TEXAS INSTRUMENTS
HOME COMPUTER

GAME WRITERS' PACK 2

CASSETTE SOFTWARE
WITH MANUAL

An integrated pack containing a series of programs on cassette that develop and graphically display major ideas covered in the accompanying book. Enables any user to progressively understand and make full use of this computer.

COLLINS
MICROSOFTWARE

TEXAS INSTRUMENTS
HOME COMPUTER

Game Writers' Pack 2

PK McBride

COLLINS
MICROSOFTWARE

1103216-0000

1 2 3 4 5 6 7 8 9

Produced and printed by Contract Books Ltd,

Contents

Introduction

In Game Writer's Pack 1, the computer was used to organize the games, and served as the board on which the games were played, but it took no really active part. Most of games covered here are those where the 99 acts as a player. Sometimes its play is automatic, but in most games, the 99 has to think! Of course, computers cannot think for themselves – at least not yet – so it's up to you, the programmer, to teach it how. This means that *you* have to think hard about how a game works, and how a human player takes decisions during a game. It takes time, but it's worth it in the end. You will find that you learn a tremendous amount about computing techniques while you are working out your own games.

The programs in this pack are mainly examples of the sorts of games you can write using the techniques covered in the book. Please feel free to extend or alter these programs to make them into games of your own. As long as you do not try to record over the original programs on the cassette, then, whatever you try, the programs will not be lost or damaged. The program LISTs are given in the Appendix for your reference.

The book assumes that you have read the two Starter Packs, and Games Writer's Pack 1, and that you have a reasonable grasp of the computing techniques covered in those. It also assumes that you will work through chapter by chapter and learn to use each new idea before moving on.

You don't need any special equipment for games programming, just a T.V., a cassette recorder, your trusty 99, lots of paper and plenty of ideas. A printer is of tremendous help in sorting out long programs, and the TI EXTENDED BASIC cartridge can make life easier, but neither are essential.

1
Word games

Word games rely on the comparison of strings or parts of strings. HANGMAN, included in Starter Pack 2, is a useful demonstration of the use of string-slicing techniques in games.

The game follows the normal rules. Correctly guessed letters are written on the dashes to build up the mystery word. Bad guesses are recorded, and the hanging man is drawn in ten stages.

The essential routines of the game are very simple. The major complications lie in the screen presentation. Using standard TI BASIC, it is not possible to PRINT at set points on the screen, and so the HCHAR sub-program has to be used. (In TI EXTENDED BASIC you have a DISPLAY AT command which makes screen presentations that much easier.)

The heart of the program is the comparison of the guessed letter with each of the letters of the word in turn. In EXTENDED BASIC it would look something like this (L$ is the Letter, W$ is the Word):

```
FOR T = 1 TO LEN(W$)
IF L$ = SEG$(W$,T,1) THEN....ELSE....
...
...
NEXT T
```

In TI BASIC it's not that easy. If you allow a letter to be INPUT, then your screen will scroll up and you will lose your display. The Letter must be collected by a CALL KEY line. You now have a code rather than a letter, so the codes of the letters of the Word need to be checked. That check line now looks like this:

```
IF K = ASC(SEG$(W$,T,1)) THEN....
```

K is the code of the letter collected by CALL KEY(3,K,S). The alternative is to transfer the letter code to a string: L$ = CHR$(K) and carry on as before.

Either way you do it, the result is the same. If the word was "HANGMAN" and the letter was "A" then this is what happens:

```
HANGMAN
  \  /
   \/
GOTO "found" routine.
```

If you look at the HANGMAN LIST in the Appendix , you will find this part of the program around lines 600 onwards.

The found routine is at 1000. The first thing to do is to print the found letter in the right place on the screen.

```
CALL HCHAR(15,T*2+8,K)
```

This prints the letter, CHR$(K), on the 15th row, starting 8 columns in, and spaced out with a single space between the letters. From the example above, the first "A" would appear at 15,12 (=2*2+8) and the second "A" at 15,20.

You could at this stage simply mark up a correct guess, (CG=CG+1) and go on, checking to see if you have as many guesses as there are letters in the word (IF CG = LEN(W$) THEN...). Unfortunately, some players cheat. There would be nothing here to stop someone using the same correct letter all the time, and building up his score that way. This is why lines 1030 to 1070 are there. They remove the guessed letter from the word and replace it with a space.

```
1030  P=POS(W$,CHR$(K),1)      (where's the letter
                                again?)
1040  L$=SEG$(W$,1,P-1)        (left-hand side up to
                                the letter)
1050  R$=SEG$(W$,P+1,L-P)      (right-hand side - L
                                is Len(W$))
1060  W$=L$&" "                (add a space)
1070  W$=W$&R$
```


Back to the example – on the first "A" this happens:

```
P=2
L$ = "H" (SEG$(W$,1,1))
R$ = "NGMAN" (SEG$(W$,3,4))
```

W$ is redefined to read "H NGMAN"

The same technique can be used, by the way, to knock a single space invader out of a string of them.

You can now add to your Correct Guesses score, and check for enough Correct Guesses. The actual program uses a different check routine. There, a Check String is created (C$) which is the same length as the Word, but filled entirely with spaces. When the letters of the word have all been replaced with spaces, then this is picked up by comparison with C$. (Lines 1090 to 1120).

Bad Guesses

You need some sort of flag in your "found" routine to show that the computer has been there. In the program it is Z. If at the end of the loop, Z is still zero, then clearly no letter has been found, and it was a Bad Guess. The computer now goes off to the drawing routine. M is the counter for the number of Misses. Look at lines 2000 onwards.

```
2000  M=M+1
2010  ON M GOSUB.....
```

Each sub-routine draws another part of the picture.

The Bad Guesses must also be recorded on the screen, and their position is held in GR (Guess Row) and GC.

```
2040  CALL HCHAR(GR,GC,K)
```

prints up the letter. GR and GC are then adjusted ready for the next Bad Guess.

Figure 1 shows the flowchart for the basic HANGMAN game.

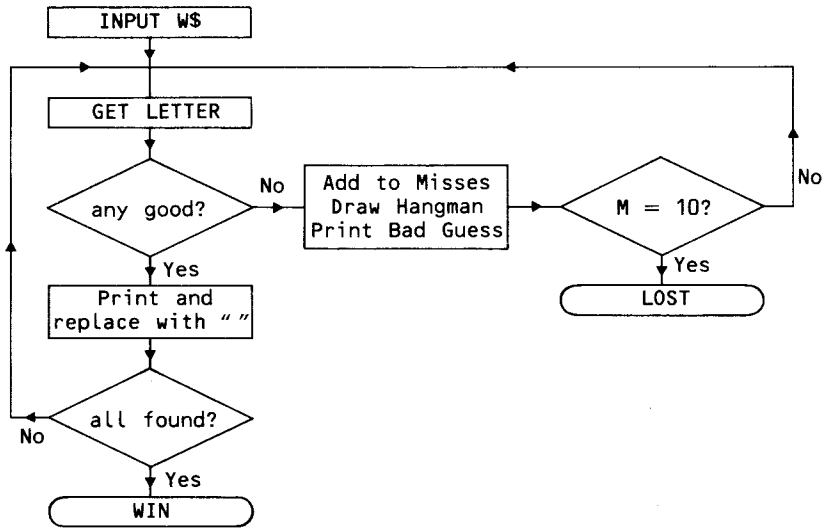


Figure 1

Here the word is entered by a second player at the start of the game.

Word Banks for Hangman

Two separate word banks are included in the HANGMAN game in Pack 2 – one of BASIC words, the other of animal names. You can very readily add extra banks, or extend these. To add extra banks simply type the words in as DATA lines starting at 8000, 9000, 10000 (or any other well-spaced numbers). Next alter the selection routine that starts at 180. You want to be able to fix the RESTORE position so that the 99 begins to read at the start of the chosen set of DATA lines. It is probably easiest to ask the player for a number reply.

```
"For BASIC Hangman enter 1
For Animal Hangman enter 2
For Geography Hangman enter 3
....."
```

You follow with something like this:

```
190  INPUT A
200  ON A GOTO 205,215,225,...
205  RESTORE 5000
210  GOTO 250
215  RESTORE 6000
220  GOTO 250
225  RESTORE 8000
...
```

The routine from line 250 READS the word bank into the array Q$(42). If you want more than 42 words you will have to change the dimension of Q$ and make sure that all your banks have the same number of words.

Words are chosen from the Q$ array by the lines from 460 to 466.

```
460  N= INT(RND*42)+1
462  IF Q$(N)="" THEN 460
465  W$= Q$(N)
466  Q$(N) = ""
```

This picks a word at random, checks that it hasn't been used, transfers it to W$ for the game, and then marks off the word in the array.

Before you can teach the 99 to play Hangman, you have to work out how you play, and break your game technique down into a series of separate steps and decisions. Most people start by guessing the commonest letters, vowels first. Then, when a few letters have been found, they will look at the shape of the word and try to guess the word. If they think they know what the word is, then they will normally try the first blank letter. Let's look at an example. Here's the line of blanks:

_____

Vowels first.

"E"	no good	_____
"A"	good one	_A___
"I"	good one	_A_I_

Think, think. No, don't recognize it yet. There are enough vowels. Let's try some consonants. "N" is the commonest.

"N"	no good	
"S"	good one	_ASI_

This looks like BASIC. Try "B" to check.

"B"	spot on	BASI_
"C"	finishes the word	BASIC

We can now write out a game plan as a flowchart. (Figure 2)

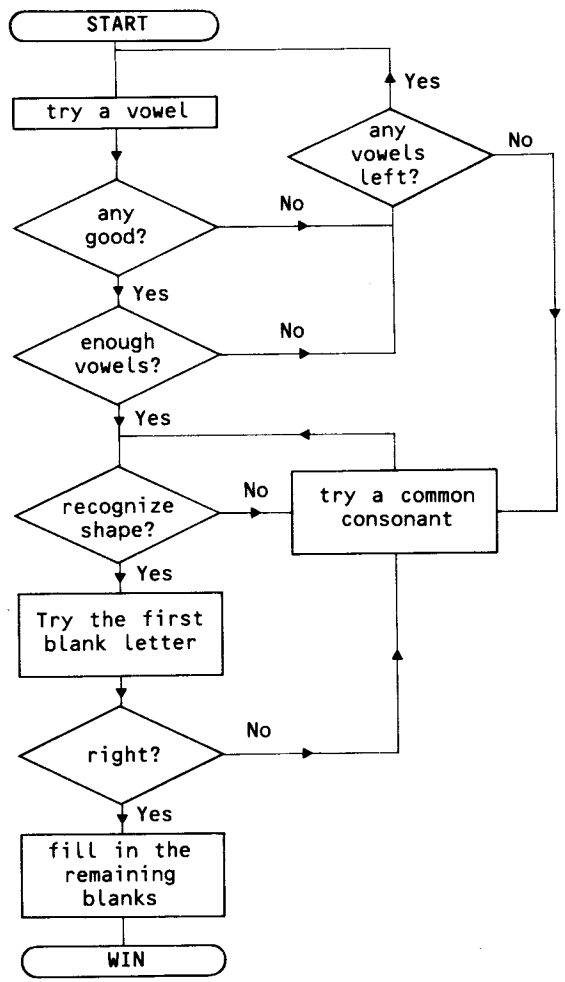


Figure 2

If you can flowchart it, you can program it. (Keep saying that to yourself when things are not going smoothly. It's a very encouraging thought.)

Take it a step at a time. How do you "try a vowel"? You will need an array of vowels – V$(1) = "E" ; V$(2) = "A" ; V$(3) = "I" ; V$(4) = "O" ; V$(5) = "U" ; V$(6) = "Y". You need that "Y" there in case of words like "WHY".

You will also need a couple of variables – VN to keep track of which Vowel Number you are trying, and VF to count how many Vowels you have Found.

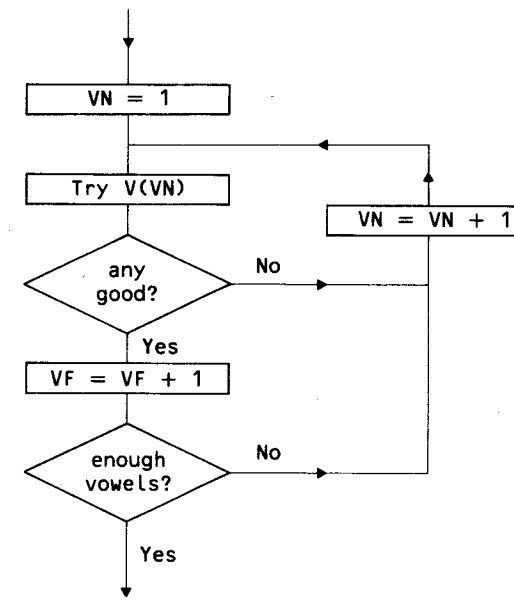


Figure 3

How many vowels are "enough"? As a general rule there's one vowel to every 2 or 3 consonants. This line compares the number of Vowels Found with the length of the word:

```
IF VF = INT (LEN(W$)/3)+1
```

With a five letter word, INT(LEN(W$)/3)+1 comes to 2. Think of a few five letter words and count how many vowels in each.

"Recognize shape"? No problem – as long as the player is only allowed to use words in the 99's word bank. First run through the word bank – Q$(..) and find one the right length (WL is the Word Length)

```
IF LEN Q$(N) = WL THEN.....
```

Transfer it to a simple string store, as this makes slicing much easier.

```
C$ = Q$(N)
```

and mark this word off in your array, so that you don't keep trying it:

```
Q$(N) = ""
```

Now go through the two words, W$ and C$, and compare them letter by letter, ignoring the blanks:

```
1500  FOR T=1 TO WL
1510  IF SEG$(W$,T,1)=" "THEN 1530
                                    (ignore blanks)
1520  IF SEG$(W$,T,1)<>SEG$(C$,T,1) THEN....
                          (back up for a different word)
1530  NEXT T
```

Suppose the word that you are thinking of is "INPUT". The computer has guessed and found I__U_. It checks the word bank and comes up with "BREAK" – the first five letter word it meets. On the first run through the letter checking loop it discovers that SEG$(W$,T,1) – "I" is not the same as SEG$(C$,T,1) – "B". It goes off for another word.

Eventually it finds "INPUT", and checks those letters. The only letters that it has guessed "I" and "U" are in the right places. It completes the loop and is ready for the next stage. Find the first blank letter, and try that.

```
P=POS(W$," ",1)
```

Finds the position of the blank.

```
SEG$(C$,P,1)
```

is the letter at that point in the C$ word.

This kind of routine works perfectly well where you have a limited number of words that the player can use. You could have a much larger word bank if you wanted. The 99 has memory space for several hundred words. With a large bank, the 99 could soon use up its guesses working through all of the words of the right length. This is where you need to write in a consonant guessing routine, so that the computer tries to guess more single letters before it begins to compare with the words in its memory.

The consonant routine works the same as the vowel routine, only now, instead of using the 6 vowels, you will use the 6 (or more) commonest consonants. In normal English these are T,N,S,H,R,D. Check your own word bank though, to see which consonants are used most there.

