is pressed, the computer will go to line 310 and begin the game. If the Select key is not pressed, the computer will loop back to line 240. When the Select is pressed variable X is incremented by 1. Should the value of X exceed 2, it will be reset to 0. A short timing loop gives the user a chance to release the key, then the computer goes to line 230 to print one more row of ducks on the screen.

Lines 310-420 place ducks in the other two DUCK strings. The color of the ducks in both these strings are chosen randomly. The strings are filled even if level one is chosen, so that the computer will rotate the ducks in these strings and keep the ducks on the screen moving at a steady pace.

Line 430 clears the screen and places the first row of ducks on the screen. Variable X contains the level number. If X is a 0, then level one was chosen and the computer is directed to line 460.

Line 440 prints the ducks in DUCK1$ on the screen if levels two or three were chosen.

Line 450 prints the ducks in DUCK2$ on the screen if level three was chosen.

Line 460 sets variable A to 196. This is the position in A$ where the tip of the dart begins. Variable AP is set to 121. This is the column for the dart. The characters in ARROW$ are transferred to A$. Each character is the code for part of the dart. The dart is printed on the screen by moving the player onto the screen. Location 53248 is POKEd with the column number for the dart. The score, stored in variable SC, is printed on the screen; variable TR (trigger) is set to 0, and the number of darts left (AL) is printed in the lower left corner of the screen.

Line 470 disengages the ATTRACT mode by POKEing location 77 with a zero. If the red button on the joystick is pressed, variable TR is set to 1 and the computer is directed to line 510.

Line 480 checks to see if the joystick is being pushed to the right. If it is, variable AP is increased by 8. The value of AP is also checked to make sure it is under 201. If it is, it is reset to 201. This is the rightmost column on the screen. A higher value would put the dart off the screen.

Line 490 checks to see if the joystick is being pushed to the left. If it is, the value of AP is decreased by 8. The value of AP is checked again to make sure the dart will be printed on the screen.

Line 500 moves the dart to the new column on the screen. The dart moves eight columns at a time. If it moved only one column at a time it would take too much time to get across the screen.

Line 510 rotates the top row ducks on the screen. The contents of the first element of DUCK$ is placed into TEMP$. Then all the elements of DUCK$ are moved up by 1. The character placed in DUCK$ is placed in the last element of the string. The first 18 elements of the string are printed on the screen.

Line 520 rotates the second row of ducks (DUCK1$) the same way. Even if these ducks are not printed on the screen, they are still rotated.

Line 530 rotates the third row of ducks, DUCK2$. If level three was chosen, these ducks will be printed on the screen. Line 540 checks the value of TR. If variable TR is not a one (NOT TR), the computer will be sent to line 470 to see if the trigger or joystick has been pressed or moved and to keep the ducks moving on the screen.

Line 550 is executed when the value of TR is 0. Location 53278 is POKEd with 0. This clears all the registers used to count hits. These registers contain a value when a player or missile occupies

the same position on the screen as another player, missile, or character. The characters of ARROW$ are placed into A$ at a location one less than the last. This will move the dart up on the screen. When the value of A is less than 20, the dart is no longer on the screen and the computer is sent to the sound subroutine at line 900. The number of darts is decreased by 1. If there are no darts left, the computer is sent to line 800 to end the game.

Line 555 sends the computer to line 630 if the dart is off the screen.

Line 560 checks the contents of memory location 53252. If it is 0, the dart has not hit a duck and the computer is sent to line 510 to move the ducks and the dart.

Line 570 is executed when location 53252 contains a value other than 0. The hit sound subroutine at line 910 is used. Then the dart is removed from the screen by POKEing location 53248 with 0. Variable R is set to 0. This variable is for the row number of the duck that has been hit. If the value of A is greater than 47 the duck cannot be in the zeroth row. The value of R is changed to 2. If the value of A is greater than 79, the duck must be in the third row, and the value of R is reset to 4.

Line 580 calculates the column the duck is in. The 49th column for the player or dart is the equivalent of the zeroth column for a character. There are eight player columns to every character column. By subtracting 49, then dividing by 8, we know which column the dart is in. The LOCATE command sets the value of the character in the column and row that has been hit. Three spaces are printed at this location. The first space is printed just before the column that has a hit. This way, if the end of the duck was hit, the entire duck is erased from the screen.

Line 590 adds ten to the score (SC). If the value of Q is less than 5, the duck was blue and 40 more points are added to the score.

Line 600 checks the value of SC to see if the player has gone over 10,000 points. If he has, the computer is sent to line 800 to end the game.

Lines 600-620 erase the duck from the string. The value of R indicates from which string the duck should be erased.

Line 630 clears the A$ again. If A$ was not cleared, the dart would reappear near the top of the screen along with the new dart near the bottom.

Lines 640-720 check to see if all the ducks have been hit.

Line 640 uses the level number to send the computer to the line that checks all the strings used at that level. If there are ducks

left, the computer is sent to line 460 to continue the game. If all the ducks have been hit, the computer continues with line 730.

Lines 730-780 place ducks into DUCK$. After the first string is filled, the computer is sent to line 310 where the other strings are filled if the game is at level two or three.

Lines 800-820 end the game. The dart is cleared from the screen, the message "GAME OVER" is printed on the screen, and A$ is cleared for another game. The computer loops at line 810 until the Start key is pressed. The new game begins at line 140.

Lines 900-910 are the sound subroutines used in this program.

MARBLES

Objective of the game: To hit your opponent's marbles.

Directions: This is a two-player game that requires two joysticks. One joystick is plugged into the first joystick port, the other into the second joystick port. The first player's marble is printed on the left side of the screen. The second player's marble is printed on the right side. To move the marble, move the joystick up, down, left, or right. To shoot the marble over the line, press the fire button on the joystick. The longer the fire button is held down, the further the marble will go. However, if you hold the button down too long, the marble will hardly move. To get the marble across the field takes good timing.

The first player shoots a marble, trying to keep it on the left side of the field. The second player then tries to hit the opponent's marble. The game continues until one player scores 10 or more points. Every time the opponent's marble is hit, it is erased and two points are added to the score of the player who shot the marble. If a player hits his own marble, one point is subtracted from the score. The score cannot go below 0. Figure 5-3 is the flowchart, Fig. 5-4 the character set, and Listing 5-2 is the code for this program.

Line 50 sets aside the string area. M$ must be the first variable in the program. This string is used as the player represented by the marble. If it is not the first variable used, the routine that changes the string storage will not work properly. WORD$ holds the word printed on the screen. This program uses ANTIC mode E. Letters and characters are drawn on the screen in this mode rather than printed, so the word or character to appear on the screen is stored in WORD$ before it is placed on the screen. BLBYTE$, GRBYTE$, and ORBYTE$ contain the byte patterns for printing the words or characters in blue, green, or orange. M1$ and M2$

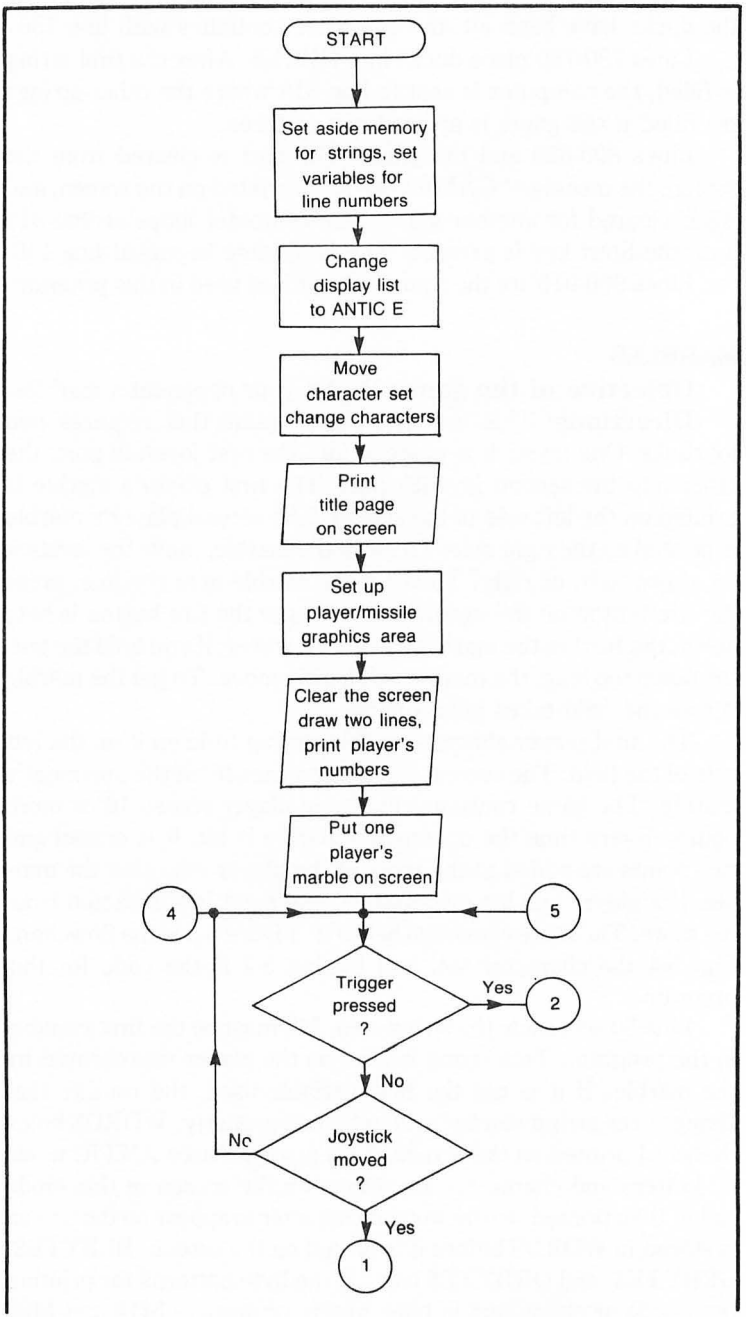
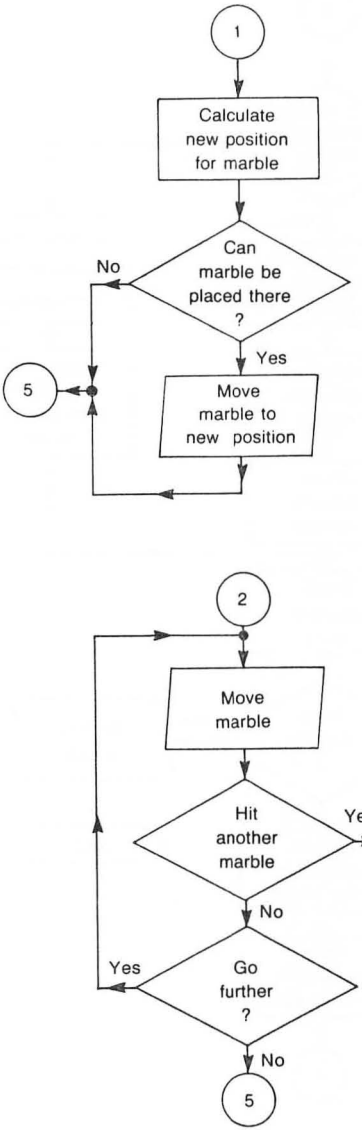


Fig. 5-3. Flowchart for Marbles. (Continued through page 236.)


1
Calculate new position for marble
Can marble be placed there ?
No
Yes
5
Move marble to new position
2
Move marble
Hit another marble
Yes
3
No
Yes
Go further ?
No
5

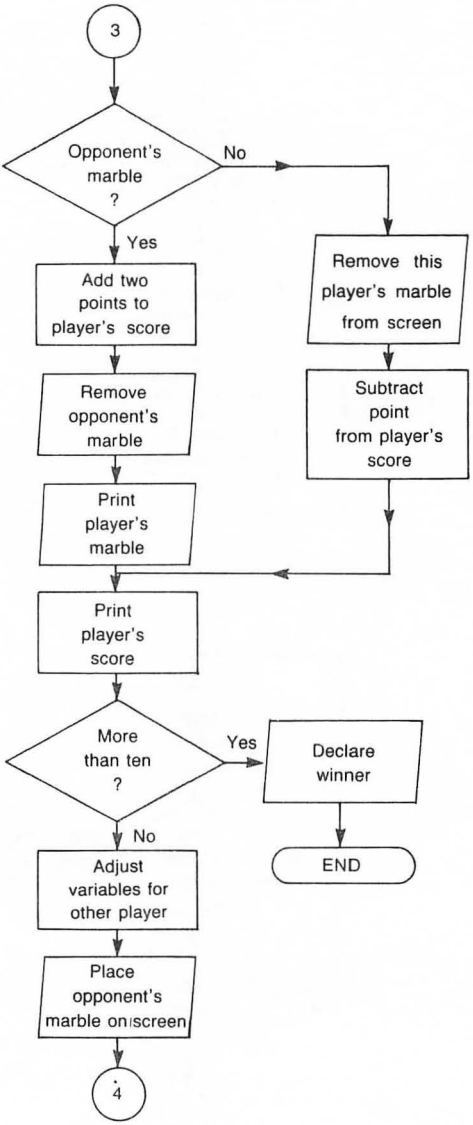


Fig. 5-3. Continued.

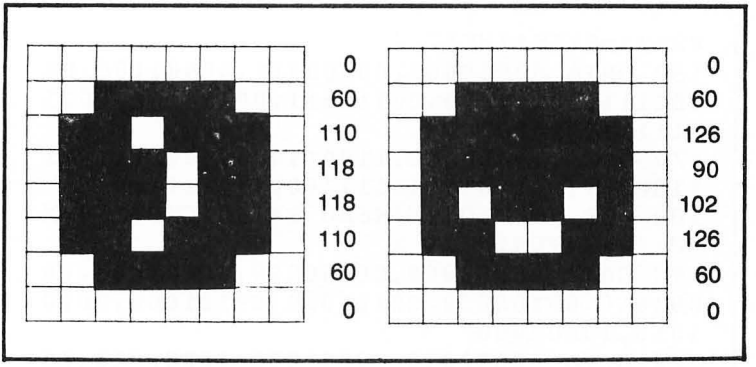


Fig. 5-4. Character set for Marbles.

Listing 5-2. Marbles.

```
10 REM MARBLES - A YOUNG BOY'S GAME
20 REM CHAPTER 5 - SIMULATIONS
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM M$(1),WORD$(12),BYTE$(16),BLBYT
E$(128),GRBYTE$(128),ORBYTE$(128),M1$(
8),M2$(8),B$(8)
60 SCRNPRNT=1000:GRAPHICS 24:REM GRAPH
ICS 8 WITH NO TEXT WINDOW
70 DLIST=PEEK(560)+PEEK(561)*256:POKE
DLIST+3,78:REM CHANGE TO ANTIC E
80 FOR X=DLIST TO DLIST+200:IF PEEK(X)
=15 THEN POKE X,14:REM CHANGE TO ANTIC
 E
90 IF PEEK(X)=79 THEN POKE X,78:REM CH
ANGE THE SECOND LIST TOO
100 NEXT X
110 A=PEEK(106)-40:POKE 204,A:POKE 206
,PEEK(756):FOR X=1536 TO 1555:READ V:P
OKE X,V:NEXT X
120 DATA 104,162,4,160,0,177,205,145,2
03,200,208,249,230,206,230,204,202,208
,242,96
130 Q=USR(1536):POKE 756,A:CHARBASE=A*
256+24:FOR X=CHARBASE TO CHARBASE+15:R
EAD V:POKE X,V:NEXT X
140 DATA 0,60,110,118,118,110,60,0,0,6
0,126,90,102,126,60,0
150 FOR X=0 TO 15:READ BYTE$:BLBYTE$(X
```

```
*8+1)=BYTE$:NEXT X
160 DATA 00000000,00000011,00001100,00
001111,00110000,00110011,00111100,0011
1111,11000000
170 DATA 11000011,11001100,11001111,11
110000,11110011,11111100,11111111
180 FOR X=0 TO 15:READ BYTE$:GRBYTE$(X
*8+1)=BYTE$:NEXT X
190 DATA 00000000,00000010,00001000,00
001010,00100000,00100010,00101000,0010
1010,10000000
200 DATA 10000010,10001000,10001010,10
100000,10100010,10101000,10101010
210 FOR X=0 TO 15:READ BYTE$:ORBYTE$(X
*8+1)=BYTE$:NEXT X:BYTE$="000000000000
0000"
220 DATA 00000000,00000001,00000100,00
000101,00010000,00010001,00010100,0001
0101,01000000
230 DATA 01000001,01000100,01000101,01
010000,01010001,01010100,01010101
240 H=1:WORD$="$":R=50:C=64:CL=2:GOSUB
 SCRNPRNT
250 H=4:WORD$="MARBLES":R=50:C=86:CL=1
:GOSUB SCRNPRNT
260 H=1:WORD$="$":R=60:C=200:CL=3:GOSU
B SCRNPRNT
270 POKE 54279,A:POKE 559,62:POKE 5327
7,3:POKE 53248,0:VT=PEEK(134)+PEEK(135
)*256:STTB=PEEK(140)+PEEK(141)*256
280 STROFF=A*256+1024-STTB:ST3=INT(STR
OFF/256):ST4=STROFF-256*ST3:POKE VT+2,
ST4:POKE VT+3,ST3:POKE VT+4,0
290 POKE VT+5,1:POKE VT+6,0:POKE VT+7,
1:M$(1)=CHR$(0):M$(255)=CHR$(0):M$(2)=
M$:B$=M$
300 RESTORE 140:FOR X=1 TO 8:READ V:M1
$(X,X)=CHR$(V):NEXT X:FOR X=1 TO 8:REA
D V:M2$(X,X)=CHR$(V):NEXT X
310 POKE 704,66:FOR X=1 TO 10:R=INT(RN
D(1)*183)+1:C=INT(RND(1)*150)*2+2
320 M$(R+33,R+40)=M1$:V=INT(RND(1)*2):
IF V THEN M$(R+33,R+40)=M2$
330 CL=2:WORD$="$":POKE 704,PEEK(709):
IF V THEN WORD$="$":CL=3:POKE 704,PEEK
(708)
```

```
340 POKE 53248,C/2+48:GOSUB 3000:GOSUB
 SCRNPRNT:POKE 53248,0:M$(R+33,R+40)=B
$:NEXT X
360 ? #6;CHR$(125):COLOR 1:PLOT 70,0:D
RAWTO 70,191:PLOT 71,0:DRAWTO 71,191:P
LOT 242,0:DRAWTO 242,191
365 PLOT 243,0:DRAWTO 243,191
370 PL1=0:PL2=0:H=1:WORD$="one":R=0:C=
0:CL=2:GOSUB SCRNPRNT:WORD$="two":R=0:
C=262:CL=3:GOSUB SCRNPRNT
380 P=120:P1=65:POKE 704,PEEK(709):POK
E 53248,P1:M$(P,P+7)=M1$:H=1:SP=0:C1=0
:R1=0
390 POKE 77,0:IF STRIG(0)=0 THEN 470
400 IF STICK(0)=15 THEN 390
410 IF STICK(0)=14 THEN P=P-1:IF P<33
THEN P=33
420 IF STICK(0)=13 THEN P=P+1:IF P>217
 THEN P=217
430 IF STICK(0)=7 THEN P1=P1+1:IF P1>7
5 THEN P1=75
440 IF STICK(0)=11 THEN P1=P1-1:IF P1<
40 THEN P1=40
450 POKE 53278,0:M$(P,P+7)=M1$:POKE 53
248,P1:IF PEEK(53252)=0 THEN 390
460 P1=P1-2:POKE 53248,P1:GOTO 390
470 POKE 20,0
480 IF STRIG(0)=0 THEN 480
490 R=PEEK(20)*2:IF R>150 THEN R=150-(
R-150)
500 IF R<90 THEN R=90
510 ST=STICK(0)
520 POKE 53278,0:P1=P1+1:IF ST=13 THEN
 P=P+1:IF P>217 THEN P=217
530 IF ST=14 THEN P=P-1:IF P<33 THEN P
=33
540 M$(P,P+7)=M1$:POKE 53248,P1:IF P1<
R AND PEEK(53252)=0 THEN 520
550 Q=PEEK(53252):IF Q=0 THEN 620
560 IF Q=4 THEN 520
565 IF Q<>1 THEN PL1=PL1-1:POKE 53248,
0:GOTO 630
570 PL1=PL1+2:C=(P1-48)*2+13:R=P-33:SP
=0
580 TRAP 586:LOCATE C,R,Q1:IF Q1=1 THE
N R1=R:R=R-1:GOTO 580
```

```
581 IF R1=0 AND SP=0 AND Q1=0 THEN R=R
+1:GOTO 580
585 SP=SP+1:R=R-1:IF SP<3 THEN 580
586 SP=0:TRAP 40000
590 LOCATE C,R1+1,Q1:IF Q1=1 THEN C1=C
:C=C-2:GOTO 590
600 SP=SP+1:C=C-2:IF SP<3 THEN 590
610 C=C1:R=R1-1:GOSUB 3000:WORD$=" ":G
OSUB SCRNPRNT:REM TAKE AWAY ONE MARBLE
 - THEN PRINT THE OTHER
620 CL=2:WORD$="#":C=(P1-48)*2:R=P-33:
GOSUB SCRNPRNT
630 C=14:R=9:IF PL1<0 THEN PL1=0
635 WORD$=STR$(PL1):GOSUB SCRNPRNT:IF
PL1>9 THEN 950
640 M$(P,P+7)=B$:P=120:P1=175:POKE 704
,PEEK(708):POKE 53248,P1:M$(P,P+7)=M2$
:SP=0:C1=0:R1=0
650 POKE 77,0:IF STRIG(0)=0 THEN 730
660 IF STICK(0)=15 THEN 650
670 IF STICK(0)=14 THEN P=P-1:IF P<33
THEN P=33
680 IF STICK(0)=13 THEN P=P+1:IF P>217
 THEN P=217
690 IF STICK(0)=7 THEN P1=P1+1:IF P1>2
03 THEN P1=203
700 IF STICK(0)=11 THEN P1=P1-1:IF P1<
170 THEN P1=170
710 POKE 53278,0:M$(P,P+7)=M2$:POKE 53
248,P1:IF PEEK(53252)=0 THEN 650
720 P1=P1+2:POKE 53248,P1:GOTO 650
730 POKE 20,0
740 IF STRIG(0)=0 THEN 740
750 R=180-PEEK(20):POKE 20,0:IF R<95 T
HEN R=95+(95-R)
760 IF R>160 THEN R=160
770 ST=STICK(0)
780 P1=P1-1:IF ST=13 THEN P=P+1:IF P>2
17 THEN P=217
790 IF ST=14 THEN P=P-1:IF P<33 THEN P
=33
800 POKE 53278,0:M$(P,P+7)=M2$:POKE 53
248,P1:IF P1>R AND PEEK(53252)=0 THEN
780
810 Q=PEEK(53252):IF Q=0 THEN 880
820 IF Q=4 THEN 780
```

```
825 IF Q<>2 THEN PL2=PL2-1:POKE 53248,
0:GOTO 890
830 PL2=PL2+2:C=(P1-48)*2:R=P-33
840 TRAP 846:LOCATE C,R,Q1:IF Q1=1 THE
N R1=R:R=R-1:GOTO 840
841 IF R1=0 AND SP=0 AND Q1=0 THEN R=R
+1:GOTO 840
845 SP=SP+1:R=R-1:IF SP<3 THEN 840
846 SP=0:TRAP 40000
850 LOCATE C,R1+1,Q1:IF Q1=1 THEN C1=C
:C=C-2:GOTO 850
860 SP=SP+1:C=C-2:IF SP<3 THEN 850
870 C=C1-2:R=R1:GOSUB 3000:WORD$=" ":G
OSUB SCRNPRNT:REM TAKE AWAY ONE MARBLE
 - THEN PRINT THE OTHER
880 CL=3:WORD$="$":C=(P1-48)*2:R=P-33:
GOSUB SCRNPRNT:IF PL2<0 THEN PL2=0
890 C=280:R=9:WORD$=STR$(PL2)
900 GOSUB SCRNPRNT:IF PL2<10 THEN M$(P
,P+7)=B$:GOTO 380
950 M$(P,P+7)=B$:? #6;CHR$(125):WORD$=
"Game Over":R=20:C=80:H=3:CL=1:GOSUB S
CRNPRNT
960 WORD$="Player 1 ":WORD$(10)=STR$(P
L1):H=2:CL=2:R=50:C=74:GOSUB SCRNPRNT
970 WORD$(8)="2":WORD$(10)=STR$(PL2):C
L=3:R=80:C=74:GOSUB SCRNPRNT
980 IF PEEK(53279)<>6 THEN 980
990 GOTO 360
1000 TH=H:TR=R:L=LEN(WORD$):CHBASE=A*2
56
1010 FOR WORD=1 TO L:LASC=ASC(WORD$(WO
RD,WORD)):IF LASC>31 AND LASC<96 THEN
LASC=LASC-32:GOTO 1030
1020 IF LASC<32 THEN LASC=LASC+64
1030 FOR LP=0 TO 7:BYTE=PEEK(CHBASE+LA
SC*8+LP):FBYTE=INT(BYTE/16):SBYTE=BYTE
-FBYTE*16:REM GET CODE FOR CHARACTER
1040 ON CL GOTO 1050,1060,1070
1050 BYTE$(1,8)=BLBYTE$(FBYTE*8+1,(FBY
TE+1)*8):BYTE$(9,16)=BLBYTE$(SBYTE*8+1
,(SBYTE+1)*8):GOTO 1080
1060 BYTE$(1,8)=GRBYTE$(FBYTE*8+1,(FBY
TE+1)*8):BYTE$(9,16)=GRBYTE$(SBYTE*8+1
,(SBYTE+1)*8):GOTO 1080
1070 BYTE$(1,8)=ORBYTE$(FBYTE*8+1,(FBY
```

```
TE+1)*8):BYTE$(9,16)=ORBYTE$(SBYTE*8+1
,(SBYTE+1)*8)
1080 POSITION C,R:? #6;BYTE$:TH=TH-1:R
=R+1:IF TH<>0 THEN 1080
1090 TH=H:NEXT LP:R=TR:C=C+16:NEXT WOR
D
1100 RETURN
3000 SOUND 0,100,2,15:FOR T=1 TO 5:NEX
T T:SOUND 0,0,0,0:RETURN
```

contain the code for the two different marbles. This code is transferred to the player area when the marble moves on the screen. B$ is used to erase the marble from the player area.

Line 60 sets variable SCRNPRNT to 1000. This is the line number for the routine that prints the words and characters on the screen. The HEX variable contains the beginning address of another routine that divides a number into two values. The screen is set to graphics 24. This is graphics 8 with no text window.

Line 70 finds the beginning address for the display list. This address is stored in locations 560 and 561. To arrive at the one number address multiply the number stored in location 561 by 256 and add the number stored in location 560. The entire display list is set for graphics 8. We want to use ANTIC E. This mode is between graphics modes 7 and 8. The first three numbers in the display list remain the same. The fourth number is changed to ANTIC E. This location is also an instruction, so the value POKEd here is 78 rather than 14.

Lines 80-90 change the rest of the codes in the display list to ANTIC E. Each location is checked to see if it contains the value 15, the value for graphics 8. If it does, 14 is POKEd into that location. If the value of 79 (graphics 8 with an instruction), it is changed to 78. The loop continues until the entire display list is changed to ANTIC E.

Line 110 finds out how much memory is in the computer. We subtract 40 from this value to set aside space for the new character set and the player/missile graphics. This program subtracts 40 instead of 16, as other programs do, because graphics 8 takes up 8K of memory. Every time we subtract 4 from the end of memory, we are really subtracting 1K. By subtracting 40, we allow room for the screen (8K) and the character set (2K). We POKE the beginning address of the character set into location 204 and the beginning of the ROM character into location 206. These two locations

are used in the assembly language subroutine that moves the character from ROM into RAM. The FOR-NEXT loop reads the code for the assembly language subroutine from line 120 and places it in memory locations 1536-1555. This is the assembly language subroutine that moves the character set from ROM into RAM. Line 120 contains the code for the assembly language subroutine.

Line 130 uses the USR command to tell the computer to execute the assembly language subroutine at memory location 1536. After the computer returns to this line, the beginning address of the moved character set is POKEd into location 756. The CHARBASE variable contains the address of the first byte of the third character in the character set. This character and the fourth character will be changed into marbles.

Line 140 contains the code to change the two characters.

Lines 150-230 set up the bit codes in strings to print the letters or characters in different colors. We use ANTIC E in this program. To print a letter on the screen we must turn on certain bits. The pattern these bits follow determine what color appears on the screen. In line 150, we fill the screen with the pattern for blue pixels. Whenever a pair of bits in a byte are set to 1, the color on the screen is blue. This must be an even pair—bits 1 and 2, 3 and 4, 5 and 6, 7 and 8. If bits 2 and 3 are set, one pixel would be orange and the other green.

Line 180 uses the information in lines 190 and 200 to place the pattern for green pixels into GRBYTE$. The pattern for orange pixels are placed into ORBYTE$.

Lines 240-260 place the title on the screen. Variable H determines the height of the letter or character. Since the subroutine places the character information on the screen byte by byte, the letters can be 8 pixels high or 20. The character or message printed on the screen is stored in WORD$. Variable R is set to the row on which the message will be printed, and variable C is set to the column. Variable CL determines the routine that will be used to print the message in the correct color. The value of SCRNPRNT is 1000. Lines 240-260 use the subroutine called SCRNPRNT to print the message on the screen.

Line 270 POKEs the value of A into location 54279. This tells the computer where the player/missile graphics begin. By POKEing location 559 with 62, we tell the computer that single-line resolution is used for the player. Location 53277 is POKEd with 3 to enable the player/missile graphics. The variable VT holds the address of the variable value table. This table contains information

on all the strings and variables used in the program. The computer stores the variables in the order they were entered in the program. This is why M$ must be the first string variable in the program. We want its information to occupy the first block of memory in this table. Variable STTB holds the address of the string storage area. This is the area of memory where the contents of strings and arrays are stored.

Line 280 calculates the offset address for M$. In the variable value table, a number is added to the value of the string storage area so the computer knows where the information for the string is stored. This value is called an *offset*. We want M$ to occupy 256 bytes of the player/missile graphics area instead of one byte in the string storage area. To change the location of the string, we need to change the offset for the string in the variable value table. The first player actually starts 1024 bytes after the value given the computer as the player/missile graphics area. We need to add this value to the beginning address and subtract the value of the string storage area.

The number remaining is the offset value. When the computer places information in the string it adds this value to the string storage value and arrives at the player/missile graphics area. The offset number must be stored as a two-digit number. The high-order value of the number is arrived at by taking the integer of the value divided by 255; the low-order value is the high-order value subtracted from the original value. These two values are stored in the third and fourth locations of the variable value table. The fifth location is POKEd with a 0. This is the low-order value for the length of the string.

Line 290 POKEs a one into the sixth location. Since this is the high-order byte, the one translates into 256 bytes. The next two bytes are set to 0, and 1 used to set aside 256 bytes of memory for this string. The entire string is cleared by setting the first and last elements of the string to 0, then setting the second element to the entire string. B$ is also cleared by setting it to M$, even though M$ is longer than B$.

Line 300 uses the RESTORE command to tell the computer to point to the information in line 140. This is the code for the marbles. The information for the first marble is stored in M1$. The information for the second marble is stored in M2$. This information will be moved into the player area when the marbles are printed on the screen.

Line 310 sets the color of the player by POKEing location 704

with 66. The FOR-NEXT loop counts from 1 to 10. The computer prints 10 marbles on the screen. Variables R and C contain random numbers. These numbers designate the row and column where the marble will be printed.

Line 320 places the first marble into M$. The row value (R) is offset by 33. The first 33 rows of a player do not appear on the screen. A random number is then chosen, which can be a 0 or a 1. If 1 is chosen, the second marble is moved into M$.

Line 330 sets CL to 2. The marble is green. The marble stored as the pound sign is used, and the value for the character at location 709 is POKEd into the player value. If variable V is 1, all these values change. The color variable (CL) is changed to 3. Now the marble is orange, the marble stored as the dollar sign is used, and the color of the player will be the same as the color stored at location 708.

Line 340 moves the player onto the screen. The value of variable C determines the column where the marble is placed. However, just as the first 33 rows of a player do not appear on the screen, some of the column locations also are off the screen area. To place the marble on the screen, the value of C is divided by 2 and the offset 48 is added. The computer uses the sound subroutine at line 3000, then the printing routine at line 1000 (SCRNPRNT).