You now have most of the routines you need to teach the 99 how to play Hangman. For a look at how to organize a computerized guessing game, run the LOGICOL program, and check through its LIST (in the Appendix).

How many words?

How many words of 3 letters or more can you make out of the word "TEXAS" – using each letter once only in each word? The 99 can make 300 – except that most of them are not proper words. We'll come back to that in a minute, but first let's put a word-maker game together. The game is fun to play, easy to write, and gives good practice in handling arrays.

Here's the flowchart.

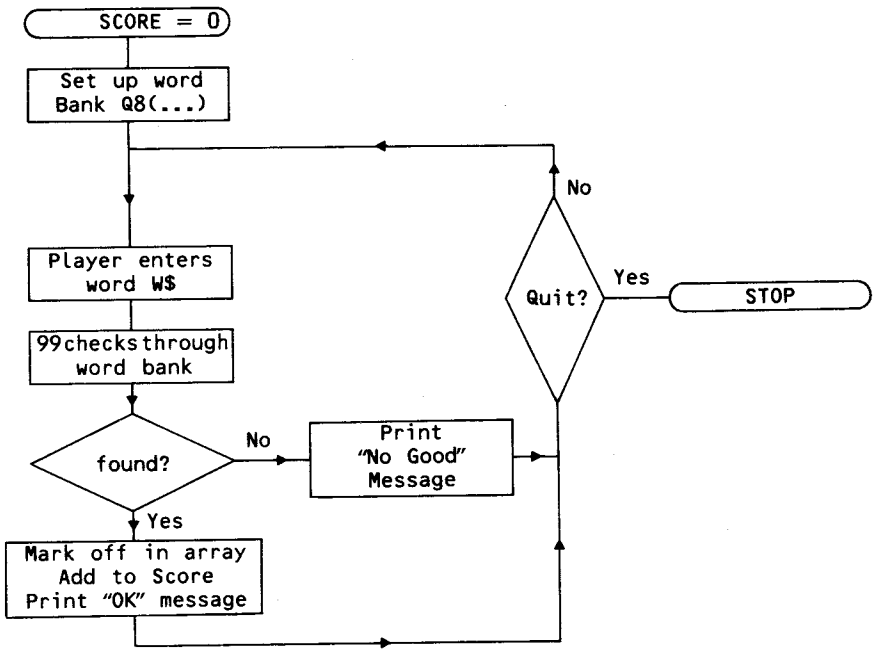


Figure 4

The program starts by READing its word bank from a DATA list into an array. This is done exactly the same as in Hangman. If you are not too bothered about the screen display, then the player's word can be entered by a normal INPUT line, otherwise use the Input Anywhere routine from Pack 2.

Comparing whole words in easy. Simply work through the word bank, checking each word in the array with the player's word.

```
FOR N =1 TO 100      (or however many)
IF W$=Q$(N) THEN.....      (off to O.K. routine)
NEXT N
..      (leads to "no good" routine.)
```

The O.K. routine removes the word from the word bank, so that it cannot be re-used:

```
Q$(N) = " "
```

It adds to the score (longer words count more):

```
XS = LEN(W$)- 2      (eXtra Score)
SCORE =SCORE + XS
```

– 3-letter words get 1 point, 4-letter words get 2, etc.
And it PRINTs an appropriate message:

```
PRINT "THAT'S A GOOD WORD."
```

Now for the Word Bank.

You want to make sure that you include in your word bank every possible word. What better way to do this than to get the 99 to work out every different combination of letters. You can then quickly check through the list to see which are real words.

This routine produces every 3-letter combination from the word "TEXAS".

```
10  W$="TEXAS"

20  FOR N=1 TO 5
30  L$(N)= SEG$(W$,N,1)
40  NEXT N
```

} This splits the word into separate letters – it makes the rest of the program simpler

```
50  FOR A =1 TO 5
60  FOR B = 1  TO 5
70  FOR C = 1 TO 5
80  IF (A=B) + (A=C) + (B=C) THEN 100
90  PRINT L$(A);L$(B);L$(C);" ";
                          (space to separate the words)
100 NEXT C
110 NEXT B
120 NEXT A
```

Look at line 80. This uses the "Value of truth" functions to check whether or not any letter is being used twice. If any of

those pairs of numbers are the same then the equation will have a value of −1, and the program will jump to line 100.

Type the program in and run it. The 99 will print out 60 3-letter combinations, starting with "TEX","TEA","TES","TXE","TXA","TXS"....

Jot down the good words as it runs and you finish up with the following list:

```
TEA,TAX,EAT,ATE,AXE,SET,SEX,SEA,SAT
```

We can add to the program so that it does the same for 4-letter combinations.

```
75  FOR D=1 TO 5
95  NEXT D
```

and alter lines 80 and 90

```
80  IF (A=B)+(A=C)+(B=C)+(A=D)+(B=D)+(C=D)
THEN 95
90  PRINT L$(A);L$(B);L$(C);L$(D);" ";
```

This time you will get 120 combinations. The useful ones are here:

```
TEAS,EATS,EAST,AXES,SEAT
```

You can add a further loop to check for 5-letter combinations. It is interesting – but an almost complete waste of time with "TEXAS". The only other acceptable word from the 120 new combinations is "TAXES".

Try this program out with a different base word. Make sure that the number in the loops is the same as the length of the word.

2
Dice and board games

Time to take a break from the complexities of computer strategies and turn to simple board games. Here the computer can still take an active part, but its play is automatic. Sometimes it's useful to have a second "player" sitting in the 99 ready for when you feel like a game.

The actual screen display is a vital part of any board game. But don't worry too much at first about the quality of your colours or the details of the graphics characters that you are using. They can always be improved later, once the game is running properly. Likewise with any messages or special effects – they can be sorted out later. Get your pieces moving in the right way at the right time first.

The simplest type of dice-based race is one where the counters start on the left-hand side of the screen, and move across to the right, travelling as many squares as are shown on a dice. (Figure 5)

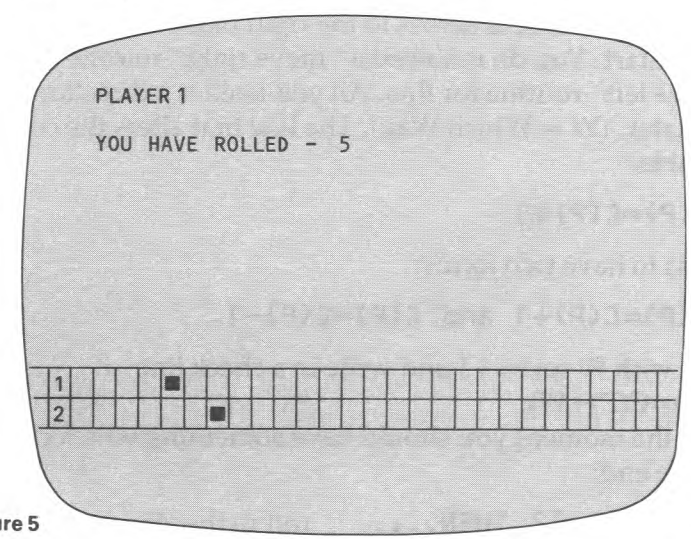


Figure 5

C(1) and C(2) hold the Column numbers for the counters. D is the random number produced for the "dice". A player's movement is run through a simple loop which blanks out his previous position and re-prints one column further on.

```
FOR N = 1 TO D      (how many moves)
CALL HCHAR(15+P,C(P),32)      (P = Player number)
C(P)=C(P)+1
CALL HCHAR(15+P,C(P),120)      (120 being your
                                graphic)
NEXT N
```

Notice here how the player number (P) is used not only to control which Column variable is used, but also to adjust the Row position, so that Player 1 is on Row 16, Player 2 on Row 17.

Try writing a game using a routine like this. Leave the dice as a simple number display for the moment. Get it running and play it until you are bored.

Bored with it yet? Right, now's the time to look at some variations.

1 There and Back again

This time the race is across to the right hand edge, and back to the start. You do not need a "move right" routine and a "move left" routine for this. All you need is a direction indicator. (W = Which Way). The line that alters the column variable:

```
C(P)=C(P)+1
```

needs to have two forms:

```
C(P)=C(P)+1 and C(P)=C(P)-1
```

Start with W set to +1 and write in a check line (C(P)=C(P)+W).

At the moment you should have something which checks for the end:

```
IF C(P)=32 THEN....      (off to the Win routine)
```


Alter that line so that it sends the program off to a line which makes W=−1. Your end-check line should now be:

```
IF C(P)=1 THEN...
```

2 Improve the Display

Moving one square at a time means that you haven't really got room to draw a decent sized board. The only way you can mark out your board is by making little "square" graphics:

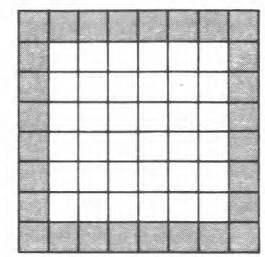


Figure 6

and printing these instead of spaces when you wipe out the counters.

If you make each move two squares long, then it leaves space for marking out a board.

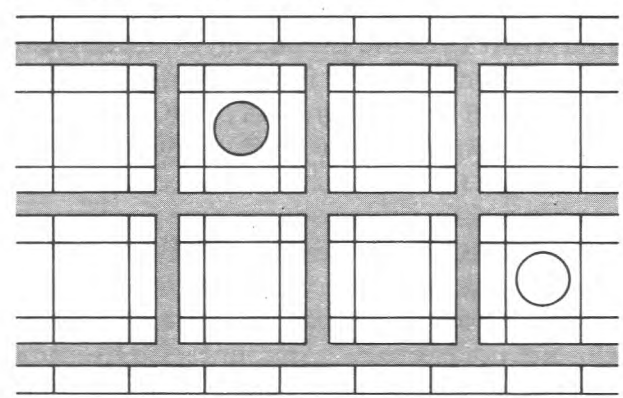


Figure 7

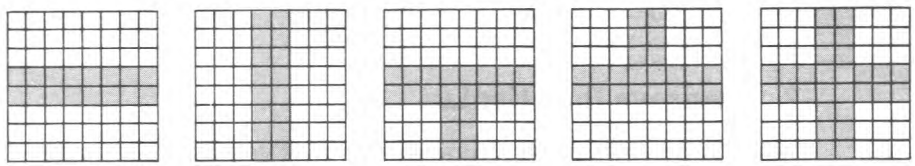


Figure 8

You will need to create a set of graphics like those of figure 8, to produce a board like figure 7. Drawing lines through the middle of squares in this way means that your counters are in the centre of the marked squares.

The movement routine remains the same as before, except that W is now either +2 or −2.

```
C(P)=C(P)+W
```

If you look at the DICERACE program you will see that there the movement is of three spaces at a time. This produces a very spread out board, and does so using only the one board graphic.

3 Round the Board

In the "single-track" races, each counter has had its own line to run along. With a "round the board" game, this could get complicated, and you will find it easier to use a track made out of a single set of large squares, as in DICERACE. There are complications with this as well, but only minor ones.

You do not want to have counters printed on top of each other – unless you are making a Ludo type of game (see below). In DICERACE, each counter has its own corner of the squares, and the three-at-a-time movement keeps it in the same corner. Now the problem is that a single row or column check is not enough. Two counters in the same square could be at rows 20 and 21, and columns 6 and 7. The movement section of DICERACE starts at line 970. Look

there to see how the counters' positions are checked. You will also notice there that a different way of changing positions is used. If a counter is on the right-hand side (below the top square) the program goes to 1040 where the row (P(N,1)) is reduced by three. This is easier than using a "Which Way" variable, as the row number can be added to, reduced, or stay the same. The routines needed to alter each Which Way variable at the appropriate time would be rather complicated.

4 Dice with spots on

The dice display in DICERACE is big and easy to read, but rather slow in appearing. This is because each dice picture is made up of 9 squares. The pictures are all held in the D$() array, and the 99 has to find the right picture and display it with a HCHAR routine, one square at a time. (see lines 5000–5060)

A single character dice could be displayed much quicker, and a better "rolling" effect could be produced, but the graphic would be rather small. A compromise solution is to use a 2×2 dice display, but now you need 24 graphics characters, as each corner of every "face" is different: You may like to alter DICERACE to give it a smaller, but faster dice. Remove the graphics for characters 120 and 122 (line 160), and the whole routine from 425 to 650. Re-dimension the D$() array (line 80) and write in your own routine to define graphics and to set them into the D$() array. You could use the characters from 96 to 119. Don't forget to define your colours for those sets.

Start something like this:

```
430  FOR N=96 TO 119
440  READ G$
450  CALL CHAR(N,G$)
460  NEXT N
470  DATA 0000000000010307,000000000080C0E0,
     0703010000000000,E0C0800000000000..
```

That's the one spot.

Put the characters in the array like this:

```
500  D$(1,1)= CHR$(96) & CHR$(97)
510  D$(1,2)= CHR$(98) & CHR$(99)
```

5 Counting Exercise

Rather than moving the counter automatically after the dice has been rolled, have a routine which asks the player to press the number of the dice. This turns the game into a simple counting exercise for young children. If you do this, then you should include a check routine, so that if the child is unable to find the right number after 2 or 3 goes, the 99 tells him which to press. This can be slotted into DICERACE as a sub-routine, with a line

```
965  GOSUB 7000
```

to send it there. Here's the flowchart for the routine.

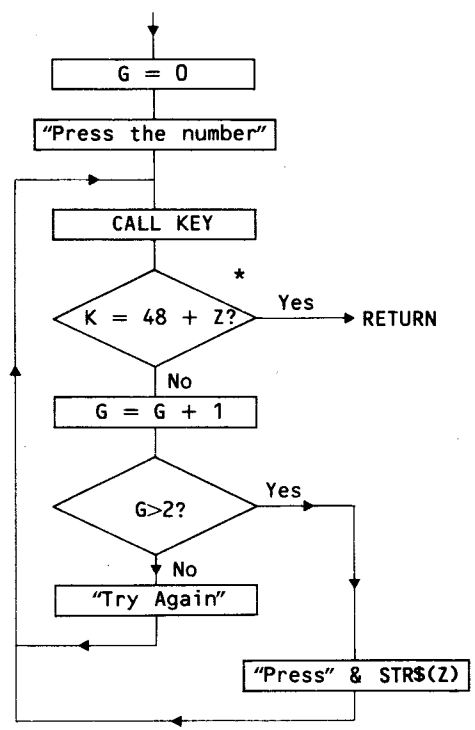


Figure 9

"Z" is the random number for the dice. Add 48 so that it can be compared with the "K" ASCII code.

6 Incidents

Add some excitement to the race by writing in incident routines. These can be of the immediate, or delayed action types – either "go back to the start" or "miss the next go". The section of DICERACE from 3000 is left for your incidents.

First decide where your incidents are, and write in check lines to pick up any counters that land there.

```
IF (P(N,2)>29)*(P(N,1)<5)...
```

This line in the DICERACE game would pick up any counter in the top right-hand square.

```
IF (P(N,1)>10)*(P(N,1)<13)...
```

This would pick up counters on rows 11 or 12, on either side of the board.

To move a counter backwards or forwards is very simple now. You know exactly where it is, so you can tell exactly how far and in which direction you want it to move. You do not need the same kind of checks that you have in the main movement loop.

```
P(N,1)=P(N,1)+6
```

This would move a counter on two, if it was on the left-hand side, and back two if it was on the right.

7 Snakes and Ladders

This is really a variation on There and Back again, with incidents. A Which Way variable is used to control movement, but the routine which changes W also moves the counter up a row.

```
W=W * -1
R(P)=R(P)-1
```

These lines work equally well for both edges. Multiplying by −1 changes plus to minus, or minus to plus.

The counters will rub out your nicely drawn snakes and ladders as they move round, but do not worry, this can be put right. Use the CALL GCHAR sub-program to find out what character is on a square before you print a counter there. When the counter moves off, replace it, not with a space, but with the right graphic. The GHAR check can also be used to spot the tops of snakes and the bottoms of ladders. You will need as many up and down sub-routines as you have lengths of snakes and of ladders. Jump the counter from one end to the other if you are feeling lazy, or when you first put the program together. Move the counter square by square if you are up to making the effort. The 99 has enough memory space to cope with a very detailed game.

8 Ludo

DICERACE can be turned into a simple one-counter game of Ludo. Change the start positions of the counters so that they all occupy the same square. Next write in a GCHAR check so that if a counter lands on top of another at the end of its move, the original counter is sent back to the start. You will also need a GCHAR check on the squares that a counter passes over during a move, so that any covered counter can be reprinted afterwards.

For a proper Ludo game you will need a totally new program. The board needs to be drawn with a set of paths leading to the "homes". Each player needs 4 counters, so the array that holds information about the counters needs to be changed. P(4,4,3) will hold row, column and graphics number for 4 counters for 4 players.

Most importantly, you will have to teach the 99 the tactics of the game. Which of its four counters should it move? Should it chase the one ahead, run from the one behind, or just try to get "home" as quickly as possible? When should it turn off the main track and up the path to "home"?

The game has many complications, and is probably best left until you have reached the War Games section of the book. There you will come across ways of teaching the 99 to make these kinds of tactical decisions.

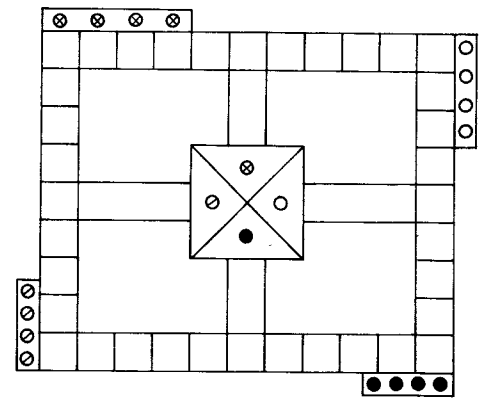


Figure 10

3
Don't get cross!

No, don't get cross if the 99 won't let you win at CROSSES – just fix the program so that it will.

CROSSES is included in this pack for two reasons – it shows how arrays can be used to keep track of a game, and it also shows that it is possible to turn your 99 into an expert at a game. The 99 can learn any game as long as you can analyse the game thoroughly enough and as long as the memory space can cope with all the possible game variations.

Noughts and crosses is a simple game, and there are only a limited number of moves. If you think about how two humans play the game, you will realize that after the first few moves, neither player has much real choice. He is either stopping his opponent from completing a line, or finishing one of his own. In either case, the player is looking for two of the same marks in one line. The game is really decided in those first few moves. That is the more difficult part, so we will leave it until later. Let's look first at the "two-in-a-line" checker.

For the 99 to play intelligently it must be able to "see" the game. You could use the GCHAR function for this, but it is much easier to record the moves in an array. A simple 3×3 array will match the screen display.

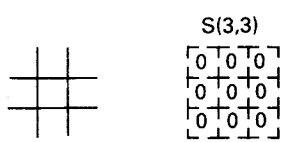


Figure 11

We can now indicate a move by changing the value in the array. In CROSSES, "1" is used to show the player's move.

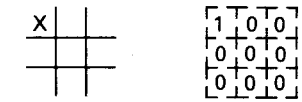


Figure 12

Two crosses in a line would appear as 110, 101, or 011. Whichever way they are, if you add up the values in that line you get a total of 2. The 99 now has a simple way to spot a line that needs stopping. Likewise, if the total of the line is 3, then it knows that you have won – an unlikely event!

We need to mark the 99's move in the array. "1" shows a cross, but we can't use 2 or 3 because a single nought in a line would then have the same value as 2 or 3 crosses. The next available number is 4. Let's use that. A total of 8 now shows two noughts, and the 99 will know to finish that line and win.

Over twenty-five different combinations of noughts and crosses (and spaces) are possible, but if you remember that a single nought, or cross, or space in a line gives the same total wherever it is, then there are only 10 combinations to think about.

Screen display	Array	Line Total	Action?
___	000	0	No
X__	100	1	No
XX_	110	2	Yes – put an O quick!
XXX	111	3	Player wins
O__	400	4	No
OX_	410	5	No
OXX	411	6	No
OO_	440	8	Yes – put an O quick!
OOX	441	9	No
OOO	444	12	99 wins

Figure 13

You will see that if the total is 2 or 8, then the 99 must search for the space in the line and put its nought – either to

stop the player, or to win. A total of 3 or 12 is the end of the game. 6 and 9 both show full lines which can be ignored, and a line total of 5 is no use either, as neither player can win on it. This leaves 0,1 and 4 lines for free use, but in fact, once you have checked for key totals, the overall shape of the game is more important than any single line.

Checking Routines

The checks on the lines cannot simply add across the rows of the array. There are 8 possible winning lines in noughts and crosses.

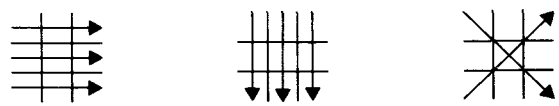


Figure 14

Each line must be checked. One way to do this is to transfer the values of the lines from the screen array, S(3,3) to a working array, W(8,3).

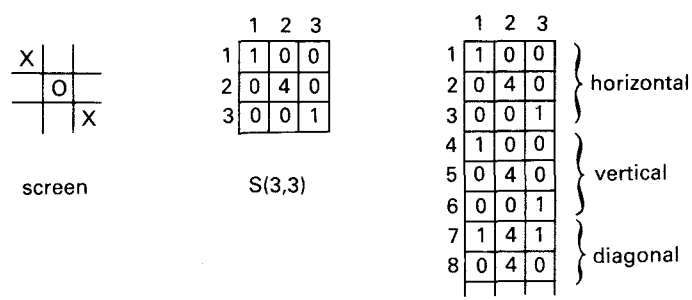


Figure 15

The first three lines are the simplest to transfer:

```
FOR R=1 TO 3
FOR C=1 TO 3
W(R,C)=S(R,C)
NEXT C
NEXT R
```

For the next three, the order is reversed, so that the lines are read down, not across. The only difference is in the actual transfer line:

```
FOR R=1 TO 3
FOR C=1 TO 3
W(3+R,C)=S(C,R)
NEXT C
NEXT R
```

These routines can be happily merged into one. (See lines 500 – on in the LIST of the CROSSES program in the Appendix)

The diagonals are a bit more fiddly, and can best be understood by looking at the co-ordinates:

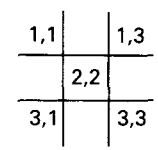


Figure 16

```
FOR R=1 TO 3
X(7,R)=S(R,R)
W(8,R)=S(4-R,R)
NEXT R
```

Line W(7) in the working array now stores the values for squares (1,1), (2,2) and (3,3). W(8) holds those for (3,1), (2,2) and (1,3). This routine is tucked into the R loop at lines 550 and 560.

The program is now able to run through the working array and check line totals. It has to do it in a certain order of priority. It is no good simply totalling each line to see what comes up. The 99 must first check for a line total of 3. (Lines 710 to 770). If the player has won, then there is no point in going further. The next most important line total is 8 – winning chances must not be ignored. (Lines 780 to 830). The third total to look for is 2 so that the player can be stopped. (Lines 840 to 880).

Here is an example of the check routines in action.

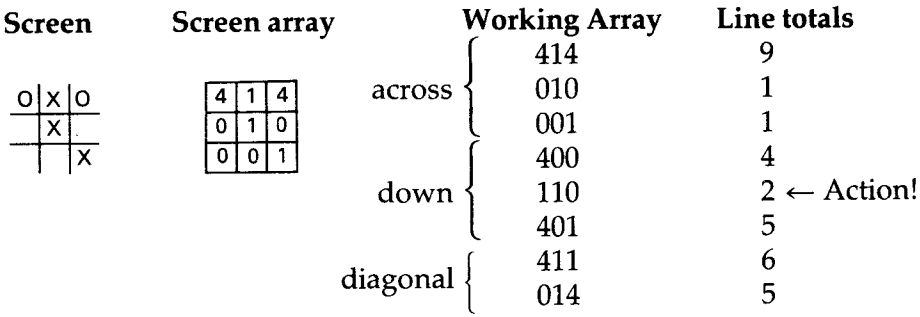


Figure 17

The 99 found no 3 or 8 totals, but it did spot the 2 in line 5 – the centre vertical. The program goes off to the sub-routine at 2000 to look for the blank in the line and write in its 4. This value is then transfered back from the working array to the screen array (see lines 1000 to 1110), and the move is displayed on the screen. (Lines 1130 to 1200).

If the 99 does not find any 2,3 or 8 total, and it almost certainly won't in the early part of the game, then it must "think" about its move.

First move strategy is straightforward. The centre square is so important, that if the player has not already gone there, then the 99 will. If the centre square is in use, then the corner squares are more use than middle of line squares. The routine from 600 on covers this first move. Look at it in the LIST – remember that W(2,2) is the centre square. You will notice a line:

```
610  IF RND>.9 THEN 700
```

This is one of the things that gives you a chance of winning. 9 times out of 10 the 99 will take the centre square if it can, but every now and then it will miss its chance. Remove this line completely if you want to make the 99 even harder to beat. Reduce the random limit if you want to improve the player's chances.

Line 640 will always mark a nought in the top-left hand corner. Because the crosses board is so symmetrical, all corners are, in effect, the same. You could, if you liked, add an extra routine here so that the 99 chooses a corner at random. It would add variety to the game's appearance, but would make no difference to the way it played.

At the second move stage there are two dangerous situations that the line totals check will not spot. They are these.

Figure 18

The 99 could either go for a corner or for one of the middle-of-line squares. Here's what happens with a corner move.

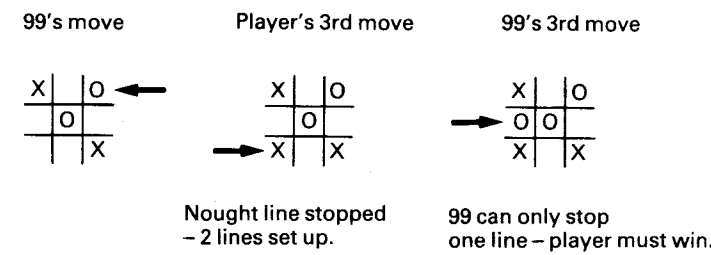


Figure 19

A middle square move takes the game to a draw.

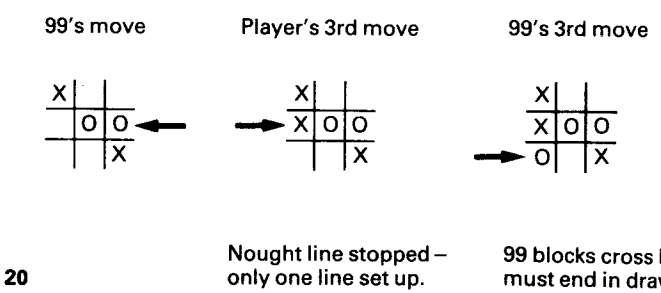


Figure 20

The lines that check for these situations are between 900 and 970. Look at line 900:

```
IF (RND>.9)*(W(1,1)=1)*(W(3,3)=1) THEN 940
```

You will see that another random factor has been built in. Remove that random expression and the 99 will never miss such a dangerous situation. Reduce the random limit to improve the human's chances.

There is one last routine written in to cover those odd situations where there is no obvious move for the 99 to make. In practice the routine will be hardly ever used, but it must be there to fall back on. It is written in from line 2500 onward.

If all else fails, the 99 will mark its nought in the top right (W(1,3)) if it's free, otherwise the bottom middle, or the first empty square it comes to.

Letters to Co-ordinates

You might have expected the squares on the board to have been identified by co-ordinate numbers:

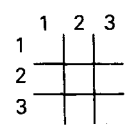


Figure 21

The player would then have been asked "Which Square? Row? Column?" The numbers could have been simply collected by two CALL KEY lines and transferred directly to R and C variables.

```
CALL KEY (3,K,Z)
IF Z = 0 THEN...
R=K-48      (changes ASCII code into number)
```

By the way, you may note that in the program the CALL KEY line has been changed from its usual form . . CALL KEY(3,K,S) to CALL KEY(3,K,Z)

S cannot be used as a simple variable because it is used elsewhere in the program as the arrayed variable S(3,3). Trying to use a simple and an arrayed variable of the same name will give you a *NAME CONFLICT message.

CROSSES uses letters rather than co-ordinates. It makes the program a bit fiddlier, but is easier for the player to operate. The letters are written into the board by the routine between 270 and 310. The key line is 290.

```
CALL HCHAR(9+R*2,13+C*2,93+3*R+C)
```

This gives the spacing – 9+R*2 produces the Row numbers 11,13 and 15, 13+C*2 gives you Columns 15,17 and 19. The third expression turns the R and C variables into the codes of letters from "A" to "I".

R	C	3*R+C	93+3*R+C (ASCII Code)	Letter
1	1	4	97	A
1	2	5	98	B
3	1	10	103	G

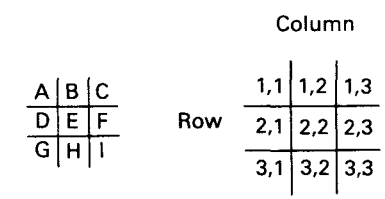


Figure 22

To convert the letter code of the player's move back into co-ordinates, you need another bit of mathematical juggling.

```
400 R=INT(K-62)/3)
410 C=K-(61+3*R)
```

Not very friendly equations, but see how they work. Suppose the player has gone for the centre square. He presses "E", Code "E" is 69. K collects large capitals.

K	K–62	(K–62)/3	INT((K–62)/3)
69	7	2.33	2

If "G" was pressed. .

71 9 3 3

To find C, you reverse the expression used earlier in the HCHAR line.

Letter	K	R	61+3*R	K−(61+3*R)
"E"	69	2	67	2
"G"	71	3	70	1

The 99 Starts

CROSSES has been written so that the player always starts. This was mainly to keep the program simple, but it should also give the player a much-needed advantage. However, it can be readily adapted so that the 99 and the player take it in turns to start. The first change to make is to put the codes for printing the noughts and the crosses into variables. At the moment you have this line after the player's move:

```
CALL HCHAR(9+2*R,13+2*C,88)
```

88 is the code for "X". Change this to:

```
CALL HCHAR(9+2*R,13+2*C,PM)
```

PM (Player's Mark) is set earlier to be 88, but is changed to 79 ("O") at the end of the first game. You will also need CM for the Computer's Mark and this will likewise be switched between 79 and 88. The routine which updates the screen, after the computer's go, is between 1130 and 1200.