In ANTIC E, it takes a few seconds for the character or letter to be drawn or printed on the screen. By placing the marble on the screen using the player, the marble appears instantly. The printing routine draws the same marble on the screen behind the player. When the player is moved off the screen, the character marble is left in its place. The area the marble occupied in M$ is erased with B$. The loop continues until 10 marbles are placed on the screen.

Line 360 clears the screen. Using color 1, two lines are drawn on the screen. These lines define the playing field.

Line 370 sets variables PL1 and PL2 to 0. These variables are used to keep the score of both players. Variable H is set to 1. The next words printed on the screen are eight pixels high. The word "*one*" in lowercase is placed into WORD$. Normally, only uppercase *or* lowercase letters can be printed on the screen, but when ANTIC E is used, any letter or character can be printed as long as the character pattern can be sent to the subroutine. The word "one" is printed in orange beginning at row 0, column 0. The word "two" is printed in green in the same row but at the 262nd column. The computer was set to graphics 8 at the beginning of the program, so it counts every pixel in the row as a column.

Line 380 sets variable P to 120 and variable P1 to 65. This is the row and column where the player will place the marble on the screen. It is the first player's turn. Variables SP, C1, and R1 are cleared.

Line 390 disables the ATTRACT mode. Now the screen does not change colors while you are playing the game. If the trigger on the joystick is pressed, the computer goes to line 470 to move the marble.

Line 400 loops back to line 390 if the joystick has not been moved.

Line 410 decreases the value of P if the stick is moved to the up position. If the value of P is less than 33, it is reset to 33. The marble would not appear on the screen if the value were less.

Line 420 increases the value of P if the joystick is moved down. Again, the value of P is checked to make sure the marble doesn't disappear off the bottom of the screen.

Line 430 moves the marble to the right if the joystick is moved to the right. The new value of P1 is compared to 75. This is the last position at which the marble can be placed.

Line 440 moves the marble to the left if the joystick is moved to the left. If the value of P1 is less than 40, the marble would be off the screen, so variable P1 is reset to 40.

Line 450 clears the hit register at location 53278. The character M1$ is placed in M$ and the marble is moved to the new location by POKEing 53248 with the value of P1. Even though only the value of either P or P1 could be changed, this line resets the marble for both values. The computer checks location 53252. If it is 0, the marble has not hit anything on the screen and the computer is sent to line 390.

Line 460 subtracts 2 from the value of P1 and moves the marble two columns to the left if a character was hit.

Line 470 begins the routine to move the marble. Location 20 is the clock. This location is cleared.

Line 480 loops until the trigger button is released.

Line 490 takes the value of location 20 and multiplies it by 2. The value in this location will indicate how long the trigger button was pressed. This value will determine how far the marble travels across the field. The longer the button is pressed the greater the distance the marble travels. If the button is held too long, the distance will begin to shorten.

Line 500 checks the value of R. The shortest distance the marble can travel is to column 90.

Line 510 takes the value of the stick. The direction of the marble can also be controlled with the joystick. If the joystick is pushed up or down when the trigger button is released, the marble travels at an angle in the direction the joystick was pushed.

Line 520 clears the hit register. Variable PL is increased by 1. This moves the marble one column to the right. Variable ST is checked to see if it is 13. If it is, the joystick was pushed down. Variable P is increased by 1, but it cannot exceed 217.

Line 530—if the variable ST is 14 the joystick was pushed up and variable P is decreased by one. It cannot be less than 33.

Line 540 places the marble in M1$ in the correct position in M$. The new column is POKEd into location 53248. The computer then checks to see if the column value is less than the value of R and if no other marble has been hit. If both conditions are true, the computer goes to line 520 and moves the marble again. This routine continues until the marble is in the column set by R or it hits another marble.

Line 550 places the value of location 53252 into variable Q. If Q is a zero, nothing has been hit and the computer is sent to line 620.

Line 560 checks the value of Q for 4. If it is, then the marble crossed the field line, and the computer is sent to line 520 to continue across the field.

Line 565 subtracts 1 from the player's score if the value of Q is not 1. If the player hit the opponent's marble the value of Q is 1. If the player hit his own marble, he loses a point, the marble is removed from the screen and the computer is sent to line 630 where the new score is printed on the screen.

Line 570 adds 2 to the player's score when the opponent's marble is hit. The approximate column and row of the opponent's marble is calculated and stored in variables C and R.

Line 580 uses the LOCATE command to see what is at that position. If the value of the pixel at that location is 1, it is the opponent's marble. The value of R is decreased by 1, and the LOCATE command is tried again. This line loops until the value of Q1 is not 1.

Line 581 checks the three variables R1, SP, and Q1. If all three are 0, the value of R is increased.

Line 585 adds 1 to the value of SP. This variable counts how many times this line was executed. The value of R is decreased by 1 again. The actual height of the characters is 16 pixels high. The program was originally set to graphics 8, which uses one row of pixels for every byte. We changed the mode to ANTIC E, which uses two rows of pixels for every byte placed on the screen. When

we use the LOCATE command, the computer thinks it is in graphics 8, so it looks at a location only one bit wide by one pixel high. We need to be three rows above the last row that was a one, because the top of the marble has no corners.

Line 586 resets variable SP to 0 and disables the TRAP.

Line 590 uses the same principle to find the column in which the opponent's marble begins. The LOCATE command gets the value of the pixel indicated by the row and column variables. If the value is 1, the value of variable C is decreased by 2. Each column in ANTIC E is two pixels wide; in graphics 8, it is only one pixel wide. The computer thinks it is in graphics 8, so we decrease the column variable by 2 to be in the next column.

Line 600 increases variable SP by 1 and decreases the column variable by 2. The computer is sent to line 590 until variable SP is 3. This places the column variable in the correct position to erase the opponent's marble.

Line 610 sets the variables for the row and column. The sound routine is called, then WORD$ is set to a space. The computer uses the subroutine at line 1000 to print the space on the screen. This erases the opponent's marble.

Line 620 sets the color variable to 2 and places the player's marble character into WORD$. The row and column position of the player's marble is calculated and the computer uses the subroutine at line 1000 to print the marble on the screen. This marble is printed in the same color as the player marble, and is printed behind the player marble. When the player is moved off the screen, the character remains.

Line 630 sets the row and column again so the computer can print the new score on the screen. The player's score is checked to see if it is less than 0. If so, the score is reset to 0.

Line 635 changes the contents of variable PL1 into a string and places it in WORD$. The computer uses the subroutine at line 1000 to print the score on the screen, then checks to see if the score is greater than 9. If it is, the player has won and the computer is sent to line 950 to end the game.

Line 640 erases the marble character from the player. The new values for the second marble are set, the color of the player is changed to the color of the second player's character, and the player with the marble is placed on the screen by POKEing the value of P1 into location 53248 and placing M2$ in M$. The variables used to locate the hit marble are cleared. It is now the second player's turn.

Line 650 disables the ATTRACT mode so the screen will not change colors. Then the computer checks to see if the fire button has been pressed. This game is set up for one joystick; if you want to make this a two-player, two-joystick game simply change the commands in the next few lines to STRIG(1) and STICK(1). Then the computer will check the second joystick port for the second player.

Line 660 checks to see if the joystick has been moved. If it has not, the computer loops back to line 650. These two lines repeat themselves until the fire button has been pressed or the joystick is moved.

Line 670 decreases the value of P by 1 when the joystick is moved up. If the value of P is less than 33, it is reset to 33. A value less than 33 would place a marble off the top of the screen.

Line 680 checks to see if the joystick has been moved down. If it has, the value of P is increased by 1. This time the value is checked to see if it is greater than 217. The value of P cannot exceed 217 or the marble will not be visible on the bottom of the screen.

Line 690 checks to see if the joystick has been moved to the right. If it has, the value of P1 is increased by 1. A value greater than 203 will place the marble off the right edge of the screen.

Line 700 decreases the value of P1 by 1 when the joystick is moved to the left. The value of P1 cannot be less than 170 or the marble will cross the line on the field.

Line 710 clears the hit register by POKEing location 53278 with 0. The marble is placed into M$ at the location indicated by variable P, and moved on the screen by POKEing its location into 53248. The computer then looks at location 53252. If it is 0, the marble has not hit anything on the field and the computer goes back to line 650 to wait for another move by the joystick.

Line 720 adds two to variable P1 and POKEs this new value into location 53248. This moves the marble back from the character that it hit.

Line 730 clears location 20. This is a clock. This location will be used later to see how long the fire button was pressed.

Line 740 checks to see if the fire button is depressed. The computer loops at this line until the button is released.

Line 750 calculates the new position of the marble based on the length of time the fire button was pressed. The longer the button is held down, the further the marble will roll, up to a point. If the button is held too long, the distance decreases. The column

value cannot be less than 95.

Line 760 checks to see if the column value is larger than 160. If it is, it is reset to 160 so the marble rolls into the field.

Line 770 looks at the joystick to see if it has been moved. The value of the joystick port is placed into variable ST. If the joystick is moved up or down when the fire button is being pressed, the marble travels on a diagonal.

Line 780 decreases variable P1. This is the column to which the marble will be moved. If the joystick has been moved down, variable P is increased by 1, but its value cannot exceed 217.

Line 790 checks to see if the joystick has been moved up. If it has, variable P is decreased by 1, but cannot be less than 33.

Line 800 clears the hit register. The character for the second marble is placed in M$ at the position indicated by variable P. The column position is POKEd into location 53248. If column value P is greater than the column the marble should be moved into, and the marble has not hit another marble (location 53252 is 0), the computer is sent to line 780 to move the marble again. This loop continues until the marble is in the column whose value is the same as the value of variable R, or the marble hits an opponent's marble.

Line 810 places the value of location 53252 in variable Q. If the value is 0, the marble did not hit another marble and the computer is sent to line 880.

Line 820 checks to see if the value of Q is 4. If so, the marble crossed the field line and the computer goes to line 780 to continue the loop.

Line 825 checks to see if Q is 2. If it is not, the player's marble hit his own marble. One point is subtracted from the score, the marble is removed from the screen, and the computer is sent to line 890.

Line 830 adds 2 to score variable PL2 because the player's marble hit the opponent's marble. The row and column of the opponent's marble is calculated. These two values are used to find the opponent's marble.

Line 840 sets the TRAP should an error occur. The computer then checks the value of the pixel at the location set by variables C and R. If the value of Q1 is 1, the pixel contains a color; the value for the row is decreased by 1 and the line is repeated. The computer will loop at this line until variable Q1 does not contain 1, which means the opponent's marble is not at that position.

Line 841 checks to see if all the variables contain 0. If they do, the row variable is increased by 1 and line 840 is repeated.

Line 845 adds 1 to variable SP. This variable counts how many

rows contain a value other than one. Variable R is decreased by 1, and the value of SP is compared to 3. If it is less, the computer goes back to line 840. Because of the shape of the marble character, we need to begin the character three rows before the first pixel that contains a 1.

Line 846 resets variable SP to 0 so it is ready for the next counting routine. The trap is removed and the program continues.

Line 850 is similar to the last routine. This time we are looking for the column the marble begins in. If variable Q1 contains a 1, the pixel is used for the marble character. Variable C is decreased by 2 because two bits of the byte are used to color the marble in this mode. The line repeats until the value of Q1 is not 1.

Line 860 increments variable SP, decreases variable C by 2 and checks to see if variable SP is less than 3. If it is not, the computer goes back to line 840 and repeats this routine until the column variable is six positions before the last column used for the marble.

Line 870 subtracts 2 from variable C1, sets the row to variable R1, and uses the subroutine at line 3000 to make a sound. The space is placed into WORD$, and the subroutine beginning at line 1000 prints the space on the screen. This erases the opponent's marble from the screen.

Line 880 sets the color routine value to 3, places the marble for this player in WORD$, calculates the row and column of the marble, and uses the subroutine at line 1000 to print the marble on the screen. This marble is printed in the same position as the player marble already on the screen. Now when the player is removed, the marble remains in position. The value of PL2 is checked to see if it is less than 0. If it is, it is reset to 0, since the score for any player cannot be less than that.

Line 890 sets the row and column to place this player's score in the correct position on the screen. Variable PL2 is changed to a string and placed in WORD$.

Line 900 uses the subroutine at line 1000 to print the new score on the screen. The score is checked again to see if it is less than 10. If it is, this player's marble is erased from the player area and the computer is sent to line 380 for the first player's turn.

Line 950 is the end of the game routine. The marble is erased from the player area and the screen, the screen is cleared, and the phrase "GAME OVER" is placed in WORD$. The variables for the row, column, height of the letters, and color routine are set and the computer uses the subroutine at line 1000 to print the message on the screen.

Line 960 places PLAYER1 in WORD$, then converts this player's score to a string and places it in the string variable. Again, the row, column, letter height and color routine are set and the computer uses the subroutine at line 1000 to print this information on the screen.

Line 970 changes the player number in the screen from 1 to 2, and places the second player's score in the string. The color routine, height, and row variables are set and the subroutine at line 1000 prints this information on the screen.

Lines 980-990 loop until the Start key is pressed. When this key is pressed, the computer goes to line 360 and begins a new game.

Line 1000 is the subroutine that prints the messages on the screen. Because we are using ANTIC E in this program, the characters, letters, and numbers cannot be printed on the screen using the conventional PRINT command. Each character must be taken byte-by-byte and changed into two values. These new values represent the code that will be printed on the screen so the character will appear correctly on the screen. The value of variable H is transferred into variable TH. This value determines the height of the character. The value of variable R is transferred into temporary variable TR. This is the row on which the character is printed. Variable L contains the length of the string and CHBASE is the beginning of the new character set in RAM.

Line 1010 begins the FOR-NEXT loop that takes each letter of the message individually and recodes it for this mode. The ATASCII value of the letter is placed in variable LASC. If the value of this letter is greater than 31 and less than 96, the variable must be adjusted by subtracting 32 from itself. The characters in the character set are *not* in ATASCII order; therefore this adjustment must be made. If it is not, the message appearing on the screen will differ from the message in WORD$. If this adjustment is made, the computer is directed to line 1030.

Line 1020 checks to see if the ATASCII value of this character is less than 32. If it is, this is a graphics character and 64 is added to its value. The graphic characters are located after the uppercase letters and before the lowercase letters in the character set.

Line 1030 begins another FOR-NEXT loop. This routine takes each byte of the character, one byte at a time, and converts it into two bytes. The character is found by adding the value of LASC (multiplied by 8) to the beginning of the character set, then adding the value of LP. Variable LP determines which byte of the character

will be converted. The first four bits of the byte are calculated by dividing the value of the byte by 16. The second four bits are arrived at by subtracting the value of the first byte, multiplied by 16, from the value of the byte. Now the correct code for each *nybble* (four bits) can be printed on the screen.

Line 1040 uses the ON-GOTO command to send the computer to the correct color code line. The value of CL determines which color codes are used.

Line 1050 is used when the value of CL is 1. The character is printed in blue on the screen. The computer takes the value of the first four bits (stored in FBYTE), multiplies it by 8, and adds 1. This is the position in BLBYTE for the correct code for this value. The last element of this code is found eight positions later. The value of FBYTE is increased by 1, then multiplied by 8. This code is stored in the first eight elements of BYTE$. The next four bits are converted in the same manner. This code is stored in the next eight elements of BYTE$.

The computer is sent to line 1080 to print the code on the screen. The values of FBYTE or SBYTE can be any value from 0 to 15. One must be added to the value after it is multiplied by 8 to find the correct position in the string because the string begins with the first position. If the value of FBYTE is 0, the computer would try to begin with the zeroth element and an error would result. By adding 1, the computer begins with the correct element for the code. The code placed in BYTE$ is taken from the string set up in the beginning of the program. BLBYTE$ contains the correct two-bit codes to place blue characters on the screen.

Line 1060 is the same as line 1050 with the exception of the string the codes are taken from. This line is used when the value of CL is 2 and green characters are needed. The codes are removed from GRBYTE$.

Line 1070 take the codes from ORBYTE$ and places them in BYTE$. This line is used when the value of CL is 3. The characters printed on the screen are orange.

Line 1080 uses the values of C and R to print the byte on the screen. This is only one row of the characters. Variable TH is decreased by 1 and the row variable is increased by 1. If variable TH is not 0, the line repeats and the byte is printed again on the screen, this time one row below the last byte. Variable TH determines the height of the character, so the message can be eight rows high with a value of TH, or cover the entire screen.

Line 1090 resets variable TH. The loop continues until all eight

bytes that make up this character are printed on the screen. After this character is on the screen, variable R is reset and the column variable is increased by 16. This moves the next letter over the correct distance on the screen. A value of less than 16 would place the characters on top of each other, and a value greater than 16 would spread the letters too far apart on the screen. The second loop continues until the entire message has been printed on the screen.

Line 1100 returns the computer to the line that called this routine.

Line 3000 is the sound subroutine, used whenever a marble hits another marble.

JACKS

Objective of the game: To collect a certain number jacks before the ball hits the ground.

Directions: This game is written for one to four players. There are four levels of play. One joystick is needed for this game.

The game begins by asking for the number of players and the level of play. At level one, the ball travels very slowly on the screen. At each level the ball increases in speed. At level four you need exceptional hand-eye coordination.

The screen clears, the player whose turn it is appears on the screen. The level the player is at is placed in the upper right corner. This is the number of jacks that must be collected before the ball hits the ground. Near the ground on the right side of the screen is a number. This number changes as more jacks are collected. This lets the player keep track of how many jacks are in the cup. At higher levels it is difficult to remember how many jacks have already been picked up. A blue cup is on the left side of the screen near the ground. Ten jacks are scattered on the purple ground.

Push forward on the joystick. The length of time the joystick is pushed determines how high the ball travels on the screen. If the joystick is pressed too long the ball will not travel very high. Once the ball begins to travel up the screen, the joystick can be moved in any of the four directions. Press the fire button to turn the cup over. A jack cannot be picked up by the cup unless the wider part of the cup is on the bottom. Move the cup over a jack. The number on the right side of the screen changes to one. If you are at level one, press the fire button again and position the cup under the ball. The cup must be in the proper position to catch the falling ball. If the cup catches the ball, the cup is moved to the start-

ing position and you play again. The game continues until the ball is dropped or all 10 jacks have been picked up. If the ball is dropped, the turn ends and it is the next player's turn. If all 10 jacks have been picked up, another group of 10 jacks are scattered on the ground, the number in the upper right corner of the screen changes to 2 and the game continues. The game ends when one player collects all 10 jacks in one turn. Figure 5-5 is the flowchart, Fig. 5-6 is the character set, and Listing 5-3 is the code for this program.

Line 50 sets aside the string space needed for this program. B$ must be the first variable used. It will be changed into a player and used for the ball. TEMP$ holds the code for the ball and JK is a numeric array used to hold the scores of the players.

Line 60 finds out how much memory is available by PEEKing at location 106. The program moves the character set from ROM into RAM and uses some of the memory for player/missile graphics. The value of A contains the starting address of the character set and the player/missile graphics area. This value is POKEd into location 204. The starting address of the ROM character set is stored in location 206. These two addresses are used by the assembly language subroutine that moves the character set from ROM into RAM.

Line 70 reads the assembly language subroutine from line 80 and places it into RAM. This begins with location 1536 and ends with location 1555. Line 80 is the decimal code for the routine.

Line 90 changes the screen to graphics mode 17. This is graphics 1 with no text window. The USR command tells the computer to use the assembly language subroutine that begins at location 1536. When the computer completes the subroutine, it POKEs the value of A into location 756. This tells the computer to use the character set that begins at this location. If the value of A is incorrect, the screen contains garbage after the command is executed. Variable CHARSET contains the address of the fourth character in the character set. Variable A contains the high-order address of the character set. To get the decimal address, the value of A must be multiplied by 256. The code fourth character begins with the 32nd byte of the character set. The FOR-NEXT loop reads the code for the new characters and places it in the character set. These characters are the two different jacks and the cup in the upright and inverted positions.

Line 100 POKEs the value of A into location 54279. This tells the computer where the player/missile graphics begin. Since the

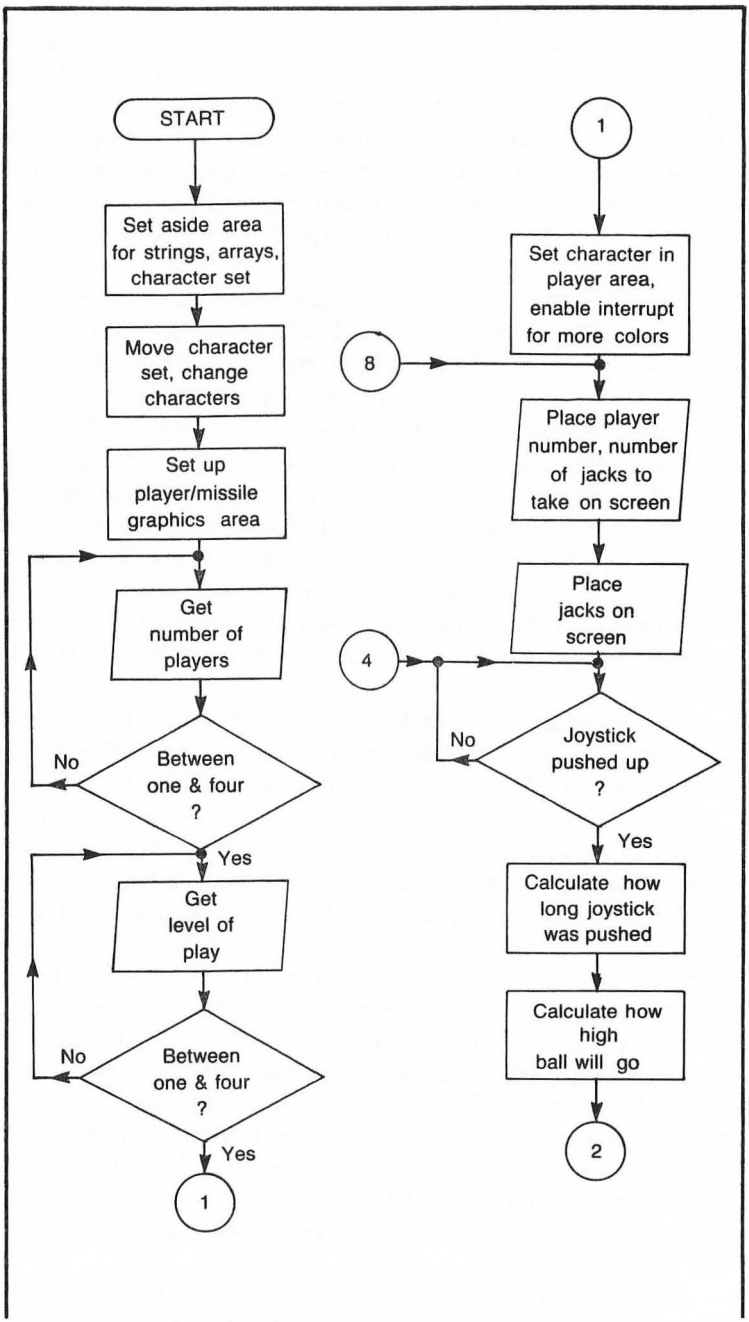
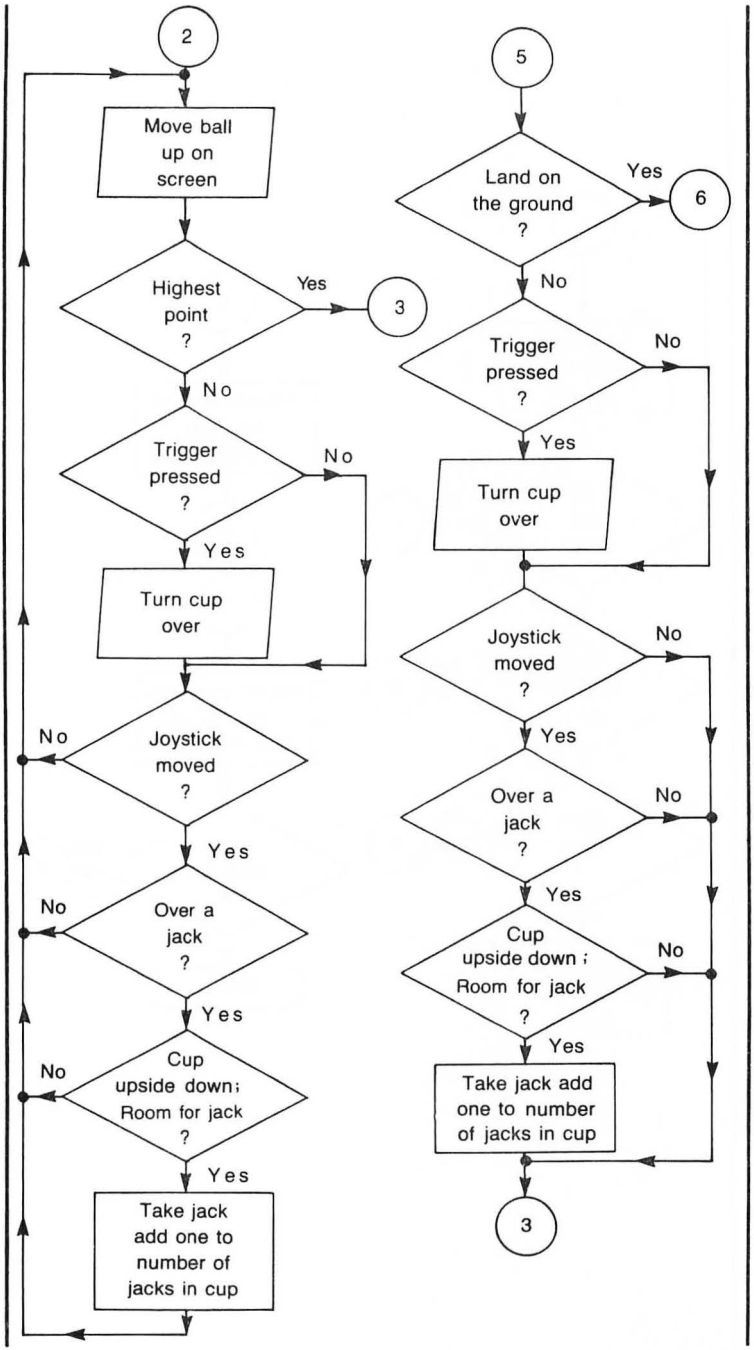
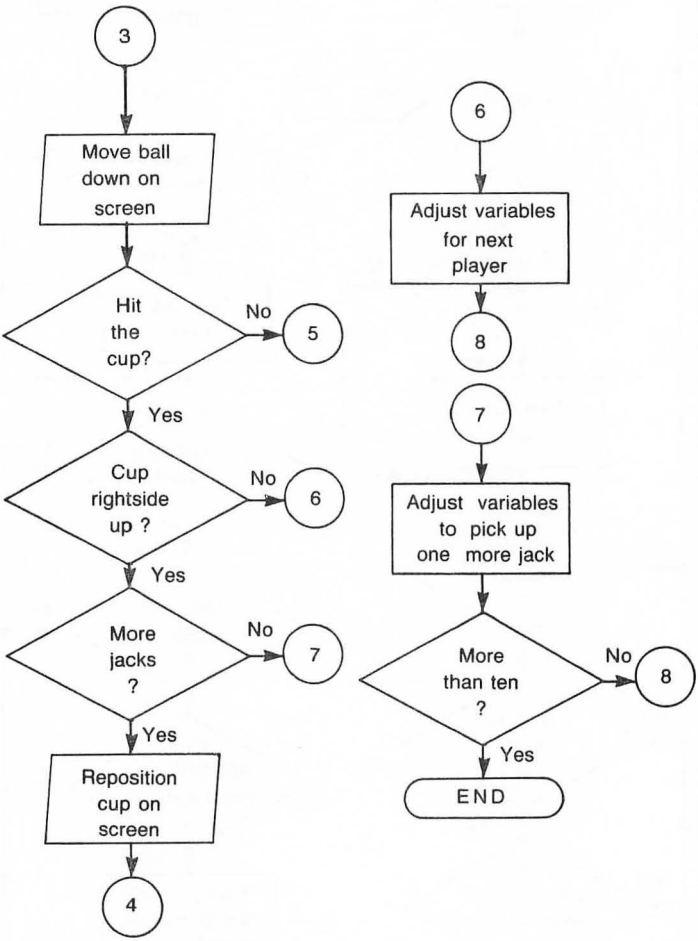


Fig. 5-5. Flowchart for Jacks. (Continued through page 258.)


2
Move ball up on screen
Highest point ?
Yes
3
No
Trigger pressed ?
No
Yes
Turn cup over
Joystick moved ?
No
Yes
Over a jack ?
No
Yes
Cup upside down ; Room for jack ?
No
Yes
Take jack add one to number of jacks in cup
5
Land on the ground ?
Yes
6
No
Trigger pressed ?
No
Yes
Turn cup over
Joystick moved ?
No
Yes
Over a jack ?
No
Yes
Cup upside down ; Room for jack ?
No
Yes
Take jack add one to number of jacks in cup
3


3
Move ball down on screen
Hit the cup?
No
5
Yes
Cup rightside up ?
No
6
Yes
More jacks ?
No
7
Yes
Reposition cup on screen
4
6
Adjust variables for next player
8
7
Adjust variables to pick up one more jack
More than ten ?
No
8
Yes
END

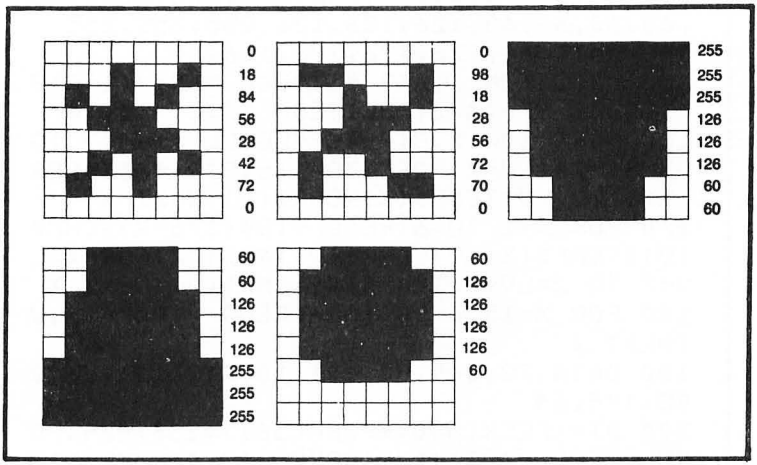


Fig. 5-6. Character set for Jacks.