To swop those "O" and "X" codes over, use a simple switch routine of the bubble-sorting type. (See next chapter).

First Move

Insert 2 lines before the start of the main game loop.

```
314  IF PM=88 THEN 320
```
(jump next line if player starts)

```
316  GOSUB 1500
```

At 1500 you can write in whatever first move you want. Taking the centre square is the best bet, but some people object to the first player going there. You can always write a randomized routine so that different first squares are chosen. You will also need to include a line to display the move. The game can now return to the main loop and carry on as before.

If, as you play your new version, you discover new dangerous situations, then include more check lines to cover them, after line 900.

4
Dealing and sorting a pack of cards

A full analysis of a card game program is beyond the scope of this book. Even the simplest card games require a very careful study of the tactics and strategy of play. Card-playing isn't simply a matter of logic either. A good player will watch his opponents' expressions, note the way they play during each game, and make guesses about the kinds of hands they hold. However, in this chapter, and further on in "Do you want to bet?", we will cover some of the techniques needed in card game programs.

The program CARDS shows how arrays can be used to hold information about the pack, and about individual hands. It also shows how to sort a hand by order of card value. The program shuffles and deals out all 52 cards, as if in preparation for a game of whist or bridge, and it makes sure that no card is used twice. It aims to do this as quickly as possible. There is always a problem with picking numbers at random, and it is this. The more numbers you pick, the more likely you are to get one that has cropped up already. By the time you are down to the last few numbers, it is very unlikely that you will find an unused one. A little program will show this. Let us suppose that you wanted the 99 to pick the numbers 1 to 10 in random order – each one only once. After a while all the numbers will have been found except one (say 6). This program shows how many times it would have to pick a number before it found that special one.

```
10  RANDOMIZE
20  N=1     (Number of pickings)
30  X=INT(RND*10)+1
40  PRINT N,X
50  IF X=6 THEN 80
60  N=N+1
70  GOTO 30
80  STOP
```

Type this in and run it a few times. How long does it take to find 6? Now change that random number in line 30 to "52".

In the CARDS program, the pack is held in a two-dimensional array – C$(4,13). Each point in the array contains two characters, which show the Suit and the Value of each card. Special graphics are defined for the suits and for the high value cards. The suits are Spades (120), Hearts (113), Diamonds (112) and Clubs (104). Notice that the higher value suits (for bridge players) have higher graphics character numbers. 10, Jack, Queen, King and Ace are defined twice into character numbers 114 to 118 and 121 to 125. This is so that they can be coloured black and red to fit the suits.

The routine from 3000 to 3160 collects the characters into the C$() array. It works through one suit at a time, first READing the Suit graphic (S$), and then calculating the value character, using the T loop (3050 to 3130). The first 8 numbers are converted to card values 2 to 9 with the line:

```
3060  V$= CHR$(49+T)
```

The higher value cards characters are produced by lines

```
3090  V$= CHR$(112+T)     (the black graphics)
...
3110  V$= CHR$(105+T)     (the red graphics)
```

The cards are then "Shuffled and Dealt" by the routine from 3200 to 3300. You will see that this only deals three hands. This avoids the long searches for the last remaining cards. The program first checks that a card has not been used, before it transfers the card to the hand (H$()), and then marks that card off in the pack array.

```
3250  IF C$(X,Y)="" THEN....  (new RND numbers)
3260  H$(T,N)=C$(X,Y)
3270  C$(X,Y)=""     (this card now dealt)
```

To collect the last hand together (H$(4,..)), a separate routine is used – 3400 to 3530. This scans through the whole pack array looking for undealt cards, and transferring them to the fourth hand.

Sorting Cards

The human player would probably sort his cards by first grouping them into suits, and then ordering each suit. You could get the 99 to do it this way, but a simpler method is used in CARDS. It is called "Bubble sorting" or "Ripple sorting".

Every card has two values – the suit and the card number. These "values" are actually the character numbers, and the sorting routine is the same one that would be used for any alphabetical sort. Type in this program:

```
 10  OPTION BASE 1
 20  DIM W$(10)
 30  FOR N=1 TO 10
 40  INPUT W$(N)
 50  NEXT N
 60  FOT T=1 TO 9
 70  FOR N=1 TO 9
 80  IF W$(N)<=W$(N+1) THEN 120
 90  X$=W$(N)
100  W$(N)=W$(N+1)
110  W$(N+1)=X$
120  NEXT N
130  FOR P=1 TO 10
140  PRINT W$(P);" ";
150  NEXT P
160  PRINT
170  NEXT T
```

(INPUT stage. Enter words of any length)

(Sort routine)

(PRINT-out)

Line 80 compares the ASCII codes of pairs of W$() words. If the words are "A" and "B", then it notes that ASCII "A" is 65, and less than ASCII 'B" (66). The sort routine would be jumped. If the words were "AN" and "ANT", then it would find that the first two letters were the same, but that "T" is more than nothing. Again the sort routine would be jumped. If W$(N) ="ZOO" and W$(N+1) ="ANIMAL", then it would go through the lines from 90 to 110.

"ZOO" is there transferred to a temporary store (X$);W$(N) is redefined to be "ANIMAL" and finally, W$(N+1) takes "ZOO" from the temporary store. You could think of it like this:

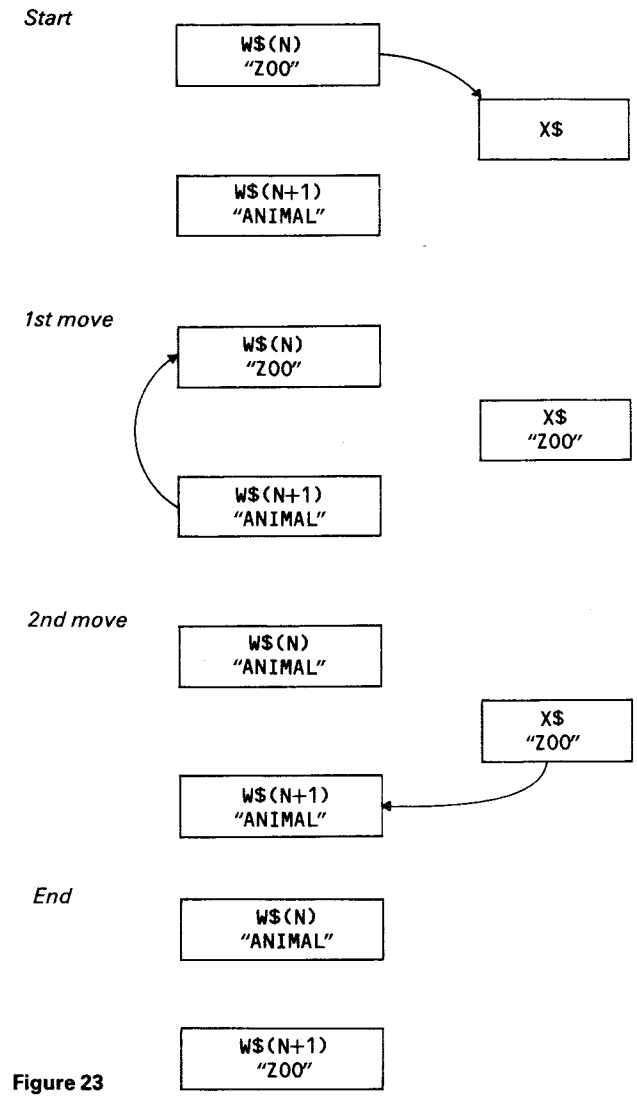


Figure 23

Exactly the same process takes place in the CARDS, except that the sign in the "does-it-need-sorting" line is reversed > = and not < =. This is so that the highest value cards are at the start of the line.

The suit characters codes are fixed so that Spades are worth more than Hearts, and Diamonds and Clubs follow after. The suit characters appear first in the string, so that any Spade is worth more than any Heart, just as any word starting with "B" has a higher value than any word starting with "A". The high card graphics are similarly ordered. Thus, the Ace of Spades has the ASCII codes 120 (Spades) and 125 (Ace). The King of Spades has codes 120 and 124. The 10 of Diamonds has codes 112 (Diamonds) and 114 (ten); the 9 of Diamonds has codes 112 and 57 (normal 9 character).

The sorting routine in the CARDS program is between lines 320 and 440. The fact of having four hands to sort makes it appear a little more complicated, but compare it with the program given above. Notice how in both programs the numbers in the sorting loops are one less than the number of items to be sorted. This has to be, as otherwise the expression W$(N+1) would take you beyond the edge of the array. Both programs also travel round that sorting loop for one fewer times than there are items. On any one run through the inner loop the items can be passed several places to the right, but at most one place to the left. A hand that had the Ace of Spades on the far right would need to be rippled 12 times to pull it across to the far left.

If you want to slow the sort down, so that you can see it better, write in:

```
395 INPUT A$
```

Now press ENTER when you are ready for each new move.

CARDS has been written so that it can form the basis of a card game program of your own. Delete those lines which PRINT the pack (3125), the hands (220–280) and the sorting process (3550–3650). You now have a pack of cards, and no-one can see them!

5
Decisions, decisions

Computers make decisions by means of branching lines, IF. . .THEN. . . ELSE. . .or ON. . . GOTO. . . . You met them in the CROSSES program:

IF (the centre square is free) THEN (put a nought there) LOGICOL uses a far more complicated branching routine to work the 99's strategy. Let's start by looking at the other part of the game, where the human is trying to guess the 99's hidden colours.

LOGICOL may well remind you of the game Mastermind, itself a plastic peg version of an old paper and pencil game. It is similar in a way to HANGMAN, in that a set is to be guessed. Here it is a set of colours. With HANGMAN it is a set of letters (that form a word), but there the player guesses only one item at a time, rather than a whole set.

The computer's checking routine is much the same, but now, instead of checking one variable against a string, the 99 is checking a string against another string. Look at the LIST for the program in the Appendix, and find lines 900 to 1000. P$(4) holds the player's 4 guessed colours. C$(4) holds the 4 colours that the computer picked at the start of the game. The two strings need to be compared to see if any colours are in the same place in both strings (line 920 IF P$(T)=C$(T) THEN 950), and also if any of the player's colours turn up anywhere in the computer's set. (Line 930 IF P$(T)= C$(N) THEN 970). In the following example, the computer's set is Blue, Cyan Green and Magenta. The player has just guessed Red, Green, Blue and Magenta.

```
C$(4) B C G M
P$(4) R G B M
```

On the first run through the T loop, the 99 finds that C$(1) and P$(3) are both "B" – blue. It marks off one Right Colour.

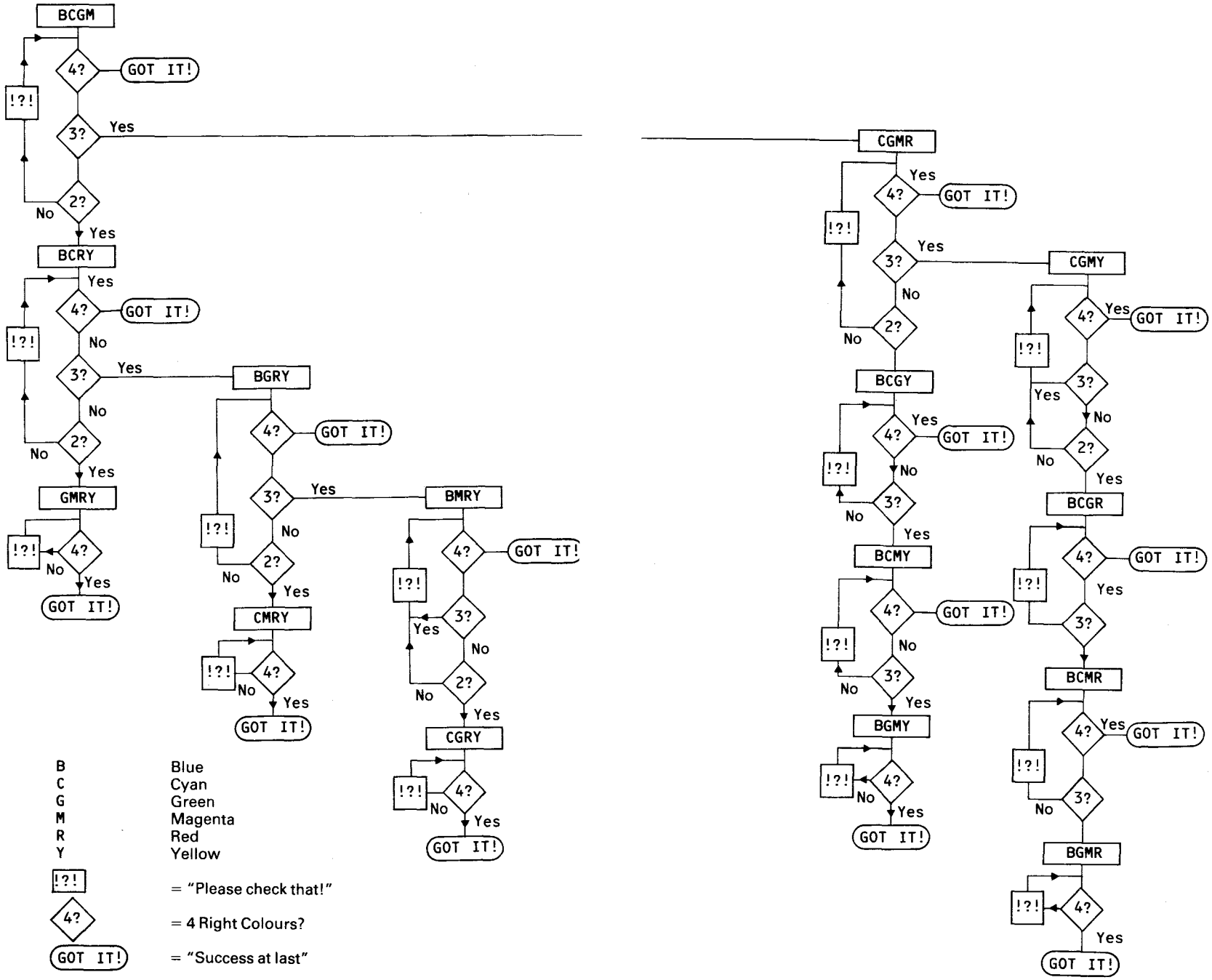

BCGM
4?
GOT IT!
!?!
3?
Yes
2?
No
Yes
BCRY
Yes
4?
GOT IT!
!?!
No
3?
Yes
BGRY
No
2?
No
Yes
GMRY
!?!
4?
No
Yes
GOT IT!
4?
GOT IT!
!?!
3?
Yes
BMRY
No
2?
No
Yes
CMRY
!?!
4?
No
Yes
GOT IT!
4?
GOT IT!
!?!
3?
Yes
No
2?
No
Yes
CGRY
!?!
4?
No
Yes
GOT IT!
B Blue
C Cyan
G Green
M Magenta
R Red
Y Yellow
!?! = "Please check that!"
4? = 4 Right Colours?
GOT IT! = "Success at last"
CGMR
Yes
4?
GOT IT!
!?!
3?
Yes
CGMY
No
2?
No
BCGY
Yes
4?
GOT IT!
!?!
No
3?
No
Yes
BCMY
4?
GOT IT!
!?!
No
3?
No
Yes
BGMY
!?!
4?
No
Yes
GOT IT!
4?
Yes
GOT IT!
!?!
3?
Yes
No
2?
No
Yes
BCGR
4?
GOT IT!
!?!
Yes
3?
BCMR
4?
Yes
GOT IT!
!?!
No
3?
BGMR
!?!
4?
Yes
GOT IT!