Listing 5-3. Jacks.

```
10 REM JACKS - A YOUNG GIRL'S GAME
20 REM CHAPTER 5 - SIMULATIONS
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM B$(1),TEMP$(16),JK(4)
60 A=PEEK(106)-16:POKE 204,A:POKE 206,
PEEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256+32:FOR X=CHARSET TO CHAR
SET+31:READ V:POKE X,V:NEXT X
100 POKE 54279,A:POKE 559,62:POKE 5327
7,3:POKE 53248,0:VT=PEEK(134)+PEEK(135
)*256:STTB=PEEK(140)+PEEK(141)*256
110 STROFF=A*256+1024-STTB:ST3=INT(STR
OFF/256):ST4=STROFF-256*ST3:POKE VT+2,
ST4:POKE VT+3,ST3:POKE VT+4,255
120 POKE VT+5,0:POKE VT+6,255:POKE VT+
7,0:B$(1)=CHR$(0):B$(255)=CHR$(0):B$(2
)=B$:POKE 53248,0:POKE 623,4
130 DATA 0,18,84,56,28,42,72,0,0,98,18
```

```
,28,56,72,70,0,255,255,255,126,126,126
,60,60,60,60,126,126,126,255,255,255
140 DATA 60,126,126,126,126,60
150 POSITION 2,5:? #6;"HOW MANY (1-4)?
":GOSUB 570:A=Z:POSITION 17,5:? #6;A
160 POSITION 2,8:? #6;"LEVEL (1-4)?":G
OSUB 570:LV=Z:POSITION 15,8:? #6;Z:FOR
 X=1 TO LV:TEMP$(X)=CHR$(0):NEXT X
170 FOR X=1 TO 6:READ V:B$(176+X)=CHR$
(V):TEMP$(X+LV)=CHR$(V):NEXT X:FOR X=L
V+7 TO 2*LV+6:TEMP$(X)=CHR$(0):NEXT X
180 FOR X=1536 TO 1546:READ V:POKE X,V
:NEXT X
190 DATA 72,169,98,141,10,212,141,26,2
08,104,64
200 DI=(PEEK(560)+PEEK(561)*256)+24:PO
KE DI,PEEK(DI)+128:POKE 512,0:POKE 513
,6:POKE 54286,192:REM THE INTERRUPT
210 POKE 711,182:POKE 704,52:CP=166:BC
=88:BR=177:IC=0:FOR X=1 TO 4:JK(X)=1:N
EXT X
220 FOR P=1 TO A:POSITION 0,0:? #6;CHR
$(125);"player ";P:POSITION 17,0:? #6;
"  ":REM player IS INVERSE
230 POSITION 17,0:? #6;JK(P):FOR X=1 T
O 10:J=INT(RND(1)*2)+4:REM LIGHT GREEN
 JACKS
240 R=INT(RND(1)*4)+20:C=INT(RND(1)*15
)+2:LOCATE C,R,O:IF O<>32 THEN 240
250 POSITION C,R:? #6;CHR$(J):NEXT X:T
J=10:TP=JK(P):ST=1:POSITION 17,19:? #6
;"   "
260 C=5:R=18:POSITION C,R:? #6;CHR$(CP
)
270 POKE 20,0:IF STICK(0)<>14 THEN 270
280 IF STICK(0)=14 THEN 280
290 HT=INT(PEEK(20)/2):IF HT<61 THEN H
T=30+(30-INT(HT/2))
300 GOSUB 620:B$(255)=CHR$(0):POKE 532
48,BC:O=32:OC=C:OR=R
310 POSITION 18,19:? #6;IC:IF STRIG(0)
=0 AND PEEK(20)>30 THEN CP=333-CP:POKE
 20,0
320 IF STICK(0)=15 THEN 380
330 POKE 77,0:IF STICK(0)=7 THEN POSIT
ION C,R:? #6;" ":C=C+1:IF C=18 THEN C=
17
```

```
340 IF STICK(0)=11 THEN POSITION C,R:?
 #6;" ":C=C-1:IF C=0 THEN C=1
350 IF STICK(0)=13 THEN POSITION C,R:?
 #6;" ":R=R+1:IF R=24 THEN R=23
360 IF STICK(0)=14 THEN POSITION C,R:?
 #6;" ":R=R-1:IF R=17 THEN R=18
370 POSITION OC,OR:? #6;CHR$(O):LOCATE
 C,R,O
380 POKE 53278,0:POSITION C,R:? #6;CHR
$(CP):OC=C:OR=R
390 BR=BR-ST*LV:IF BR<=HT THEN ST=-1:B
C=BC+(INT(RND(1)*HT)+1/2):POKE 53248,B
C
400 B$(BR)=TEMP$:IF BR>=185 THEN GOSUB
 610:GOSUB 550:NEXT P:GOTO 220
410 IF IC<>JK(P) AND O<>32 AND CP=167
THEN IC=IC+1:O=32:TJ=TJ-1:GOSUB 630
420 IF ST<>-1 OR PEEK(53252)<>4 OR CP<
>166 THEN 310
430 IF IC<>TP THEN BR=185:GOTO 400
440 GOSUB 620:GOSUB 550:IF TJ<JK(P) TH
EN TP=TJ
450 IF TJ<>0 THEN 260
460 JK(P)=JK(P)+1:IF JK(P)<>11 THEN 23
0
470 POSITION 3,8:? #6;"THE WINNER IS
      PLAYER ";P
480 POSITION 2,11:? #6;"play again (y-
n)"
490 OPEN #2,4,0,"K:":GET #2,Z:CLOSE #2
500 IF Z>127 THEN Z=Z-128
510 IF Z=78 OR Z=110 THEN 600
520 IF Z=59 OR Z=121 THEN 210
530 GOTO 490
550 POSITION C,R:? #6;" ":B$(1)=CHR$(0
):B$(255)=CHR$(0):B$(2)=B$:CP=166:ST=1
:IC=0:BC=88:BR=177
560 B$(BR-LV)=TEMP$:POKE 53278,0:POKE
53248,BC:RETURN
570 OPEN #2,4,0,"K:":GET #2,Z:CLOSE #2
:IF Z>127 THEN Z=Z-128
580 Z=Z-48:IF Z<1 OR Z>4 THEN 570
590 RETURN
600 POKE 53248,0:END
610 SOUND 0,220,8,10:FOR X=1 TO 10:NEX
T X:SOUND 0,0,0,0:RETURN :REM MISS
```

```
620 SOUND 0,12,6,10:FOR X=1 TO 10:NEXT
 X:SOUND 0,0,0,0:RETURN :REM THROW AND
 CATCH
630 SOUND 0,12,10,10:FOR X=1 TO 10:NEX
T X:SOUND 0,0,0,0:RETURN :REM TAKE JAC
K
```

computer does not use the first 1K of the area set aside for player/missile graphics, we can and do use it for our character set. By POKEing location 559 with 62, we tell the computer we are using a single-resolution player. POKE location 53277 with 3 to enable the player/missile graphics. POKE location 53248 with 0 to make sure the player is off the screen. The memory used for the player may contain garbage; we do not want that on the screen. The VT variable is the location of the variable value table. This table stores information for all the variables, string and numeric, used in the program. The location of this table is stored in locations 134 and 135. The STTB variable contains the beginning address of the string storage area. The address of the area is stored by the computer in locations 140 and 141.

Line 110 calculates the offset of the first string. When we entered line 50, B$ had to be the first variable. The variable value table contains an offset for the strings. This offset value is added to the address of the string storage area. This way the computer knows where the information for a string or variable is located. We want to change this offset to point to the player/missile area for B$ rather than the actual string storage area.

To get the offset we need to multiply the value of A by 256. This is the beginning address of the player/missile graphics area, but not the beginning of the first player. The first player begins 1K past this address, so we add 1024 to this value. Since the computer adds the address of the string storage area to the offset, we need to subtract this amount from the player area. This is the offset stored in the variable value table. If the number is larger than 255 it must be divided into two numbers, the high-order byte (the integer of the address divided by 256), and the low-order byte (multiply the high order byte by 256 and subtract the product from the original number). These two numbers are POKEd into the third and fourth locations of the first variable in the variable value table. The fifth location of the table is changed to 255. The length of a player is 255 bytes.

Line 120 changes the sixth location in the variable table to 0.

Two locations are used for the length and dimension values of the string. The first is the low-order byte, the second the high-order byte. The seventh location in the variable table is also set to 255, the eighth to 0. The storage area for B$ is located in the player/missile graphics area. B$ is 255 bytes long and has been dimensioned to 255 bytes. To clear this area, set the first and last element of B$ to a CHR$(0). Then set the second element of B$ to B$. This clears the entire string. Location 623 sets the priorities for the players and characters. This location is set to 4. The cup will cover the ball and jacks on the screen.

Lines 130-140 contain the code for the jacks, the cup, and the ball.

Line 150 asks how many players will play this game. The computer uses the subroutine at line 570 to get the answer. That number is stored in variable A. This number is printed on the screen.

Line 160 asks which level will be used. The subroutine at line 570 is used again. The number is stored in variable LV and is printed on the screen. The FOR-NEXT loop clears any information that might be in TEMP$.

Line 170 reads the code for the ball from line 140 and places it in B$ and TEMP$. The remaining locations of TEMP$ are cleared. This way the computer will take the entire string when placing the ball in B$ rather than the last element used by the ball.

Line 180 reads the assembly language subroutine from line 190 and places it in locations 1536-1546. The assembly language subroutine is used by the computer to change the color of the bottom portion of the screen. Line 190 contains the decimal code for this routine.

Line 200 finds the beginning address of the display list. The display list tells the computer, row by row, which mode to use on the screen. To add an assembly language subroutine the computer will execute every time it uses the display list, we have to tell the computer there is, in fact, one to use. We do this by setting the high-order bit of the code, or adding 128 to the existing code. The color on the screen will be changed at the 20th row. We add 24 to the display list because the first three bytes are not used and there are two bytes used for instruction. The first byte is byte zero, 20 plus 4 is 24. The computer PEEKs at this location and adds 128 to the value it finds there.

Now the computer knows that it should use an assembly language subroutine after this row is placed on the screen. The address for the subroutine is POKEd into locations 512 and 513. The

low- order address is POKEd into location 512 and the high-order address is POKEd into 513. POKE 192 into location 54286 to enable the interrupt.

Line 210 sets the color for the fourth character, lowercase inverse, and the first player, the ball. Variable CP is set to 166. This is the character code for the cup in blue and in the upright position. Variable BC is set to 88. This is the position of the ball on the screen when it is under the cup. BR is set to 177. This is the first row in which the ball will appear when it is released from the cup. IC is set to 0. This is the number of jacks in the cup. The FOR-NEXT loop sets the level of play for all the players to one.

Line 220 begins the game. The game loops from one to the number of players. For each player, the screen is cleared, and the word "PLAYER" and the player's number are printed at the top of the screen.

Line 230 prints the number of jacks the player must pick up at a time in the upper right corner. The FOR-NEXT loop places 10 jacks on the floor. The computer chooses one of two jacks.

Line 240 chooses a position for the jack. The R variable is the row and variable C is the column. The LOCATE command checks to see if that position is empty. If variable O is 32 the location is empty, otherwise this line is repeated until an empty place is found.

Line 250 prints the jack on the screen. The loop continues until all 10 jacks have been placed. The TJ variable is set to 10. This is the number of jacks on the floor. Variable TP is set to the amount in the JK array at location P. This is the number of jacks that must be picked up at one time.

Line 260 sets the variables for the row and column of the cup. The cup is printed on the screen in the upright position.

Line 270 clears the amount in location 20. This is a clock location. The joystick is checked to see if it is pushed forward. If it is not, the line loops.

Line 280 loops as long as the joystick is pushed forward. The height of the ball is determined by how long the joystick is pushed forward.

Line 290 takes the value of location 20 and divides it by 2. If the amount is less than 61, a formula will be used to figure out how high the ball should travel.

Line 300 uses the subroutine in line 620 to make a sound. The last element of B$ is cleared and the ball is placed on the screen by POKEing location 53248 with the value of BC. Variable O is

set to 32. This variable will hold the value of the character the cup will cover. The values of R and C are stored in temporary variables OR and OC.

Line 310 prints the number of jacks in the cup on the screen. Until a jack is picked up, this variable is 0. The fire button is checked to see if it is pressed and if the value of location 20 is greater than 30. If it is, the value of CP is subtracted from 333 and location 20 is cleared. There are two characters that can be used for the cup, character 166 and character 167. To alternate between the two characters, the current character is subtracted from 333, the sum of the two characters. This way, the variable is set to the other character without using IF-THEN statements or other commands. This method is much faster and more efficient, and it can be used in any routine that alternates between two specific numbers. Location 20 is used so the cup keeps flipping on the screen. Waiting until the value of location 20 is more than 30 gives the player a chance to take his finger off the fire button. This location is reset every time the cup is flipped.

Line 320 checks to see if the joystick is moved. If it isn't, the computer is directed to line 380.

Line 330 disables the attract mode by POKEing 0 into location 77. If the joystick is moved to the right, the cup is erased from its current position and the value of variable C is increased by 1. The value of the column variable is checked to make sure it doesn't exceed 17. If it does, it is reset to 17.

Line 340 checks to see if the joystick is pushed left. The cup is erased and variable C is decreased by one. If the variable reached 0, it is reset to 1. This keeps the cup on the screen.

Line 350 checks to see if the joystick is pushed down. If it is, the cup is erased and variable R is increased by 1 and checked to see if it is equal to 24. The variable is reset to 23 if it is.

Line 360 checks to see if the joystick is pushed up. It erases the cup and decreases variable R. This variable will be reset to 18 if it reaches 17.

Line 370 prints the CHR$ of variable O in the old row and old column. The first time the variable will be 32. If the cup goes over a jack and doesn't pick it up, the character value of that jack will be stored in variable O. Then when the cup is moved again, the jack is printed on the scrèen again. The LOCATE command is used to get the value of the character the cup is printed over. This value is stored in variable O.

Line 380 clears the hit register. Then the cup is printed on the screen. The new position of the cup is stored in variables OC and OR.

Line 390 calculates the position of the ball in B$ by multiplying the value of ST by LV and subtracting it from the value of BR. If the game is at level one and the value of ST is 1, the value of BR will decrease by 1 each time this line is executed. The ball travels very slowly up the screen at level one. If variable ST is −1, the value of BR increases and the ball travels down the screen. The value of BR is compared to the value of HT. This is the highest position the ball can travel to on the screen. If the ball is at its peak, the value of ST is changed to −1. A new value for the column the ball travels down is calculated and the new position is POKEd into location 53248.

Line 400 places the ball in B$ at position BR. The character codes for the ball are in TEMP$. Since the storage area for B$ is the player/missile area, the ball is on the screen and moves immediately. If the value of BR is 185 or more, the cup did not catch the ball. The subroutine at line 610 makes a sound and the computer is sent to the subroutine at line 550 to erase the ball and move the cup. The loop continues. After all the players have a turn, the computer is sent to line 220 to continue the game.

Line 410 compares the number of jacks in the cup with the number that should have been picked up. If these two values are not the same, the value of O is not a space, and the cup is upside down, the computer picks up the jack by increasing the value of IC, changing the value of O to 32 (space) and decreasing the value of TJ. The computer uses the subroutine at line 630 to make a sound.

Line 420 checks to see if the joystick can still move the cup. If the value of ST is −1, or ball has not hit the cup, or the cup is not in the upright position, the computer goes to line 310 and the cup can still be moved. If variable ST is −1, the ball was moving down. If variable CP is 166, the cup is upright, and if location 53252 contains a 4, the ball hit the cup. When all three of these conditions are true, the ball has been caught by the cup and the computer executes the next line.

Line 430 checks to see if the number of jacks in the cup is correct. If it is not, the value of BR is set to 185 and the computer is sent to line 400. This is the same as missing the ball.

Line 440 uses the subroutine at line 620 to make a sound. It then uses the subroutine at line 550 to move the cup and ball back

to the starting position. The computer then checks to see if the number of jacks left on the floor is less than the number of jacks that should be picked up at one time. If there are less, the value for the number of jacks that should be picked up is changed. This happens on almost every turn, beginning with "threes." The pickup sequence for "threes" are three, three, three, one. On the next turn the jacks are picked up by fours: four, four, two. If the value of TP is not changed, the computer does not accept the last number of the sequence.

Line 450 checks to see if all the jacks have been picked up. If TJ is not 0, the computer is sent to line 260 to continue the game.

Line 460 is used when all the jacks have been picked up. The number of jacks to be picked up at one time are increased by 1. If this value is not 11, the computer is sent to line 230 to continue the game.

Line 470 declares the winner of the game. The first player to pick up all 10 jacks in one turn wins.

Line 480 asks if you want to play again.

Line 490 waits for a response.

Line 500 checks to see if the value of Z is greater than 127. If it is, 128 is subtracted from the value.

Line 510 checks to see if the value of Z is an upper- or lowercase N. If so, the computer is sent to line 600 to end the game.

Line 520 checks to see if an upper- or lowercase Y was entered. If so, the computer is sent to line 210 for another game.

Line 530 sends the computer to line 490 to wait for another entry.

Line 550 erases the character at the screen position determined by variables R and C. The B$ is cleared and the variables are reset.

Line 560 places the ball into the correct position of B$, clears the hit register, and places the ball on the screen. The computer then returns to the line that called this routine.

Line 570 is the routine that gets the number for the two questions at the beginning of the game. If the value of the key pressed is greater than 127, the inverse key has been pressed. The value of Z is determined by subtracting 128 from its value.

Line 580 subtracts 48 from the value of Z. This gives another value. If a correct number key has been pressed, the value is between 1 and 4. If the number does not fall between these values, line 570 is repeated.

Line 590 sends the computer back to the line that called it.

Line 600 ends the game. Location 53248 is POKEd with 0 to

make sure the player is removed from the screen.

Lines 610-630 are sound routines. The routine used depends on whether the cup was missed, the ball is being thrown or caught, or a jack is taken.

SKI

Objective of the game: To make it through the course without hitting any flags or trees.

Directions: This is a one-player program. One joystick is needed. It should be plugged into the first joystick port.

You will be asked for a level of play, of which there are five. The higher the level, the more trees you will have to avoid. You will also be asked how many gates. The number you enter will give you that number of rows of gates with five flags in each row.

After you answer both questions, the screen clears and the course is printed on the screen. The skier is in the upper left corner. To score points, you must maneuver him under the red flags and over the blue flags. Sometimes there is a tree quite close to the flag. Your skier can squat down to avoid the trees. Push the fire button and the skier becomes smaller. If you hit a flag, points are subtracted from your score. If you hit a tree, the words "PRESS START" appear on the screen. The game can be played again by pressing the Start key. Figure 5-7 is the flowchart, Fig. 5-8 is the character set for this program, and Listing 5-4 is the code.

Line 50 sets aside the memory needed for the strings. SKI$ must be the first variable used in this program. It will be relocated to the player/missile area of memory. The four subsequent SKI$s contain the four different characters that can make up the skier. The NUMBER variable is set to 1000. This is the line number for the subroutine that reads the keyboard.

Line 60 uses the PEEK command to locate the end of RAM. We need 4K for the character set and the player/missile graphics area. The beginning location for the new character set is POKEd into location 204 and the location of the character in ROM is POKEd into location 206. These two locations are used by the assembly language subroutine when it moves the character set from ROM into RAM.

Line 70 reads the code for the assembly language subroutine from line 80 and places it in memory locations 1536-1555. This is the assembly language subroutine that moves the character set from ROM into RAM. Line 80 contains the decimal code for the assembly language subroutine.

assembly language subroutine. The beginning address of the subroutine is placed within the parentheses. The new address of the character set is POKEd into location 756. The new characters begin with the third character in the character set. To calculate where this character is located, we multiply the value of A by 256 and add 24. This is the first byte of the third character. The FOR-NEXT loop reads the codes for the new characters and POKEs the code into the character set. These characters form the flags and the trees.

Line 110 relocates SKI$ from the first position of the string storage area to the player/missile graphics area. Location 54279 is POKEd with the value of A. This is the beginning of the new character set, but because the first 1K of the player missile graphics area is not used by the player/missile graphics, we can use the same address and begin the player right after the new character set. Location 559 is POKEd with 62. The player is single-line resolution. POKE location 53277 with 3 to enable the player/missile graphics. The address of the variable value table is stored in locations 134 and 135. This table contains the location and other information on all string and numeric variables. To relocate SKI$ we need to know where this table is located. The area of memory that stores the string and array information is called the *string storage area*. Its location is stored in memory locations 140 and 141.

Line 100 calculates the offset for the string. The variable value table contains a number called an *offset*. This number is added to the location of the string storage area so the computer knows where a particular string is located. The first string has an offset of 0.

We want to relocate SKI$ to the first player area of the player/missile graphics area. To calculate this offset value multiply the value of A by 256 and add 1024. This is the first byte of the first player. The value of STTB is subtracted from this amount since this is the value that will be added to the offset. This new value is the offset stored in the third and fourth locations of the variable value table. The offset value is divided into two numbers before it is stored. The high-order byte is arrived at by taking the integer of STROFF divided by 256. The low-order byte is the difference between STROFF and the high-order byte. These two values are stored in the third and fourth locations of the variable value table. The fifth location is POKEd with 255. This is the low-order byte of the length of the string. The string is 255 bytes long.

Line 120 places a 0 in the next location, the high-order byte

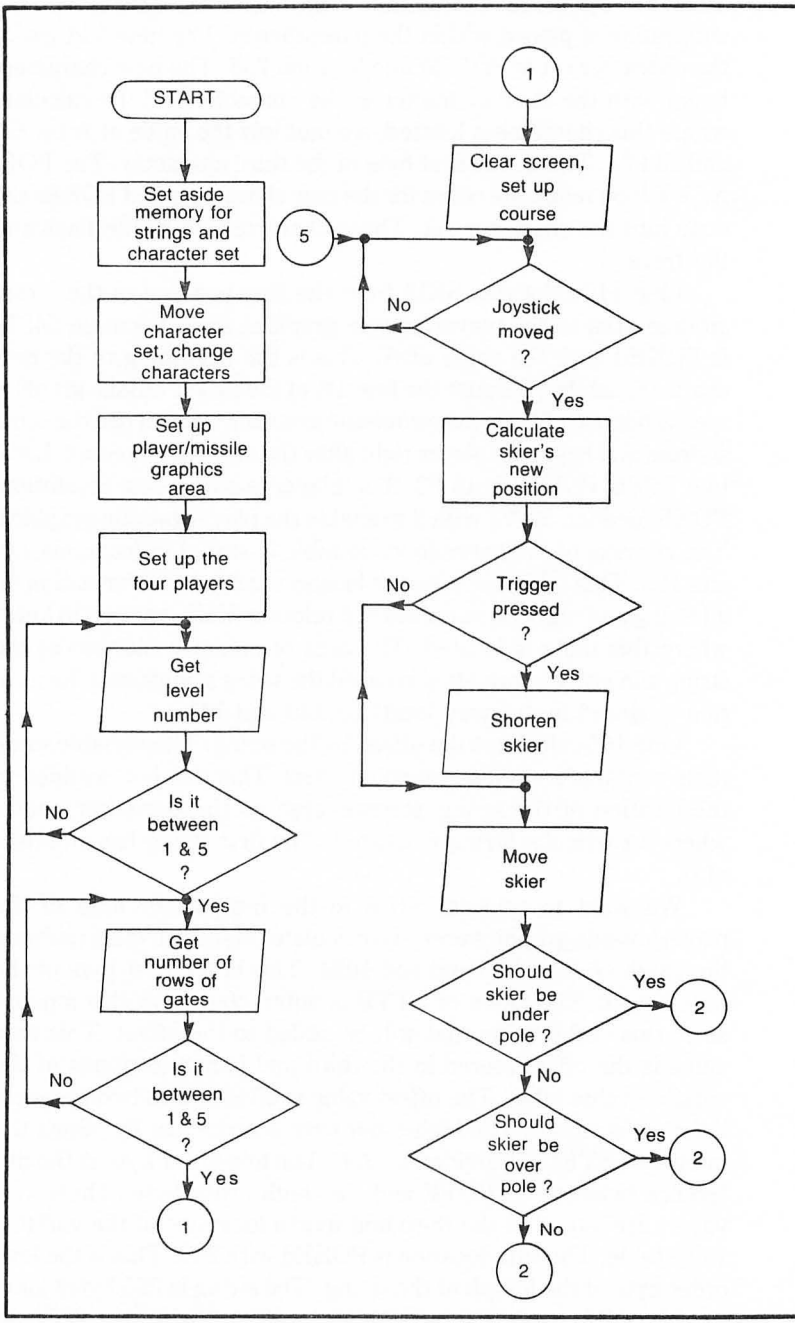
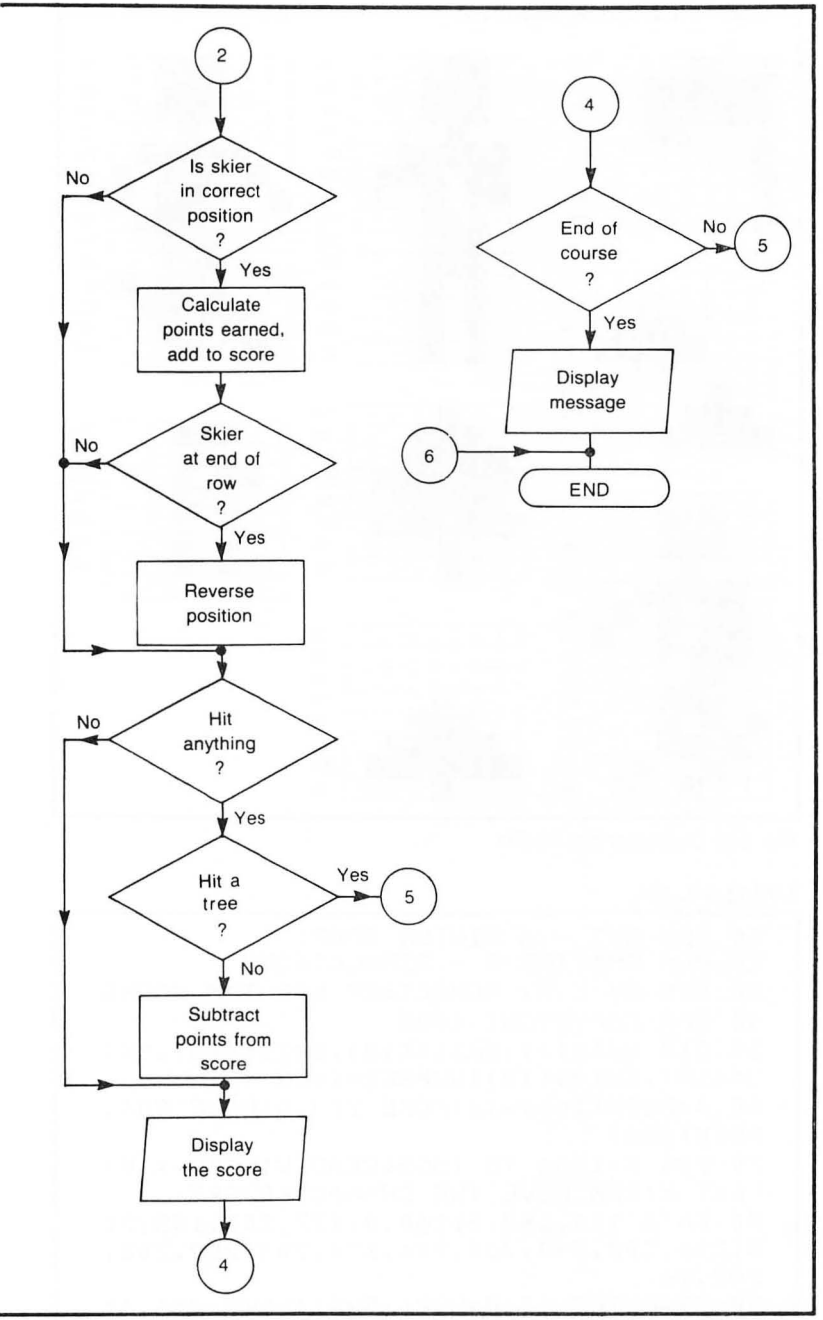


Fig. 5-7. Flowchart for Ski.


2
Is skier in correct position ?
No
Yes
Calculate points earned, add to score
Skier at end of row ?
No
Yes
Reverse position
Hit anything ?
No
Yes
Hit a tree ?
Yes
5
No
Subtract points from score
Display the score
4
4
End of course ?
No
5
Yes
Display message
6
END

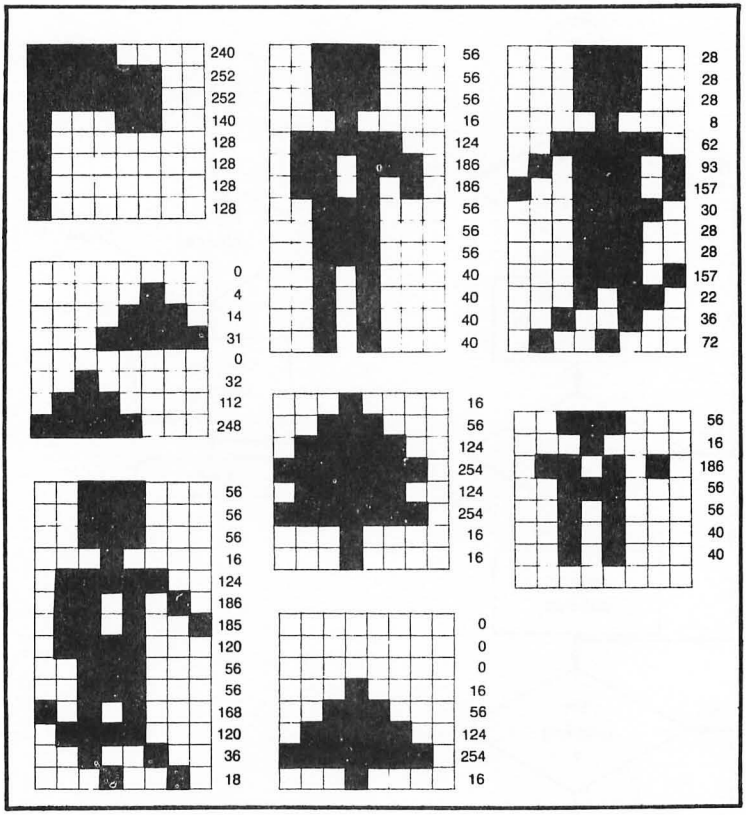


Fig. 5-8. Character set for Ski.

Listing 5-4. Ski.

```
10 REM SKI - A WINTER SPORT
20 REM CHAPTER 5 - SIMULATIONS
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM SKI$(1),SKI1$(18),SKI2$(18),SKI
3$(18),SKI4$(18):NUMBER=1000
60 A=PEEK(106)-16:POKE 204,A:POKE 206,
PEEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256+24:FOR X=CHARSET TO CHAR
```

```
SET+31:READ V:POKE X,V:NEXT X
100 POKE 54279,A:POKE 559,62:POKE 5327
7,3:POKE 53248,0:VT=PEEK(134)+PEEK(135
)*256:STTB=PEEK(140)+PEEK(141)*256
110 STROFF=A*256+1024-STTB:ST3=INT(STR
OFF/256):ST4=STROFF-256*ST3:POKE VT+2,
ST4:POKE VT+3,ST3:POKE VT+4,255
120 POKE VT+5,0:POKE VT+6,255:POKE VT+
7,0:POKE 623,4
130 DATA 240,252,252,140,128,128,128,1
28,16,56,124,254,124,254,16,16,0,4,14,
31,0,32,112,248,0,0,0,16,56,124,254,16
140 FOR X=1 TO 18:READ V:SKI1$(X)=CHR$
(V):NEXT X
150 DATA 0,0,56,56,56,16,124,186,185,1
20,56,56,168,120,36,18,0,0
160 FOR X=1 TO 18:READ V:SKI2$(X)=CHR$
(V):NEXT X
170 DATA 0,0,28,28,28,8,62,93,157,30,2
8,28,157,22,36,72,0,0
180 FOR X=1 TO 18:READ V:SKI3$(X)=CHR$
(V):NEXT X
190 DATA 0,0,56,56,56,16,124,186,186,5
6,56,56,40,40,40,40,0,0
200 FOR X=1 TO 18:READ V:SKI4$(X)=CHR$
(V):NEXT X
210 DATA 0,0,0,0,0,56,16,186,56,56,40,
40,0,0,0,0,0,0
220 ? #6;"":SKI$(1)=CHR$(0):SKI$(255)
=CHR$(0):SKI$(2)=SKI$:POSITION 2,3:? #
6;"LEVEL (1-5)?":GOSUB NUMBER
230 POSITION 15,3:? #6;Z:L=Z:POSITION
2,8:? #6;"GATES (1-5)?":GOSUB NUMBER:P
OSITION 15,8:? #6;Z:G=Z
240 SCORE=0:? #6;"":POKE 704,86:POKE
708,80:POKE 709,198:POKE 710,150:POKE
711,68:POKE 712,8:FOR X=1 TO 6*L
250 R=INT(RND(1)*22)+1:C=INT(RND(1)*20
):LOCATE C,R,O:IF O<>32 THEN 250
260 N=INT(RND(1)*3):POSITION C,R:? #6;
CHR$(4+N):NEXT X
270 FLAG=131:FOR X=1 TO G:FOR Y=1 TO 1
7 STEP 4:POSITION Y,X*4:? #6;CHR$(FLAG
):FLAG=294-FLAG:NEXT Y:NEXT X
280 SP=30:SK=45:SKI$(SP)=SKI3$:POKE 53
248,SK:FLAG=131:SR=1:FF=0:NP=56:DIR=1:
FH=0
```

```
290 POSITION 15,0:? #6;SCORE;:POKE 532
78,0:IF STICK(0)=15 THEN 290
300 IF STICK(0)=7 THEN SK=SK+1:IF STRI
G(0) THEN SKI$(SP)=SKI1$
310 IF STICK(0)=11 THEN SK=SK-1:IF STR
IG(0) THEN SKI$(SP)=SKI2$
320 IF STICK(0)=13 THEN SP=SP+1:IF STR
IG(0) THEN SKI$(SP)=SKI3$
330 IF STICK(0)=14 THEN SP=SP-1:IF STR
IG(0) THEN SKI$(SP)=SKI3$
340 IF STICK(0)=6 THEN SP=SP-1:SK=SK+1
:IF STRIG(0) THEN SKI$(SP)=SKI1$
350 IF STICK(0)=5 THEN SP=SP+1:SK=SK+1
:IF STRIG(0) THEN SKI$(SP)=SKI1$
360 IF STICK(0)=10 THEN SP=SP-1:SK=SK-
1:IF STRIG(0) THEN SKI$(SP)=SKI2$
370 IF STICK(0)=9 THEN SP=SP+1:SK=SK-1
:IF STRIG(0) THEN SKI$(SP)=SKI2$
380 IF SK>205 THEN SK=205
390 IF SK<35 THEN SK=35
395 IF SP<20 THEN SP=20
396 IF SP>229 THEN SP=229
400 IF SP<20 THEN SP=20
410 IF SP>229 THEN SP=229
420 POKE 53248,SK:IF STRIG(0)=0 THEN S
KI$(SP)=SKI4$
430 IF SK=NP THEN GOSUB 500
440 HIT=PEEK(53252):IF HIT=2 THEN 480
450 IF HIT<>0 AND FH=0 THEN SOUND 0,10
,10,10:FOR TM=1 TO 10:NEXT TM:SOUND 0,
0,0,0:FH=1
460 IF HIT=0 AND FH=1 THEN SCORE=SCORE
-(SR*10):FH=0:POSITION 15,0:? #6;"
":IF SCORE<0 THEN SCORE=0
470 GOTO 290
480 SOUND 0,10,8,10:FOR TM=1 TO 10:NEX
T TM:SOUND 0,0,0,0:GOTO 610
500 FP=FP+1:IF FLAG=131 AND SP>30+SR*3
3 AND SP<30+(SR+1)*33 THEN 530
510 IF FLAG=163 AND SP<30+SR*33 AND SP
>30+(SR-1)*33 THEN 530
520 SOUND 0,200,8,10:FOR TM=1 TO 10:NE
XT TM:SOUND 0,0,0,0:GOTO 540
530 SOUND 0,50,10,10:FOR TM=1 TO 10:NE
XT TM:SCORE=SCORE+SR*10:SOUND 0,0,0,0:
POSITION 15,0:? #6;"    "
```

```
540 IF FP/5=INT(FP/5) THEN IF SR=G THE
N GOTO 600
550 IF FP/5=INT(FP/5) THEN SR=SR+1:NP=
NP+(31*DIR):DIR=DIR*-1
560 NP=NP+(31*DIR):FLAG=294-FLAG:RETUR
N
600 POSITION 5,10:? #6;"YOU DID IT!"
610 POSITION 5,12:? #6;"PRESS START":R
EM START IS INVERSE
620 IF PEEK(53279)<>6 THEN 620
630 GOTO 220
1000 OPEN #2,4,0,"K:":GET #2,Z:CLOSE #
2
1010 IF Z>127 THEN Z=Z-128
1020 Z=Z-48:IF Z<1 OR Z>5 THEN GOTO NU
MBER
1030 RETURN
```