Figure 24

The common "G" is picked up on the third run. Finally it will spot that C$(4) and P$(4) are the same, and will add one to the Right Place score.

Once you have had your go, whether you got it right or have simply run out of guesses, then the 99 will ask you if it can have a go.

You don't have to let it, and, unlike some human players, it won't sulk if it doesn't get a turn. It's not very good at the game in any case, and can only sort out which are the right colours, not where those colours should be. The program allows for four out of six colours to be used, and each one can only be used once. This means that there are 15 possible combinations. The flowchart in figure 24 shows how the 99 works out which combination. You will notice that the 99 solves the problem in no more than six guesses.

That flowchart is the result of a lot of game-watching, and several lengthy sessions with pencil and paper. This is essential with any game program. Do not expect to be able to write it directly into the 99. Think about how you and other humans play the game. Record every guess you make, every decision you take, and the reasoning behind every decision. When you have your strategy written down as a series of decisions, it is quite easy (really!) to turn it into the 99's program.

Lines 2070 to 2890 in the LOGICOL program show that flowchart turned into lines. For simplicity each stage is written in exactly the same form. The following lines cover that section of the flowchart on the top of the right-hand side:

```
2700  G$="CGMY"
2710  GOSUB 5000
2720  ON K GOTO 2730,2750,2730,3000
2730  GOSUB 3500
2740  GOTO 2720
```

G$ is the Guess. The sub-routine at 5000 displays the colours and collects the player's response into the K store. The answer must be a number in the range of 1 to 4. At any stage of the program a "1" response will send the program to a line

that leads to the sub-routine at 3500, which asks the player to "Please check that!" It is not possible to have only one colour right, as this would mean that there are 3 wrong. As the 99 has guessed 4 out of 6 colours, the most it can miss is 2.

At this particular stage a "3" response will also lead to a "Please check that." To have reached this point the 99 must have got 3 right on each of its earlier guesses. It has already found two wrong colours, and could not possibly have found a third. (See the example below).

A "2" response sends the program on to the next guess (line 2750). A "4" jumps out of the guessing routine to "How's that then!" and "Do you want another go?"

If you look at any of the sections which cover bottom of flowchart lines you will see the ON K GOTO. . . line looks like this:

```
2870  ON K GOTO 2880,2880,2880,3000
```

The branching here is really only two ways. Either the player tells the 99 that he has got all 4 right, or he doesn't. These lines could be replaced by a simple:

```
2870  IF K = 4 THEN 3000
2880  GOSUB 3500
```

Play through the game a few times, following the 99's "thoughts" on the flowchart. You should see something like this. Here the colours B,C,G,R were entered by the player.

Guess	Response	99's thoughts:
BCGM	3	Either R or Y must be right
CGMR	3	Either C G M are right and I need Y or both B and R were right. Try Y.
CGMY	2	So B and R were right. Y is no good.
BCGR	4	How's that then?

This time the colours were CGRY

BCGM	2	2 missing. Must have R and Y.
BCRY	3	And B or C; and G or M – not very helpful.
BGRY	3	Perhaps the B was right. try M.
BMRY	2	The B was wrong, and the M. But now I know the answer.
CGRY	3	"Please check that" (He's cheating!)
CGRY	4	I should think so too.

It is perfectly possible to work out a program to sort the colours into their right places. Human players can usually do it in 4 to 8 goes. The program could be a straightforward branching one, but you would need an awful lot of branches.

Think of how a human player behaves. He will check back through his previous guesses noting how many he had right, and in the right place, each time. He will see which colours can be fixed, and which are likely possibilities, and then test out his possibilities next time. The game might go like this:

Guess Number	Guess	Right Place	Right Colour	Thinks: Which? Which order?"
1	BCGM	0	3	"One wrong colour, and all out of place."
2	CGMR	0	2	"B and Y must be there."
3	BYCG	1	2	"Still a colour short."
4	YCMB	0	4	"Got the colours!"

At this point he knows that the colours are YCMB, and by checking guesses 1,2 and 4 he knows that neither B nor C nor Y should be the first colour. He can fix M as the first colour.

5	MBYC	2	2	"Not a lot of help – try a shuffle."
6	MYBC	1	3	"Ah! Now we're getting somewhere."

The second colour must be one of Y, B or C, but it can't be C (guess number 4 proves that) and it can't be Y (guess number 6). It must be B. M and B must have been the colours in the right place in guess number 5, so we just need to swop over Y
and C.

7	MBCY	4	0	"Logicol, isn't it?"

You are now in a position to start translating the human reasoning process into computer language. What follows is not the only way of tackling this, and is not, in any case, the complete solution. Rather it is a set of suggestions that you should be able to work up into the necessary routines.

For a start you will need a number of new arrays to store and handle information. Of these the most significant will be a Marking array, (M$(4,6)) in which the 99 can mark off those colours that are in the wrong places. You will need to set this up at the start of the 99's go, and it should have this form:

M$(4,6)

B	B	B	B
C	C	C	C
G	G	G	G
M	M	M	M
R	R	R	R
Y	Y	Y	Y

Figure 25

Now, whenever the 99 gets a "none in the Right Place" result it can work through that array, marking off those bad guesses.

```
IF RP=0 THEN....      (to the following routine)
FOR N=1 TO 4
FOR T=1 TO 6
IF SEG$(G$,N,1)<>M$(N,T) THEN...
                                  (jump to NEXT T)
M$(N,T) = ""
NEXT T
NEXT N
```

The player has chosen RMCB. The 99's first guess is BCGM, as usual. None are in the right place. After it has been through the mark-off routine, the M$() looks like this

M$(4,6)

	B	B	B
C		C	C
G	G		G
M	M	M	
R	R	R	R
Y	Y	Y	Y

Figure 26

3 of the colours were right, so the next guess is CGMR. Here's that array again after this time.

M$(4,6)

	B	B	B
		C	C
G			G
M	M		
R	R	R	
Y	Y	Y	Y

Figure 27

3 Right Colours again. The next guess should be CGMY, but in this version it pays to alter the order of the letters each time, as this helps to find the right places. The guess is GMYC.

This gives a "1 in the Right Place" result, which is not very helpful as the 99 hasn't a clue which one. It does, however, know that as only 2 colours were right altogether, the Y must be wrong. You can write in a little routine at this stage in the branching program to mark off all Y's.


```
FOR N=1 TO 4
M$(N,6)=""
NEXT N
```

Time for the next guess at the colours. It is written as BCGR in the simple version, but we will shuffle the letters for this version. G$ = "GRBC" This produces another "None in the Right Place" result. The M$() array is getting quite empty by now.

M$(4,6)

	B		B
		C	
			G
M	M		
R		R	R

all "Y"s marked off already

Figure 28

The next guess at the colours finds the 4 right colours. The guess has been shuffled so that G = "RBCM". This gives us a "1 in the Right Place" result (the C), but at the same time lets us mark off the other wrong colour – G.

We now need to go back to that earlier guess where we had 1 in the Right Place – GMYC. A temporary store is needed to hold those sort of guesses, so that they can be recalled later. (S$(10) will hold ten guesses). Compare that with the M$() array to see if 1, and only 1 could be right.

```
FOR X = 1 TO 10
Z = 0     (counter)
IF S$(X)="" THEN....      (jump to next X, this store
                           is empty)
S1$=S$(X)     (simple Strings are easier to handle)
FOR N=1 TO 4
FOR T= 1 TO 6
IF M$(N,T)<>SEG$(S1$,N,1) THEN....
                                  (jump to NEXT T)
Z=Z+1     (found one that matches)
T1 = T     (remember where you found it!)
N1 = N
```

```
NEXT T
NEXT N
IF Z =1 THEN.....      (next routine)
NEXT X
```

The "M" is found in column 2 (N1 = 2) and row 5(T1 = 5).

The next routine trims down the M$() array. First it goes through that array and marks off all the other appearances of the "Right Place" colour.

```
FOR N= 1 TO 4      (across the columns)
IF N= N1 THEN.....      (jump to NEXT N)
M$(N,T1)=""      (mark off) other M's, at M$(1,5),
                 M$(3,5) and M$(4,5)
NEXT N
```

In the example, the M in the second place was the Right one. The M$() array is pared down to this.

	B		B
		C	
	M		
R		R	R

Figure 29

We know that M is right in the second column, so any other letters that are there can be removed.

```
FOR T = TO 6      (down the rows)
IF T = T1 THEN.....      (jump to NEXT T)
M$(N1,T)=""
NEXT T
```

This removes the "B", leaving only the "M".

A simple check down the columns to see if any column has only one letter left, will show any other certain Right Places. The first glance at the array shows us that R must be the colour in the first place. Mark that off in the other columns and it leaves only C in the 3rd column and B in the 4th.

If that was the end of the story, then life would be (relatively) easy. Unfortunately it isn't. There will be many times when the 99 has found the 4 colours, but does not have enough information to work out their places. It will now have to try new combinations of the colours. There are 24 possible combinations! Over half of these will give 1 or 2 Right Place results, which are not a lot of use to the program. The simplest approach is to try a few shuffles and hope that something comes out of them. This routine swops the last two colours over.

```
G$ = SEG$(G$,1,2) & SEG$(G$,4,1) &
SEG$(G$,3,1)
```

This swops the two halves of the string over.

```
G$ = SEG$(G$,3,2) & SEG$(G$,1,2)
```

The "shuffle and test" routine should increase the 99's success rate. It is possible to write a far more sophisticated program that compares every guess and its results, the way a very good player does, and performs the complex logical reasoning needed to solve the problem quickly. However, who wants the computer to win every time?

To adapt the LOGICOL program so that the 99 can try to find Right Places as well, you will need to do the following. Slip in a GOSUB line somewhere between 1850 and 2070, to send the 99 to a sub-routine to set up its arrays. Add in the main "Right Place" routine at 5200. This should include the player's response, marking off M$() and storing any "1 Right Place" results. When 4 Right Colours have been found, the program should now go to a section to check if all the Places are known, and if not, to "shuffle and test". You will also need to alter the order in which the letters appear in the Guesses, so that more places are tested, and you can add in two small routines to mark off "R" and "Y" at certain places in the branching routine. Figure 29 shows the new version of this routine.

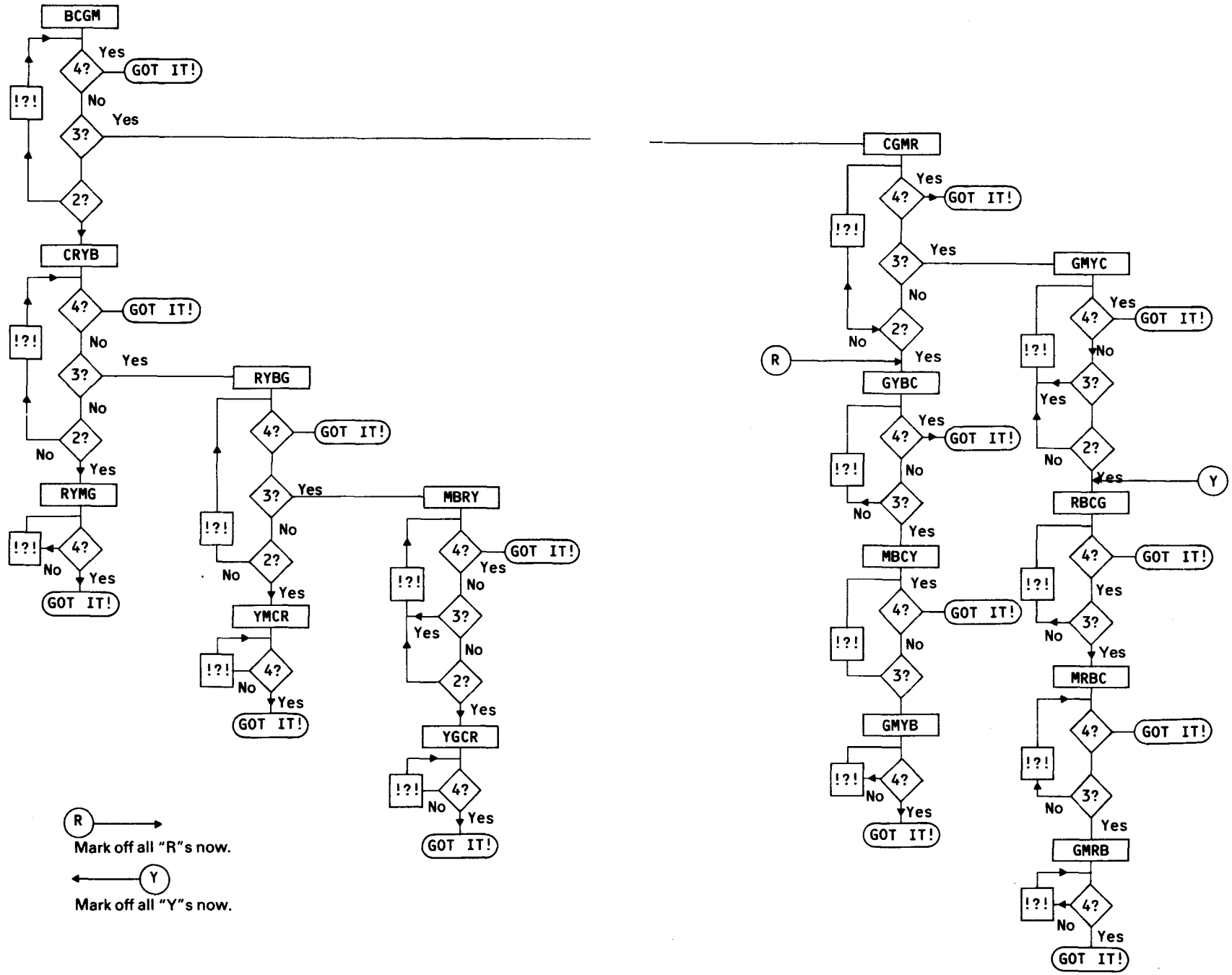

BCGM
4?
Yes
GOT IT!
!?!
No
3?
Yes
2?
CRYB
4?
GOT IT!
!?!
No
3?
Yes
RYBG
No
2?
No
Yes
RYMG
!?!
4?
No
Yes
GOT IT!
4?
GOT IT!
3?
Yes
MBRY
!?!
No
2?
No
Yes
YMCR
!?!
4?
No
Yes
GOT IT!
4?
GOT IT!
Yes
!?!
No
3?
Yes
No
2?
Yes
YGCR
!?!
4?
No
Yes
GOT IT!
R
Mark off all "R"s now.
Y
Mark off all "Y"s now.
CGMR
4?
Yes
GOT IT!
!?!
3?
Yes
GMYC
No
2?
No
R
Yes
GYBC
4?
Yes
GOT IT!
!?!
No
3?
No
Yes
MBCY
4?
Yes
GOT IT!
!?!
No
3?
GMYB
!?!
4?
No
Yes
GOT IT!
4?
Yes
GOT IT!
!?!
3?
Yes
No
2?
No
Yes
Y
RBCG
4?
GOT IT!
!?!
Yes
3?
No
Yes
MRBC
4?
GOT IT!
!?!
3?
No
Yes
GMRB
!?!
4?
No
Yes
GOT IT!

Figure 30

6
Do you want to bet?

The worst thing about playing gambling games with a computer is that it won't pay up when it loses. The best thing is that it doesn't expect you to either! "Pick-a-Straw" and coin-tossing games were covered in the earlier Game Writer's Pack. Here we will look at two games where the 99 has to think about its game.

Vingt-et-un

This game is also known as "Pontoon" and "21". The object is to get a set of cards that total 21, or as near to 21 as possible. All picture cards have a value of 10, and the Ace counts 11, though in some versions of the game it can also be counted as 1. Cards are dealt by the "Banker", who also acts as a player. At first each player is dealt 2 cards, further cards are dealt at the players' request. A player whose hand totals more than 21 is "bust" and out of the game for that round, otherwise, the one whose hand is closest to 21 wins. In the version that we will work out now, each player pays 5 chips at the start of each round, and a further chip for each extra card. The bets are collected into a kitty which goes to the winner, or the banker if all the other players are "bust".

Figure 30 shows the flowchart for the game.

A number of arrays are needed here.

P$(4,13) The ordered Pack
D$(52) The shuffled Deck
N$(6) Names of up to 5 players and "Banker"
H$(6) Their Hands
HV(6) The Hand Values
B(6) The Bets
C(6) The players' Chips – start with 100 each.

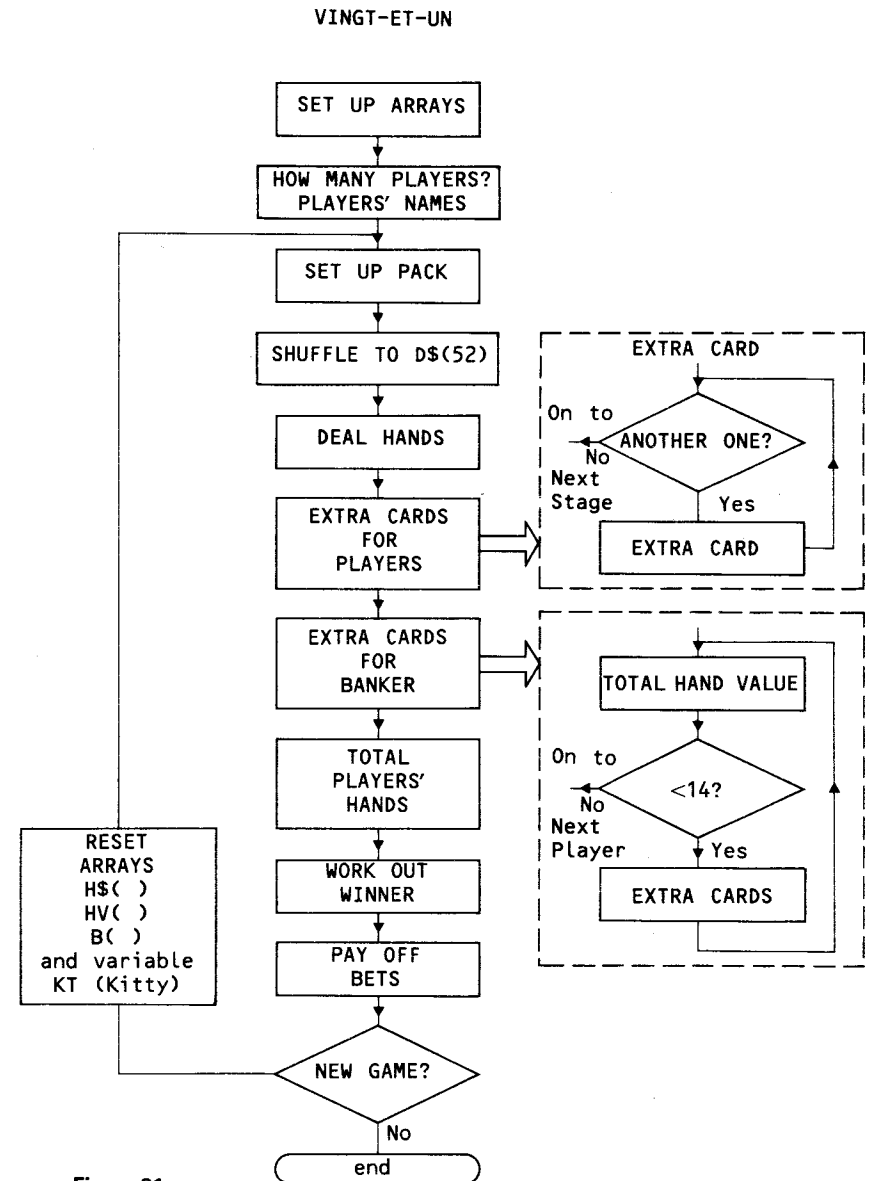


Figure 31

You will also need several sub-routines which are not shown on the flowchart. These are to allow you to PRINT anywhere, INPUT anywhere, and check INPUTs for number

values. Normal PRINT and INPUT commands will obviously destroy any screen display. If you are working in EXTENDED BASIC you can use the DISPLAY AT and ACCEPT AT commands instead.
You can adapt the CARDS program to give you some of the routines that you will need. It will also save you the trouble of defining your own graphics. Notice a significant change here. In CARDS the pack was shuffled and dealt in one operation. You could use a variation of that technique, by picking a card at random from the Pack, each time one was wanted. Here another technique is used, for the sake of variety. Instead of the cards going, at random, to different hands, they are collected into a single array – D$(), the shuffled Deck. In this particular game, you will never actually need a full deck. Even with 6 players, you will only use at most 30 cards, as no hand is likely to have more than 5. It is enough, therefore, to shuffle only 30 or so cards into D$(). This will save the long hunts for the last few unused cards.

The number of players is collected at the start of the game into variable PN. This is then increased by 1, and the last player is the 99. Players' names are collected by a loop:

```
FOR T = 1 TO PN-1
INPUT "NAME ":N$(T)
NEXT T
N$(PN)="BANKER"
```

Hands are dealt into single string arrays. The variable CN keeps track of the Card Number in D$(). At first 2 cards are dealt to each player:

```
FOR C = 1 TO 2
FOR T = 1 TO PN
H$(T)= H$(T) & D$(CN)
CN=CN+1
NEXT T
NEXT C
```

If a player is lucky, then his hand might be like this – H$(4) = "♣ 10 ♦A" A variation on this routine is needed for the

"Extra Cards for Players". Work through the players one at a time (not including the 99) and ask if another card is wanted. If yes, add the next D$() card to his Hand (H$()). Ask again. Players can have as many extra cards as they want. Include a line in that routine to add 1 to his Bet (B()) and take 1 from his pile of chips (C()).

You cannot, of course, ask the 99 if it wants another card. It must work that out for itself. To do this we will need a routine that will work out the values of hands. Write it as a subroutine, as it will also be wanted in the later stage where the hands are assessed to see who wins. At that stage the hands will be run through a loop (FOR T=1 TO PN−1), so send the 99's hand off using the same variables.

```
T=PN
GOSUB.....      (totalling)
```

This sub-routine is here numbered from 7000, but it could be anywhere. The 99 works through the hand, picking out the card value characters, and ignoring the suits. The ASCII code of those characters is then converted into numbers between 2 and 11. All "Picture Cards" count 10, all Aces count 11.

```
7000 FOR L=2 TO LEN(H$(T)) STEP 2
7010 V$=SEG$(H$(T),L,1)
7020 V=ASC(V$)-48
7030 IF V<9 THEN 7080      (cards 2 to 9)
7040 IF (V=70)+(V=77) THEN 7070      (Aces)
7050 V=10      (all picture cards)
7060 GOTO 7080
7070 V=11
7080 HV(T)=HV(T)+V
7090 NEXT L
```

Suppose the hand consisted of the 3 of diamonds, the Ace of spades and the queen of clubs. H$()= "♦3 ♠A ♣Q" Going through this in steps of 2, line 7010 will pick up 3, A and Q. Watch what happens to each. First the 3, ASCII code 51, line 7020 gives V a value of 3. This is less than 9, so the program jumps to line 7080. The total Hand Value is, so far, 3. Next the Ace. It's a black Ace, so its ASCII code is 125; line

7020 gives 77 (125–48). This is more than 9, and the program runs past 7030 to 7040. From there it jumps to 7070, and V is reset to 11. Total Hand Value now 14. Lastly the Queen, ASCII code 123 (black).'Its initial Value of 75 takes it past 7030 and 7040, and the Value is changed to the 10. The program then jumps to the totalling line, and the player finds he is bust, 24 is no good.

Check that HV (the Hand Value) is set to 0 before going to the routine. If you are mathematically inclined, you can work out the percentage chances of different values of cards cropping up next, given what has already been visibly dealt. This can form the basis of the 99's decision to take another card or not. If you are not so inclined, then follow the flow-chart. It is a reasonable bet that the next card will have a value of 7 or less. (Halve the time you will be right). So, if the 99's total is less than 14, send the program off to deal it another.

At the final stage, "Work out Winner", you must first check that a player's hand is not over the limit. Next see if it has a total of 21. If so, then declare him the winner, and increase his pile of chips by the value of the KiTty. (KT). Otherwise, compare his hand with the Best Hand (BH) so far.

BH is set to 0 at the start of this routine. By the time the program reaches the Best Hand lines, it has already checked that the hand is not "bust", and not worth 21. You simply check that it is better than the previous best.

```
IF HV(T) < BH THEN....      (jump next 2 lines)
BH = HV(T)
WN = T
```

BH now has the Best Hand Value, and WN has the Winner's Number. WN should be set to the same as PN (the computer's number) beforehand, so that if there are no winners, then the computer collects the kitty.

Note that before a second game can be played, you need to reset several of the arrays, and the Kitty, and that the Pack will need to be re-organized and shuffled again.

Plan out a rough screen display before you start to put the program together, and work to that until you are satisfied

that all the routines are in and running properly. Come back to the display at that stage, and improve your layouts. That is also the time to alter routines if the program does not run the game the way you are used to it. You may, for a start, want to allow Aces to count High or Low. For the players' hands, this simply means adding a routine to ask the player what he wants to do about his Ace. For the 99 you will need add some lines in the "Extra Card" routine. At the simplest, you could check, after the extra card, whether or not the 99's hand was over the limit. If so, does it contain an Ace? If it does, knock 10 off the hand value, and check to see if it is worth having another extra card.

Pokerdice

This game combines technique from "Dice and Board Games" and from "Cards". Its flowchart is shown in figure 32.

Pokerdice, in case you have never come across the game, works like this. The "dice" used in the game have symbols on, to represent high value cards, 9 to Ace. 5 dice are used. The player rolls all the dice at first, and can then for two more turns, select which ones he wants to roll again. The object of the game is to build up "poker hands". 3 of a kind are worth more than a pair, 4 of a kind more than three. A "flush" – a set of cards in series, 9, 10, Jack, Queen, King – is worth more than 4 of a kind, but less than 5. The best hand you can collect is 5 Aces – which is possible in Pokerdice, though not in real poker (unless you are an awful cheat).

Let's work through that flowchart. You will need your graphics for the 6 faces of the dice. At first though, you could simply use numbers 1 to 6, then add your fancy graphics later. As with ordinary dice, these can be single characters, or formed out of several characters. Include a page of instructions and rules of the game if the program is likely to be used by people who don't know it.

Several arrays are used in the game, but only one will need dimensioning at the start. D(2,5) will hold the 5 dice numbers of the player and the computer –'this version is for

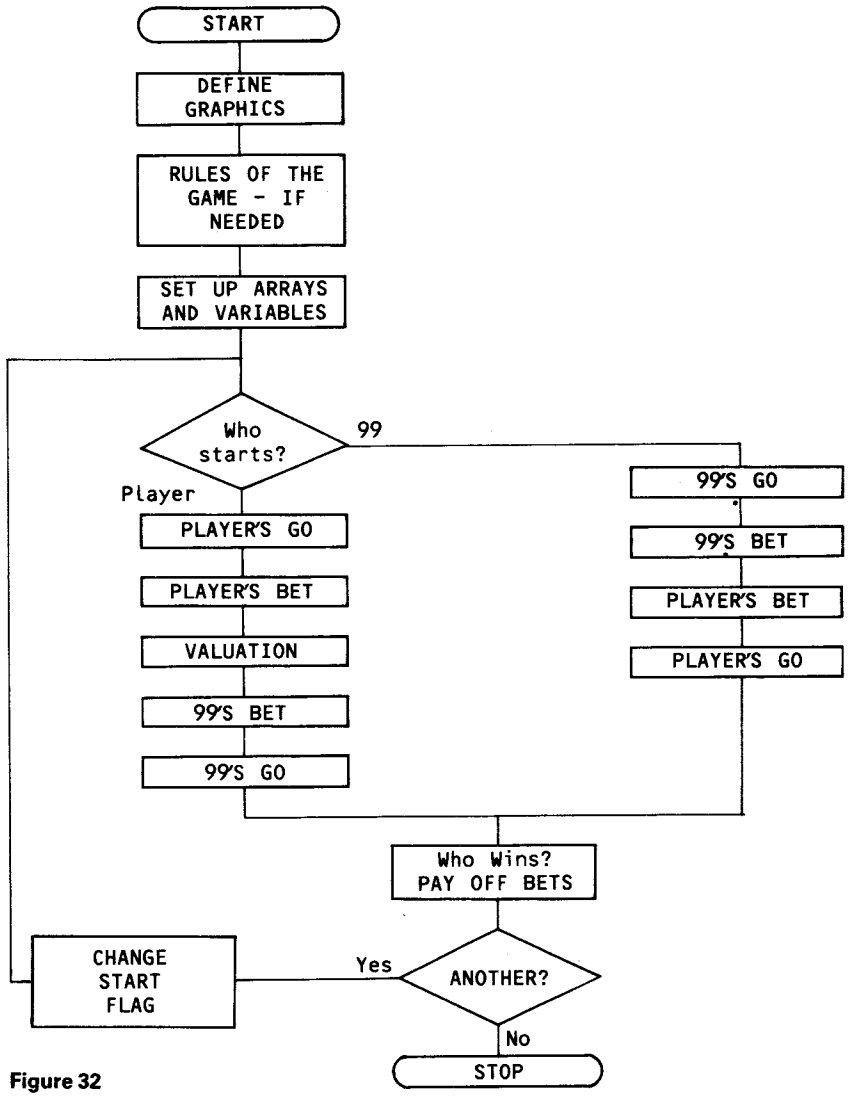


Figure 32

one human only, but could be simply adapted for several players. The other arrays and variables will become clearer as you work out the program, only one need be mentioned here. S will indicate who Starts. Set to +1 initially this lets the player start. At the end of the round it is multiplied by −1. The effect is to make it +1 and −1 in turns.