Line 90 sets the screen for graphics 17. This is graphics 2 with no text window. The USR command is used to execute the of the length of the string. The next two locations are set the DIM value. We want to change the values of these locations to 255 and zero. POKE location 623 with 4 to set the priorities of the player.

Line 130 contains the code for four characters.

Lines 140-210 place the code for the four skiers into four different strings. These characters are placed into the player string, depending on which direction the skier is going in and whether the first button has been pressed.

Line 220 clears the screen, then clears SKI$ by placing the character 0 in the first and last elements of the string and setting the second element of the string to the string. The computer asks for a level number. The subroutine at line 1000 is used to get the input.

Line 230 prints the level number after the question mark and stores this number in variable L. The computer asks for the number of gates. The subroutine at line 1000 is used again. When the computer returns to this line, it prints the number after the question mark and stores this value in variable G.

Line 240 clears the SCORE variable and the screen. The colors for the player, the flags, the trees, and the screen are set. The FOR-NEXT loop begins here. The value of L determines how many trees are printed on the screen.

Line 250 selects a random row and column for the tree. The

Locate command checks to see if that position is empty. If it is not, the line is repeated until the computer chooses an empty location.

Line 260 chooses a number from 1 to 3. This number determines which tree will be printed on the screen. The loop continues until all the trees are placed on the screen.

Line 270 prints the rows of flags on the screen. Variable FLAG is set to 294. These are the red flags. The number of rows the FOR-NEXT loop prints is set by the value of variable G. The flags are printed four columns apart. After a flag has been printed, the value of FLAG is subtracted from 294. Now FLAG contains the value for a blue flag. The loops continue until the correct number of flags are printed on the screen.

Line 280 sets the variables used in the skiing routine. The SP variable is the position of the skier in SKI$. SK is the column position of the skier. The third skier character is placed in SKI$. Location 53248 is POKEd with the column position. The skier is the first player of the player/missile graphics. These commands place the skier on the screen. The value of flag is set to131, the red flag. The SR variable keeps track of which row the skier is moving through. The score for each flag passed is increased after each row is completed.

The FP variable is the flag the skier has passed. The computer needs to know whether this is the first or last flag of the row. NP is set to 56. This is the column position the skier must cross to score points for each flag. The direction variable, DIR, is set to 1. When DIR is 1, the skier is moving from left to right. When DIR is −1 the skier is moving from right to left. The FH variable is set when the skier hits a flag. Points are subtracted from the score when a flag is hit.

Lines 290-420 maneuver the skier on the screen. The score is printed on the screen in the upper right corner. The hit register is cleared. If the joystick is not moved, the line loops. The computer proceeds to the rest of the routine when the joystick has been moved. The skier can move in any of the eight joystick directions. The column variable, SK, is increased or decreased depending on which way the joystick is moved. The row variable, SP, is also increased or decreased when the joystick is pushed down or up. If the joystick is pushed on the diagonal, both the row and column variables change.

The correct ski character is moved immediately into the SKI$. This can be any one of the three skiers, depending on which direction the skier is moving. The position of the skier is checked to

make sure he doesn't ski off the screen. The column of the skier is POKEd into location 53248. The fire button on the joystick is checked. If it is pressed, the fourth skier is placed on the screen. This is the smaller version of the skier. He can get through the trees that sometimes surround the flags.

Line 430 checks to see if the skier is in the same column as the flag he is trying to ski around. If he is, the computer uses the subroutine in line 500 to make sure the skier is going around the flag the right way, and adds the correct number of points to the score.

Line 440 checks to see if the skier has hit something. The value of memory location 53252 is stored in variable HIT. If the skier has not hit a flag or tree, this value is 0. If something has been hit, the value is different. If the value of HIT is 2, the skier has hit a tree and the computer is directed to the lines that will end the game.

Line 450 checks to see if the value of HIT is something other than 0. If it is, and the value of FH is 0, the computer makes a sound indicating that the flag was hit and set the value of FH to 1.

Line 460 subtracts points from the score when the flag is hit. The score cannot fall below 0.

Line 470 sends the computer back to line 290. This routine continues until the skier passes the last flag on the course.

Line 480 is the routine the computer uses when a tree is hit. The computer makes a sound, then is sent to line 610, which ends the program.

Line 500 is the subroutine that checks to see if the skier has gone around the pole correctly. The value of FP is increased by 1. This is the number of the flag the skier went around. The computer checks to see if the position of the skier is correct in relationship to the flag passed. If the color of the flag is red, the skier's position value must be greater than that row value, but less than the next row's value. If both conditions are met, the computer goes to line 530 to score the points.

Line 510 checks to see if the skier went above the blue flag's row, but stayed under the row of flags in the previous row. Again, if the skier went around the flag the correct way, the computer will go to line 530 to score the points.

Line 520 makes the sound that tells the player the skier went around the flag in the wrong direction. The computer goes to line 540 to calculate where the next flag is.

Line 530 makes the sound for a skier who went around the flag

correctly. The score is increased by the value of SR times 10. Each time one row is completed, the scores for the next row are increased by 10 points.

Line 540 checks to see if this was the last flag of the row. If it is, the computer checks to see if this was the last row. If it was, the computer is sent to line 600 to end the game.

Line 550 checks again to see if this is the last flag of the row. If it is, the value of SR is increased by 1, and the direction is reversed. The statement to find the position of the next flag must be included in this line, even though it is repeated in the next line, because the computer uses the value of NP to calculate the next position. By including it in this line the computer calculates the position of an imaginary flag. This position is used to calculate the correct position of the first flag of the next row.

Line 560 calculates the position of the next flag in that row. The value of FLAG is changed to the other color and the computer returns to the line that called this subroutine.

Line 600 is used when the skier completes the entire course.

Line 610 tells the player to press the Start key to play another game.

Lines 620-630 loop until the Start key is pressed. The computer is sent to line 220 to play the game again. If you want to end the game, press the System Reset key.

Lines 1000-1030 are the subroutine used in the beginning of the program to get the one-digit entry. The keyboard is opened and the computer waits until a key is pressed. The value of this key is placed in variable Z and the keyboard is closed. If the value of Z is greater than 127, 128 is subtracted from it. Then 48 is subtracted from the value of Z. The value of zero is 48, so this gives us the number of the pressed key. If the key was less than 1 or greater than 5, the computer goes back to line 1000 and waits for another key to be pressed. If a number key between 1 and 5 was pressed, the computer returns to the line that called this subroutine.

PINBALL

Objective of the game: To keep the ball in play as long as possible and to score as many points as possible.

Directions: Two joysticks are needed for this program. Plug one joystick into the first joystick port and the second into the second joystick port. To release the ball, press the first button on the first joystick. The ball will travel up the side of the playfield, across the top of the screen, then drop through one of the openings at the

top of the screen. Press the fire buttons on the joysticks to move the flippers on the bottom of the screen. You can control the direction of the ball somewhat by moving the joystick. Points are scored each time the ball hits one of the characters on the field. Figure 5-9 is the flowchart, and Fig. 5-10 is the character set for this program, and Listing 5-5 is the code.

Line 50 sets aside the memory for the strings. B$ must be the first variable used in this program. The program will relocate the memory storage area for this string. TEMP$ stores the code for the ball.

Line 60 finds out how much memory is in the computer. The character set is placed 1K above the screen memory. This amount is stored in variable A. This value is also stored in memory location 204. The beginning of the ROM-based character set is stored in memory location 206.

Line 70 places the code for the assembly language subroutine in memory locations 1536-1555. Line 80 contains the code for this routine.

Line 90 changes the screen to graphics mode 17. This is graphics mode 1 with no text window. The USR command tells the computer to go to the subroutine at location 1536. When the computer returns to this line, the value of A is POKEd into location 756. This is the beginning of the character set in RAM. If your screen contains garbage at this point the wrong value was placed in this location. If graphics characters do not appear on the screen when the game begins, the value was POKEd into the wrong location. The CHARSET variable is set to the first byte of the fourth character. This is the first character to be changed. Eleven characters in the RAM character set are changed. The FOR-NEXT loop reads the new character codes and POKEs them into these locations.

Line 100 POKEs the value of A into memory location 54279. This is the beginning of the character set, but it is also the address used for the player/missile graphics. The first player actually begins 1048 bytes after this address, so the RAM-based character set fits into memory just before the player/missile graphics area. Location 559 is POKEd with 62 to tell the computer to use single-line resolution for the players.

To enable the player/missile graphics, POKE location 53277 with 3. The address of the variable value table is placed in variable VT. This table contains information for all the variables and strings. We will change this information when we relocate B$. The area

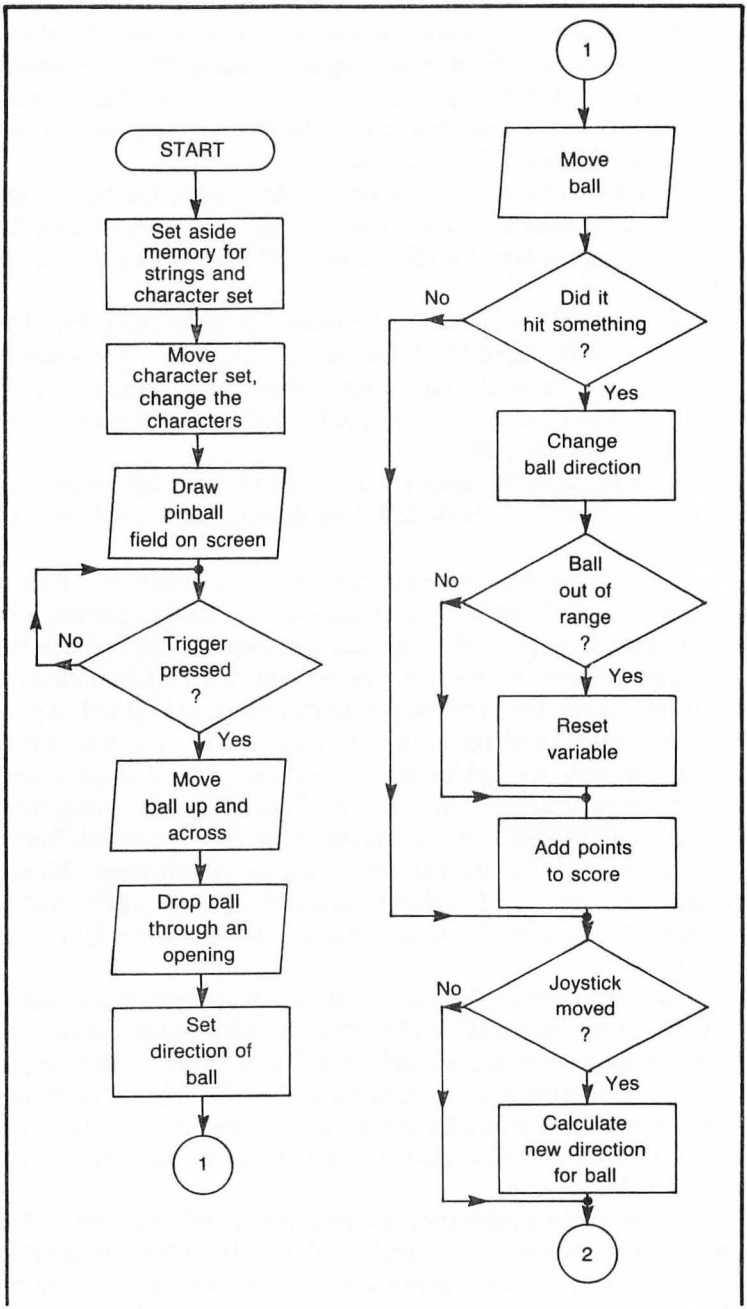
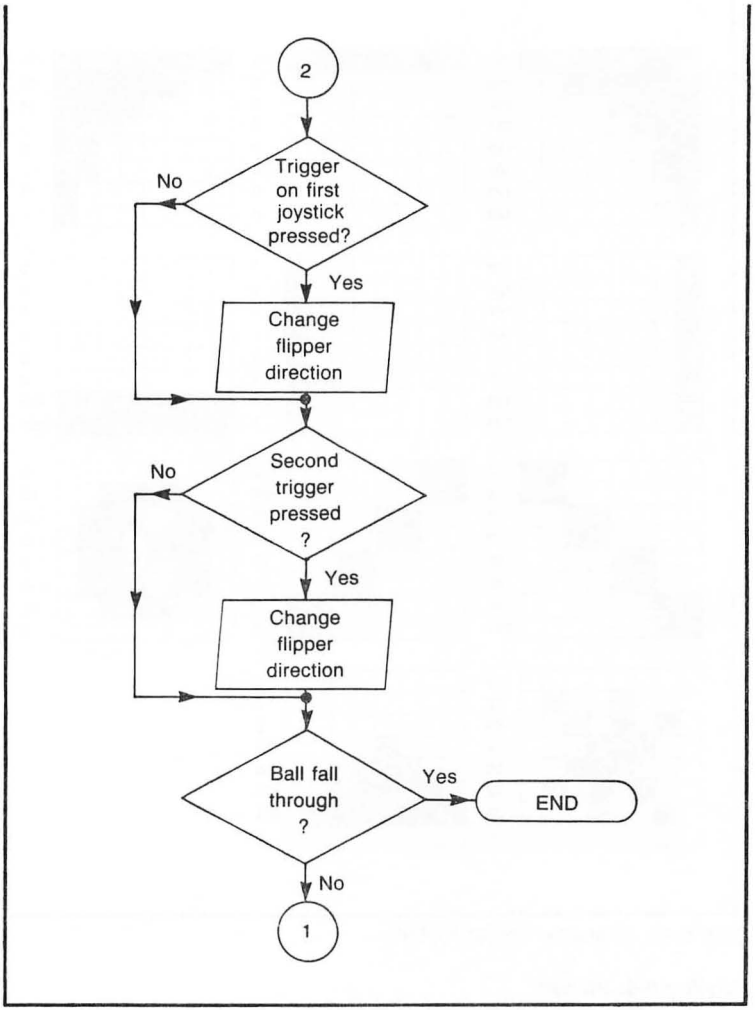


Fig. 5-9. Flowchart for Pinball.



of memory where information for the strings and arrays is stored is called the *string storage area*. Its address is stored in locations 140 and 141. This address is placed in variable STTB.