The Player's go is shown in more detail in figure 33.

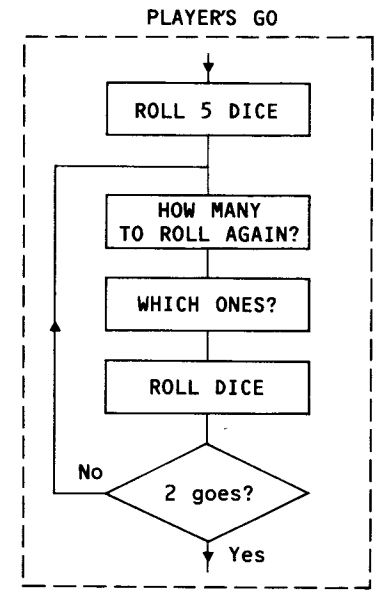


Figure 33

This routine will "roll" and display 5 dice. The dice are here single characters with codes from 128 upwards. The Player number (P) is 1 for the human, 2 for the 99.

```
3000  FOR T= 1 TO 5
3010  FOR N= 1 TO 6                    } "Rolling"
3020  CALL HCHAR(20,T*3,127+N)         } dice
3030  NEXT N
3040  D(P,T)= INT(RND*6)+1      (the way it lands)
3050  CALL HCHAR(20,T*3,127+D(P,T))
3060  NEXT T
```

Lines 3010 to 3030 "spin" the dice on screen. You might like to add a CALL SOUND(. .) line in there to slow things down. The CALL HCHAR(. . lines used here will print the dice characters at Row 20, spaced out in 3's, from 20,3 to 20,15. Adjust these to suit your own display.

You can tackle the "Which ones do you want to re-roll?" section in various ways. Here's one. This assumes that the dice have some kind of reference number shown on the screen, as in the example display in figure 34.

```
BONUSES                MONEY
5 OF A KIND     150    YOUR CASH      £100
FLUSH           120    MY CASH        £100
4 OF A KIND      90    YOUR BET         £5
FULL HOUSE       60    MY BET          £10
3 OF A KIND      30
PAIR             10

YOUR HAND VALUE 34

                    DICE NUMBERS
MY GO          1    2    3    4    5
              10    J   10    9    K

ROLLING MY DICE AGAIN
```

Figure 34

```
1500  FOR T= 1 TO 2     (you can roll again twice)
1510  INPUT "HOW MANY TO ROLL AGAIN?":M
1520  FOR N=1 TO M
1530  INPUT "NUMBER ? ONE AT A TIME,
       PLEASE":RR(N)
1540  NEXT N
1550  FOR N=1 TO M
1560  D(P,RR(N))=INT(RND*6)+1
....      (rolling display)
.....
1600  NEXT N
```


The numbers of the dice to be Re-Rolled are collected in the RR( ) array in the following manner. Suppose the player has had his first roll and is faced with this set of dice:

Dice Number	1	2	3	4	5
Face Value	A	9	J	A	Q

The player wants to keep the 2 Aces and roll the rest. He tells the 99 he wants to roll three, then enters the numbers 2,3 and 5. When the 99 reaches the routine starting at 1550 it finds that RR(1) =2; RR(2)= 3 and RR(3)= 5. Those dice are rolled.

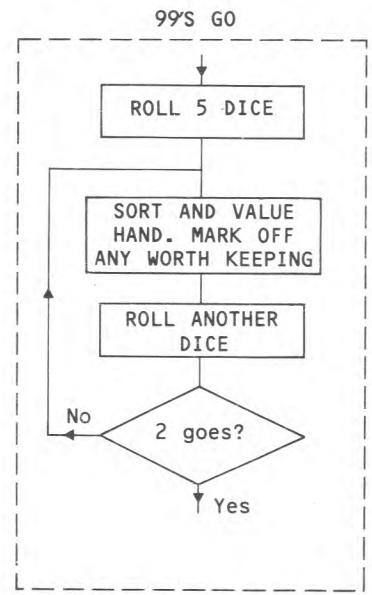


Figure 35

Now for the tricky bit. The 99's go. The flowchart for this section is shown in figure 35. The dice rolling routines are more or less the same as the human's routines, and can be left. The hardest part to program is that box marked "Sort and Value hand. Mark off any worth keeping." Look back for a moment at figure 34. When humans play poker they are able to say, at the end of the game, "3 of a kind beats any pair." or similar comments. The 99 can best tell who wins by putting number values on the hands – hence the "Bonuses".

These are worked out so that 3 of a kind will beat any pair, and so on up through the combinations. The dice have a Face Value going from 1 for a 9 up to 6 for an Ace. The poorest hand you could have with 3 of a kind would be 9 9 9 10 J. This is worth 8 (1+1+1+2+3) for Face Values, plus the Bonus of 30. Total 38. The best hand with a pair in it would be A A K Q J. Value 24 (6+6+5+4+3) plus a Bonus of 10. Total 34.

When the program reaches its final stage, and works out who has won by totalling the Values and Bonuses of each hand, part of the routine is concerned with finding pairs, or 3 or more of a kind. This is exactly what we want the 99 to do when it is "thinking" about which dice it will keep, and which it will roll again. It makes sense then to use the same routine for both purposes. Figure 36 shows the flowchart for the valuation sub-routines.

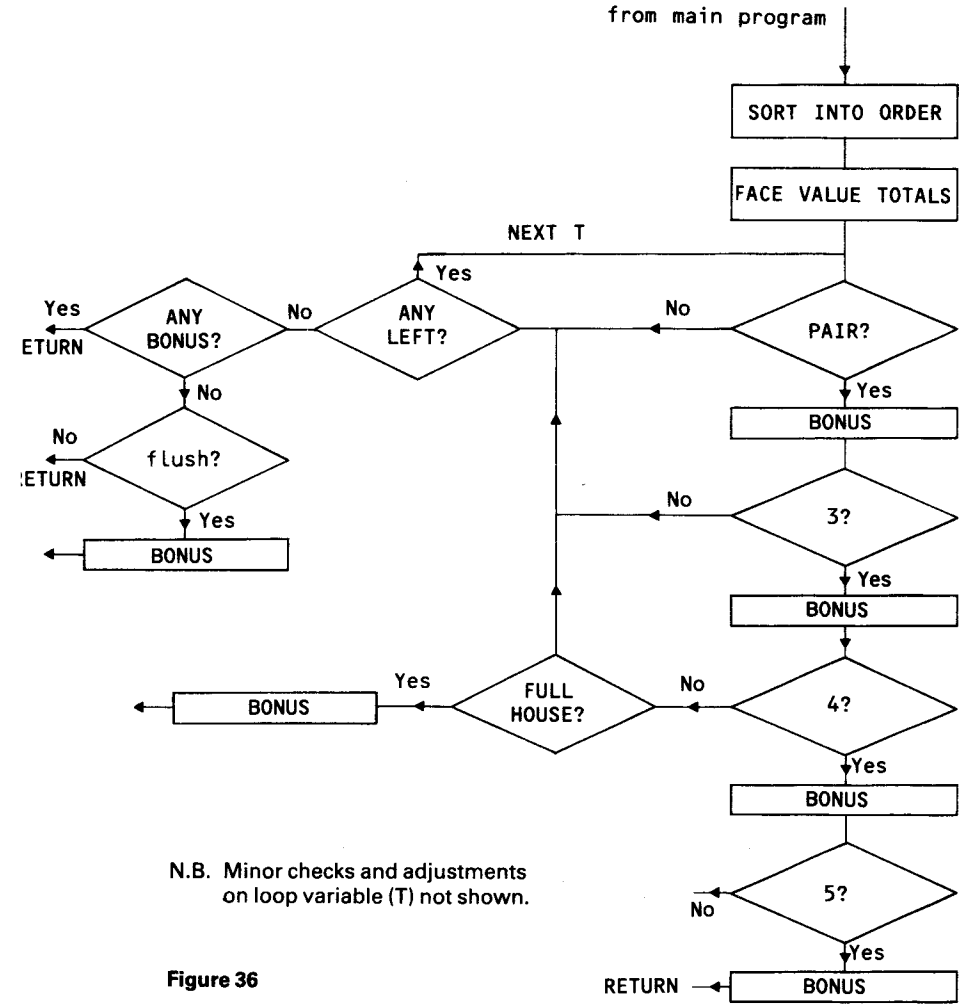
The first stage is to sort the dice into order of value. A Bubble sort routine is used for this.

```
2000  FOR T=1 TO 4
2010  FOR N=1 TO 4
2020  IF D(P,N)>=D(P,N+1) THEN 2060
2030  TS= D(P,N)      (Temporary Store)
2040  D(P,N)=D(P,N+1)
2050  D(P,N+1)=TS
2060  NEXT N
2070  NEXT T
```

You will notice that the expression in line 2020 is "more than or equal to". There is no point in swopping a pair that are the same.

We can now start to work out the value of the hand. Face Values first.

```
2080  V(P)=0      (reset the Value to 0 at the start of the
                   routine)
2090  FOR T=1 TO 5
2100  V(P)=V(P)+D(P,T)
2110  NEXT T
```



N.B. Minor checks and adjustments on loop variable (T) not shown.

Figure 36

Next, the Bonuses. These are added on to the hand value.

```
2120  REM BONUSES
2130  FOR T=1 TO 4
2140  IF D(P,T)<>D(P,T+1) THEN 2460
                              (not a pair, NEXT T)
2150  V(P)=V(P)+10      (Pair Bonus)
2160  L(T)=1
2170  L(T+1)=1
```

The array L(5) – reset to zero throughout before valuation starts – stores the numbers of any dice which are to be left. Look below to see how this fits in with the "Roll again" routine.

```
2180  IF T=4 THEN 2460
                          (jump the rest of the valuation)
2190  IF D(P,T)<>D(P,T+2) THEN 2430
                                        (not 3 of a kind)
2200  V(P)=V(P)+20      (brings total bonus to 30)
2210  L(T+2)=1
2220  IF T=3 THEN 2240
                          (check for Full House, 3 on 2)
2230  GOTO 2280      (jump to Full House 2 on 3 check)
2240  IF D(P,1)=D(P,2) THEN 2260
2250  RETURN
2260  V(P)=V(P)+30
2270  RETURN
```

Lines 2240 to 2270 are only used when T=3. To get there the 99 must have already discovered that the last 3 dice are the same (T,T+1,T+2). It is now checking to see if the first two dice also form a pair. If they do then a Full House Bonus is given. Otherwise the computer goes back to the main routine with the 3 of a kind bonus only. A similar routine is needed to check for a Full House where the pair comes after the three.

```
2280  IF T<>1 THEN 2350
                     (only do this if the first 3 are a set)
2290  IF D(P,4)=D(P,5) THEN 2310
2300  GOTO 2350
2310  V(P)=V(P)+30
2320  L(4)=1      (mark them off to leave)
2330  L(5)=1
2340  RETURN
2350  IF T>2 THEN 2450
2360  IF D(P,T)<>D(P,T+3) THEN 2450
                                        (not 4 of a kind)
2370  V(P)=V(P)+60      (bonus now 90)
2380  L(T+3)=1
2390  IF D(P,1)<>D(P,5) THEN 2450
                                        (not 5 of a kind)
2400  V(P)=V(P)+60      (total Bonus 150)
2410  L(5)=1
2420  RETURN
2430  T=T+1      (look back to line 2190)
2440  GOTO 2460
2450  T=T+2      (see line 2360)
2460  NEXT T
```

This is all a bit complicated, so let's run a few examples through to see how it works. The first hand started out as A 9 J 9 Q.

After sorting it looked like this – A Q J 9 9.

On the first three runs through the T loop, D(P,T)<>D(P,T+1), and the program jumps to 2460 for the next T. On the fourth run D(P,4) =D(P,5) and the Bonus of 10 is given. Those last two are marked off to save, and the program jumps to the end of the loop.

Here's another hand. After sorting it is like this: A K K K 10

The Kings on dice 2 and 3 are seen as a pair and the program goes through lines 2150 onwards. The third King is recognized and the 3 of a kind bonus is given by line 2200. At this stage T is 2, not 3, so line 2220 is ignored, and the program jumps to 2280. This makes it jump on again to 2350 to check for 4 of a kind. There are not 4, and the next jump takes us to line 2450 T=T+2. This is essential. Without it, the program would go through the T loop again with T=3, and it would there pick up an extra bonus of 10 for the "pair" of Kings on dice 3 and 4.

Think of a few more types of hand and trace their progress through the routine. It's the best way to see how it all works.

There is one more section that needs to be added to this valuation routine, though it is not needed by the 99 when it is working out which dice to leave, and which to roll. The player might decide to collect a flush, rather than so many of a kind. Checking for a flush is easy. The dice are already in

order of value. If they form a flush, then each die will be worth 1 more than the next – A K Q J 10.

```
2500  F=0     (Flush indicator)
2510  FOR T=1 TO 4
2520  IF D(P,T)<>D(P,T+1)+1 THEN 2540
2530  GOTO 2550
2540  F=1
2550  NEXT T
2560  IF F THEN 2580
2570  V(P)=V(P)+120
2580  RETURN
```

As long as each die is worth one more than the next, the Flush indicator remains set to 0, and the Flush Bonus is given in line 2570.

The 99 Rolls Again.

The routine which rolls the dice (lines 3000 to 3060) can be re-used for the 99's second and third rolls. Slip in a line to check the Leave-it array:

```
3005  IF L(T)=1 THEN 3060
```

If you do this, make sure that you reset L( ) to 0 throughout before you use that routine for normal rolling.

One last point. How much should the 99 bet? You will see on the flowchart that the player's hand is valued before the 99 makes his bet (player's start). The highest possible value any hand can have is 156, 5 Aces. Why not make the 99's bet 156 – HV(1) the player's hand value. When the 99 rolls and bets first, then why not one chip for every point in his hand.

This is a crude, but effective system. An alternative is to insist that the second player must match the first player's bet if he wants to stay in the game. In this case, the 99's decision to bet, when the player has started, would depend upon the player's score. When the 99 starts, he would bet only if his score was above a certain minimum – say 50. Work out your limits by running the program lots of times and collecting the range of scores. Your limit should be set just above average.

You now have the essential routines you need to write your own Pokerdice program. The line numbering suggested here does not have to be followed, of course, but is spaced to allow you plenty of room for the lines needed for the display and for organizing the 99's and the player's goes.

7
War games 1 - Co-ordinates

War Games have been around in various forms for many hundreds of years, and they include a vast range of different types of games. Chess, a stylized battle between two armies, is probably the war game that has been with us longest. Converting Chess into a computer program is a job of enormous complexity, and not a subject for a book like this. You can get some idea of the complications of a full Chess program by looking at the listings of the programs in the Chess Learner pack, in this series. There the programs only have to handle few pieces at a time, rather than the full set of 32. They are, however, written in simple TI BASIC, which means that you can list them, and they have been REM'd to help you follow what is happening. Full Chess games, like the TI Video Chess (on Solid State Module) are written in machine code. This makes them much faster, and able to handle information and calculations much quicker. The Video Chess also uses Sprite graphics for an improved presentation.