Line 110 calculates the offset for the new storage area for B$. The first player area is arrived at by multiplying the value of A by 256 and adding 1024. From this number we subtract the value of STTB. When the computer looks for the area of memory where B$ is stored it adds this value to the offset value. We subtract it now so that when it is added back the string is located at the area set aside for the first player. The value is divided into a two-byte

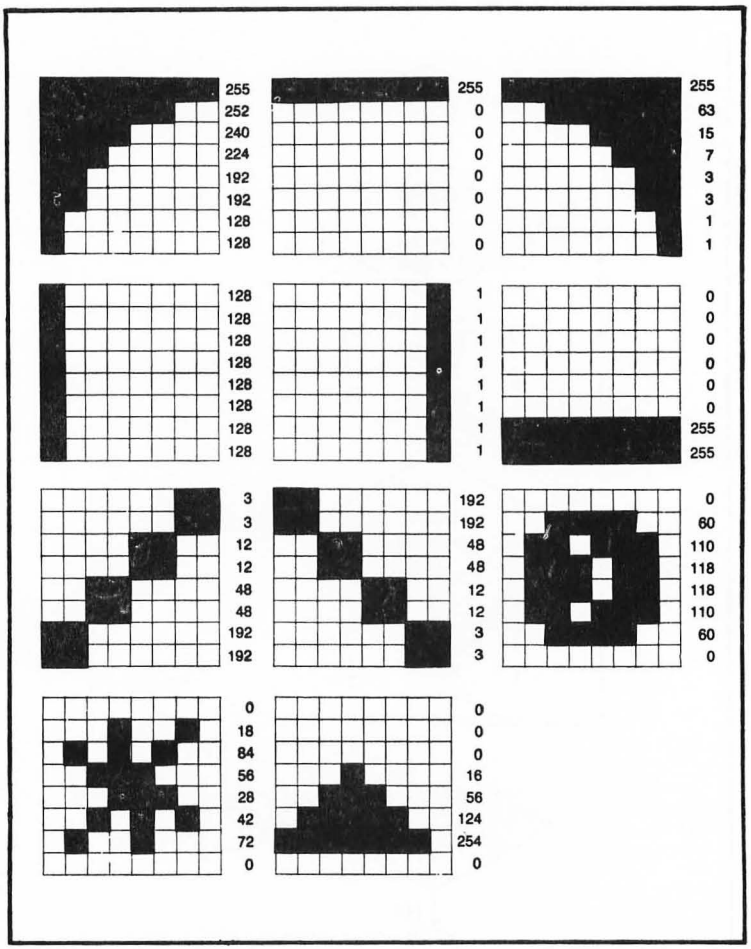


Fig. 5-10. Character set for Pinball.

Listing 5-5. Pinball.

```
10 REM PINBALL - A GAME OF SKILL
20 REM CHAPTER 5 - SIMULATIONS
30 REM BY L.M. SCHREIBER FOR TAB BOOKS
40 REM COPYRIGHT 1983
50 DIM B$(1),TEMP$(8)
60 A=PEEK(106)-16:POKE 204,A:POKE 206,
PEEK(756)
70 FOR X=1536 TO 1555:READ V:POKE X,V:
NEXT X:REM MOVE THE CHARACTER SET
80 DATA 104,162,4,160,0,177,205,145,20
```

```
3,200,208,249,230,206,230,204,202,208,
242,96
90 GRAPHICS 17:Q=USR(1536):POKE 756,A:
CHARSET=A*256+32:FOR X=CHARSET TO CHAR
SET+87:READ V:POKE X,V:NEXT X
100 POKE 54279,A:POKE 559,62:POKE 5327
7,3:POKE 53248,0:VT=PEEK(134)+PEEK(135
)*256:STTB=PEEK(140)+PEEK(141)*256
110 STROFF=A*256+1024-STTB:ST3=INT(STR
OFF/256):ST4=STROFF-256*ST3:POKE VT+2,
ST4:POKE VT+3,ST3:POKE VT+4,255
120 POKE VT+5,0:POKE VT+6,255:POKE VT+
7,0:B$(1)=CHR$(0):B$(255)=CHR$(0):B$(2
)=B$
130 DATA 255,252,240,224,192,192,128,1
28,255,0,0,0,0,0,0,0,255,63,15,7,3,3,1
,1,128,128,128,128,128,128,128,128
140 DATA 1,1,1,1,1,1,1,1,0,0,0,0,0,0,2
55,255,3,3,12,12,48,48,192,192,192,192
,48,48,12,12,3,3
150 DATA 0,60,110,118,118,110,60,0,0,1
8,84,56,28,42,72,0,0,0,0,16,56,124,254
,0
170 POSITION 0,1:? #6;"$%% %% %% %%&
SCORE":COLOR 39:PLOT 0,2:DRAWTO 0,21:C
OLOR 40:PLOT 12,2:DRAWTO 12,21
180 PLOT 13,1:DRAWTO 13,21:COLOR 41:PL
OT 1,21:DRAWTO 11,21:COLOR 172:PLOT 3,
3:PLOT 6,3:PLOT 9,3:PLOT 1,12
190 PLOT 11,12:COLOR 13:PLOT 2,7:PLOT
10,7:PLOT 5,10:PLOT 7,10:COLOR 142:PLO
T 3,19:PLOT 9,19:PLOT 6,15
195 POSITION 6,21:? #6;" "
200 POKE 704,56:RESTORE 150:FOR X=1 TO
 8:READ V:B$(200+X)=CHR$(V):TEMP$(X)=C
HR$(V):NEXT X:C=152:POKE 53248,C
210 IF STRIG(0)=1 THEN 210
220 FOR X=200 TO 30 STEP -1:B$(X)=TEMP
$:NEXT X:R=30:C1=INT(RND(1)*3):C2=120:
IF C1=1 THEN C2=96
230 IF C1=2 THEN C2=72
240 FOR X=C TO C2 STEP -1:POKE 53248,X
:NEXT X:C=X+1:FOR X=R TO R+12:B$(X)=TE
MP$:NEXT X:R=X-1
250 DC=INT(RND(1)*3)-1:DR=1
260 POKE 53278,0:R=R+DR:C=C+DC:IF R<42
 THEN R=R+DR*-1
```

```
270 B$(R)=TEMP$:POKE 53248,C:IF PEEK(5
3252)=0 THEN 310
280 Q=PEEK(53252):DR=DR*-1:DC=DC*-1:FO
R X=1 TO 5:R=R+DR:C=C+DC:B$(R)=TEMP$:P
OKE 53248,C:NEXT X
285 IF C<47 THEN C=47:POKE 53248,C:REM
 KEEP THE BALL ON THE FIELD
286 IF C>145 THEN C=145:POKE 53248,C
290 IF Q>1 THEN SCORE=SCORE+Q*10:FOR X
=1 TO 10:POKE 712,INT(RND(1)*15):SOUND
 0,INT(RND(1)*100)+100,10,10:NEXT X
300 POKE 712,0:SOUND 0,0,0,0:DR=INT(RN
D(1)*3)-1:DC=INT(RND(1)*3)-1:IF DR=0 A
ND DC=0 THEN 300
310 POKE 77,0:ST=STICK(0):IF STICK(0)=
15 THEN 360
320 IF ST=7 THEN DC=1
330 IF ST=11 THEN DC=-1
340 IF ST=14 THEN DR=-1
350 IF ST=13 THEN DR=1
360 Q=0:IF STRIG(0)=0 THEN LOCATE 5,21
,Q:POSITION 5,21:IF Q=41 THEN ? #6;"*
":FOR X=1 TO 50:NEXT X
370 IF Q=42 THEN ? #6;") ":FOR X=1 TO
50:NEXT X
380 IF STRIG(1)=0 THEN LOCATE 7,21,Q:P
OSITION 7,21:IF Q=41 THEN ? #6;"+":FOR
 X=1 TO 50:NEXT X
390 IF Q=43 THEN ? #6;")":FOR X=1 TO 5
0:NEXT X
400 POSITION 15,3:? #6;SCORE:IF R<202
AND SCORE<100000 THEN 260
410 POSITION 15,7:? #6;"game":POSITION
 16,9:? #6;"over":REM over IS INVERSE
420 IF PEEK(53279)<>6 THEN 420
430 ? #6;CHR$(125):SCORE=0:GOTO 170
```

number and stored in the third and fourth locations of the variable value table. The value 255 is placed in the fifth location. This is the length of the string.

Line 120 POKEs 0 into the next location. That is the high-order byte of the string length. The next two locations are POKEd with 255 and 0. B$ is dimensioned to 255.

Lines 130-150 contain the code for the new characters.

Lines 170-195 print the top of the pinball field and the word "SCORE" on the screen. The PLOT and DRAWTO commands place the rest of the pinball outline on the screen.

Line 200 changes the color of the first player. The pointer is reset to line 150. This is the line where the computer will read DATA when a READ command is used again. The FOR-NEXT loop reads the code in line 150 and places it in B$ and TEMP$. This is the code for the ball. By placing it in B$, it is put directly the player/missile area. TEMP$ also contains the code for the ball. When the ball is moved in B$, it is replaced with the code from TEMP$. Variable C is set to 152. This is the column in which the ball in the player/missile area will be printed. The ball appears on the screen when the value of C is POKEd into location 53248.

Line 210 loops until the fire button is pressed.

Line 220 moves the ball up the screen. The FOR-NEXT loop starts with the ball near the bottom of the screen. The loop steps backwards, raising the ball one row every time the loop is completed. Variable R is set to 30. This is the row the ball is in. Variable C1 is set to a random number between 0 and 2. This determines which opening the ball falls through. Variable C2 is set to 120. This is the third opening. If C1 is 0, the ball falls through this opening. If C1 is 1,the ball falls through the middle opening.

Line 230 checks to see if C1 is 2. If it is, the ball will fall through the first opening.

Line 240 moves the ball across the screen. The ball begins in the column it is in, and moves to the right to the column of the opening it will fall through. As variable X changes, the value of X is POKEd into location 53248. Variable C is reset to the column the ball is in, and the following FOR-NEXT loop drops the ball through the opening. Variable R is set to the row the ball is in.

Line 250 sets variable DC to a random number from -1 to $+1$. This number determines which direction the ball moves. Variable DR is set to 1.

Line 260 clears the hit register by POKEing location 53278 with 0. Variables R and C are changed. If the value of R is less than 42, the direction of the ball is reversed. This way the ball cannot go out the openings at the top of the screen.

Line 270 places the ball in the new location in B$. The new column value is POKEd into location 53248. Location 53252 is checked for a zero value. If it is, the ball has not hit anything and the computer is directed to line 310.

Line 280 places the value of location 53252 in variable Q. The

object the ball hit is determined by the value of Q. The direction of the ball is reversed by multiplying the values of DR and DC by −1. The FOR-NEXT loop moves the ball away from the object hit. The ball is placed in B$ at the new row position, and the new value of C is POKEd into location 53248. The loop continues until the ball has been moved about five rows and columns.

Line 285 checks the value of the column variable. If it is less than 47, the variable is reset to 47 and this value is POKEd into location 53248. This keeps the ball on the pinball field.

Line 286 checks the value of C to see if it is greater than 145. If it is, C is reset to 145. This keeps the ball from going into the scoring area of the screen.

Line 290 checks the value of Q. If Q is 1, the boundary of the pinball area has been hit and no points are awarded. If it is greater than 1, the value of Q times 10 is awarded the player. The screen changes colors and the computer makes a sound when the ball hits an object.

Line 300 resets the screen color and turns the sound off. A new random number between −1 to +1 is picked for the direction of the row and column. If both the row and column direction are 0, the line repeats itself. This way the ball doesn't sit in the middle of the screen and do nothing.

Line 310 disables the ATTRACT mode so the screen doesn't change colors while you are playing the game. The value of the joystick is placed in variable ST. If the joystick hasn't moved, the value is 15 and the computer goes to line 360.

Lines 320-350 change the direction of the ball based on the position of the joystick.

Line 360 clears variable Q. If the first button on the first joystick has been pressed, the computer gets the value of the character that forms the paddle. If its value is 41, the computer prints the other character on the screen. The timing loop gives the player a chance to release the fire button.

Line 370 checks the value of Q also. If it is 42, character 41 is printed on the screen. This way the other position of the paddle is always printed on the screen.

Line 380 checks to see if the first button on the second joystick has been pressed. If it has, the computer uses the LOCATE command to get the value of the character in the second paddle's position. If the value of Q is 41, the character whose value is 43 is printed on the screen.

Line 390 checks to see if the character's value is 43. If it is,

the character whose value is 41 is printed on the screen.

Line 400 prints the player's score on the screen. If the value of the row variable is less than 202 and the score is less than 1000, the game continues on line 260. If the ball is in a row whose value is greater than 202, the ball has fallen out of the pinball area and the game is over. The game is also over when a player reaches the score of 100,000 or more.

Lines 410-430 end the game. The message "GAME OVER" is printed on the screen. The value of 53279 is checked. The computer loops until the Start key is pressed. The screen clears and the score is reset to zero when the Start key is pressed. The computer goes to line 170 to play the game again. The game can be ended at any time by pressing the System Reset key.

ATARI® Fun and Games

If you are intrigued with the possibilities of the programs included in *ATARI® Fun and Games* (TAB Book No. 1945), you should definitely consider having the ready-to-run disk containing the software applications. This software is guaranteed free of manufacturer's defects. (If you have any problems, return the disk within 30 days, and we'll send you a new one.) Not only will you save the time and effort of typing the programs, the disk eliminates the possibility of errors that can prevent the programs from functioning. Interested?

Available on disk for ATARI® 400, 600, 800 and 1200 systems (including XE and XL), 32K, at $19.95 for each disk plus $1.00 shipping and handling.

I'm interested. Send me:

_____ disk for ATARI® 400, 600, 800, or 1200 systems, 32K (6324S)

_____ TAB BOOKS catalog

_____ Check/Money Order enclosed for $19.95 plus $1.00 shipping and handling for each disk ordered.

_____ VISA _____ MasterCard

Account No. _____ Expires _____

Name _____

Address _____

City _____ State _____ Zip _____

Signature _____

Mail To: **TAB BOOKS Inc.**
P.O. Box 40
Blue Ridge Summit, PA 17214

(Pa. add 6% sales tax. Orders outside U.S. must be prepaid with international money orders in U.S. dollars.)

TAB 1945

ATARI® Fun and Games

by Linda M. Schreiber

If you're in the market for some new and different games to play on your ATARI microcomputer . . . but don't want to shell out big bucks for an expensive disk or tape cartridge . . . this book is just what you've been looking for! Featuring a collection of more than 20 different action-packed game programs that run on any model ATARI with at least 16K of RAM, this sourcebook includes both new adaptations of traditional board and card games and games written specifically for computer play . . . something to please and challenge all ages and experience levels.

These ready-to-run fun and game programs are designed to make the most of your ATARI's special sound, graphics, and color capabilities and include: games for kids like Go Fish, Marbles, and Jacks; action simulations such as Ski, Battleship, and Pinball; card games including Blackjack and Old Maid; grid games like Treasure Hunt and The Great Abyss . . . plus word games (Robotman and Fractured Stories are two examples) and logic games such as Pebbles and Tower of Hanoi. Several games are especially designed to use the ATARI's unique player/missile graphics, movable character sets and interrupts.

And that's not all! If you'd like to try your hand at writing your own original game programs, this is your source for documentation techniques and other game-writing devices ranging from the use of PEEK and POKE and using machine calls to some amazingly sophisticated string memory applications. Each game includes line-by-line explanation as well as flowchart and character set so you'll be able to thoroughly understand methods used in game development . . . particularly valuable to novice programmers who want to broaden their programming horizons or for those who want to customize the games to suit their own particular interests.

Exceptionally well written and documented, this is a programming guide that will keep you and your family amused for hours on end . . . and improve your programming skills at the same time.

Linda M. Schreiber is the author of several bestselling computer books for TAB including *ATARI Programming . . . with 55 Programs*.

TAB TAB BOOKS Inc.
Blue Ridge Summit, Pa. 17214

FPT > $10.95 ISBN 0-8306-1945-3

PRICES HIGHER IN CANADA 1045-1185