Military planners use computers in their War games in many different ways, but mainly as "number-crunchers". If you need to work out how many bombers will reach their target, you have to calculate the effect of anti-aircraft defences, enemy fighter squadrons, engine failure, and many other factors. Many of these cannot be determined exactly, but you can work within a range of possibility. The anti-aircraft defences might take out between 20% and 40% of your bombers, but less if you follow a different flight path. In that event, the fighter opposition will also be different. The computer can run through all the many combinations of possibilities in a fraction of the time it would take to work out with calculators. It can also throw in a random element to allow for human error or breakdown of equipment. When the computer has done its job, the planners will have some idea of the likely outcome of a battle, and also of some of the ways in which they can improve their performance.

The computer is also used in the design of new pieces of equipment. What balance of speed, armour, gun-size and range produce the best tank? The effect of different combinations in conflict with enemy tanks of various types can be simulated on the computer. After many hundreds, or perhaps thousands, of simulations, the planners will have a better idea of how to design their new machine.

"All very interesting," you might be saying, "but what has this got to do with war games on my 99?" The answer is, "it depends how seriously you take your gaming." If you are a real enthusiast, the sort that plays games on hexagonal boards, with thick reference books to give you the outcome of engagements between two units, with speed, range, firepower, damage status and a random factor all built in, then you will want a different sort of game from the person whose usual limit is Battleships. For the enthusiast, the best use of the 99 is as a number-cruncher. Turn your 99 into the reference manual, and play the game on the usual board with your fellow enthusiasts. The 99 can store details of the effectiveness of the players' units, and work out the result of engagements, weighting the results according to fire power, relative strength, defensive advantages, ground conditions and the usual random factors. If you are this type of war-gamer, then you will know the sort of calculations you normally have to work out. Converting them to a suitable program may be tedious to do, but will make gaming far simpler and quicker in future.

What we are trying to do here, is to turn the 99 into a worthy opponent. That is difficult enough in itself, without trying to make the game realistic as well! We will look at two games in detail – COMMANDO, one of the cassette programs, and Battleships. As war games go, COMMANDO is rather one-sided. The 99's role is always defensive, with the tactics of his soldiers being to keep on the watch and on the move. The complexities of that program are mainly in handling the multiple moves. What should make the game

interesting for the player is the difficulty of predicting future situations and planning to meet them. In Battleships, the 99 should play the game in the same way as a human.

As with any program, a war game program should start life on paper. If you are converting a board game, then the programming starts on the board. Play through the game until you know its moves by heart, until you can see its typical situations in your mind's eye. Keep the game by you while you are developing the program, and refer back to it frequently. What is the aim of the game? What is the player trying to do? How does a player try to achieve those aims? Is the game played in exactly the same way all the time, or does it go through different stages?

Let's have a look at Battleships. This game, in case you have never seen it, is played on two grids, usually 10 by 10. Each grid represents an area of sea, in which each player has hidden a number of ships. The object of the game is to "fire" into each other's grids, by giving the co-ordinates of squares, and to "sink" your opponent's ships, before he sinks yours. Most players start by shooting at random until they hit one of the enemy ships. These ships may occupy anything from one to 4 squares each. To sink a 3-square ship, you have to hit each of the 3 squares. Once you have found a hit, therefore, you will normally shoot around it until you hit it again, and so on until it is sunk. You can teach the 99 to do exactly that.

The outline flowchart for a Battleships program is shown in figure 37; the board for the game, in figure 38. You will see that the grid has been divided off into 10 squares by 10, using thin lines. The 99 has, of course, no high-resolution graphics, so that you cannot actually draw thin lines.

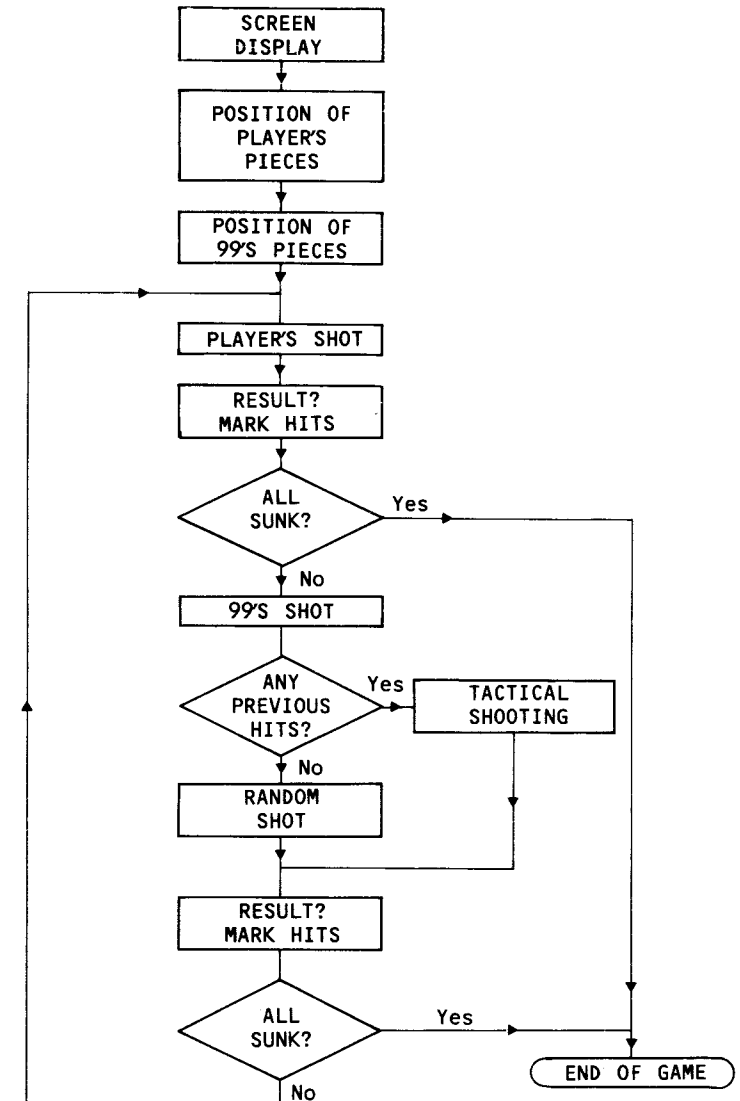


Figure 37

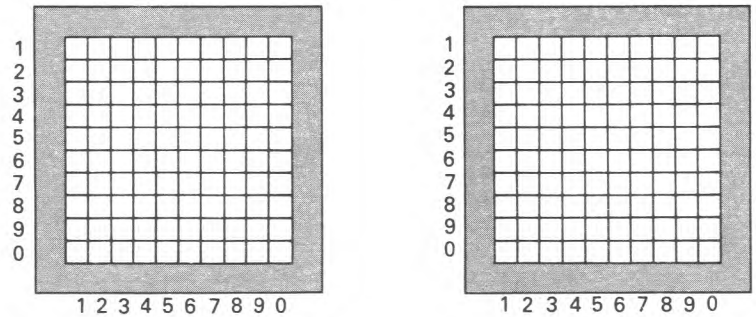
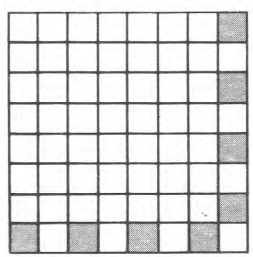


Figure 38

You can, however, create your own "grid" graphics. This is done on the COMMANDO program. The line:

```
CALL CHAR(128,"0100010001000 1AA")
```

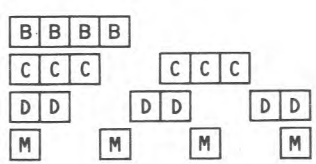
produces this character.



Put lots of those together and you have a grid. An outside edge of solid blocks completes the picture. The two grids can be drawn using just a few HCHAR and VCHAR commands.

In the version of the game we are using here, each player has 10 ships.

1 Battleship
2 Cruisers
3 Destroyers
4 Patrol Boats



The simplest way to collect the information about the position of the player's pieces, is to ask him to enter them, one square at a time. The ships are then marked on the screen.

This routine would mark on the 2 Cruisers.

```
FOR T=1 TO 2
PRINT "CRUISER ";T
FOR N=1 TO 3
PRINT "SQUARE ";N
INPUT "ROW NUMBER ? ":R
INPUT "COLUMN NUMBER ?":C
```

(The numbers added to R and C push the grid down and away from the corner. X is the code of whatever graphic you are using)

```
CALL HCHAR(R+2,C+5,X)
NEXT N
NEXT T
```

If you are likely to be dealing with awkward players, then you will need to write extra checks into this to make sure that the Row and Column numbers are within the range for the game, and also that the squares that make up a ship are in line.

The 99 will position its ships at random. In some versions of the game, diagonal placing are allowed – indeed, there is nothing here to stop the player from putting his ships diagonally – but for simplicity, the 99 will stick to either horizontal or vertical placings. This routine works out the position for the battleship. It is not, however, marked on the screen – that would ruin the game wouldn't it. Instead, the ship's position is marked in an array.

```
IF RND>.5 THEN.....     (jump next 6 lines)
R=INT(RND*10)+1
C=INT(RND*7)+1
FOR N=0 TO 3
B$(R,C+N)="B"
NEXT N
GOTO.....     (next ship's routine)
```

This finds a startpoint for the Battleship on any row, between columns 1 and 7, and marks of that square, and the next 3 horizontally after it. A similar routine produces a vertical positioning.

```
R=INT(RND*7)+1
C=INT(RND*10)+1
FOR N=0 TO 3
B$(R+N,C)="B"
NEXT N
```

Variations on this routine can be used to dot the rest of the ships about the board. Later routines must, however, include lines to make sure that the ships don't crash into each other. Write in a line, just before the point where the array is marked, to check that the array is empty at that point. If it is not, then the 99 must go back and try another pair of co-ordinates.

The Player's Shot

Unless you are working in TI EXTENDED BASIC, with its ACCEPT AT command, you are going to have to collect the co-ordinates for the players' shot through CALL KEY lines. There is a minor problem here. On a 10 by 10 grid, you are faced with a 2-digit number – 10. there are several ways to deal with this. You can identify the last Row and Column with a 0, then write an adjusting line into the routine. These lines collect the Row number:

```
CALL KEY(3,K,S)
IF S=0 THEN....      (wait for a contact)
R=K-48      (turns ASCII code into number)
IF R>0 THEN....      (jump next line)
R=R+10      (turns 0 into 10)
```

You can create an imitation input:

```
1000  CALL KEY(3,K,S)
1010  IF S=0 THEN 1000
1020  IF K=13 THEN 1060
                          (ENTER has been pressed)
1030  IF (K<48)+(K>57) THEN 1000
                              (accept numbers only)
1040  R$=R$ & CHR$(K)
1050  GOTO 1000
1060  R= VAL(R$)
```

R$ – set to "" before this routine starts – collects the numbers entered in the CALL KEY line. The routine could be used to collect numbers of any size, though here you are only concerned with two digits. You would need to add a further check line to prevent any number larger than 10 slipping through. (See the imitation input in the AIRSHIP program.)

There are two simpler solutions to this problem. Identify your grid with letters rather than numbers, or use a 9 by 9 grid.

However you decide to tackle it, you must finish up with the Row and Column co-ordinates of the player's shot. This can then be compared with the array that represents the 99's sea. The squares there will be empty except for those marked with "B","C","D" or "P" for the different ships. The presence of any letter will mean a hit, and this should be reported back to the player – perhaps by flashing that square on the screen, and marking on an "H". You also need to show whether the ship is sunk or not. With the Patrol Boats, there is no problem, because one hit sinks. With the larger ships you need some means of counting how many hits have been scored on each ship.

As always, there are several possible solutions, some better than others. One way would be to collect the letter from the hit square into a Hit string.

```
H$= H$ & B$(R,C)
```

After the first Hit on a battleship, H$="B". After 4 hits, H$"BBBB". The hit string could be compared with the possible ship strings after each hit.

```
IF H$="BBBB" THEN.....
IF H$="CCC" THEN....
IF H$="DD"....
```

When the hit string and a ship string match, a "Sunk" report is given, and H$ is emptied ready for the next time.

This routine will work perfectly well for most of the time. Every now and then, however, you will have a situation where two ships are next to each other, and the first hit is on one ship, and the second on the other. The routine cannot cope with this.

You can improve it by using a POS comparison, rather than a simple "=".

```
IF POS(H$,"BBBB",1) THEN.....
```

Now it will pick up the presence of a sinking within a more complex sequence of hits.

You will then need a clever bit of string slicing to remove the sunken ship from the hit string.

The 99's Shot

Random shooting is easy.

```
R=INT(RND*10)+1
C=INT(RND*10)+1
IF S$(R,C)<>"" THEN......      (tried that already,
                                new RND numbers)
```

S$ is the array where the 99 remembers what is happening on the player's Sea. To tell the player where the 99 is shooting, the program can either display the Row and Column numbers using a Print anywhere routine, or it could flash the shot on the screen. The routine below finds the character in that square on the player's grid, and prints that and an asterisk alternatively until the player responds with a Miss, Hit or Sunk report.

```
CALL GCHAR(R,C,Z)
CALL HCHAR(R,C,42)      (print asterisk)
CALL SOUND(50,500,1)    (beep)
CALL KEY(3,K,S)
IF S<>0 THEN....        (jump out of loop)
CALL HCHAR(R,C,Z)       (print original character)
CALL SOUND(50,500,1)    (beep again)
GOTO.....       (print asterisk line)
```


If the 99 scores a Hit, then it will want to try some tactical shooting next time round. Transfer the co-ordinates of the square to a store, and activate a hit marker.

```
HR=R
HC=C
HM=1      (Hit Marker)
```

Tactical Shooting

Finding a two-square ship is easy. All you have to do is fire away at the squares around the original hit. If the hit was on Row 5, then the second square must be on Row 4,5 or 6. You need to add −1,0 or +1 to the Hit Row.

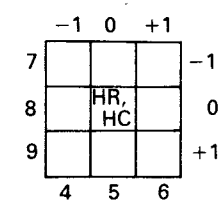


Figure 39

```
R=HR+INT(RND*3)−1
```

A similar line gives you the Column number.

Check that your new R and C numbers are within the range of the array, and check the array to make sure that the square has not been tried already. This will find a second hit just as well as any human could do it.

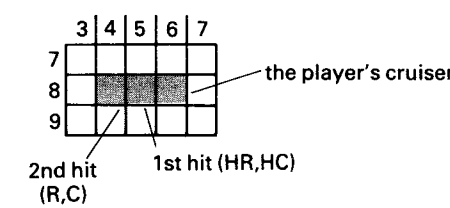


Figure 40

Ships of 3 or 4 squares present different problems, and require different solutions. Look at figure 40. The first hit

was on square 8,5. The second on square 8,4. A human player can see immediately that the rest of the ship must be on 8,3 or on 8,6. If you leave the 99 with routines given so far, and no more, then it would now start potting away around 8,4 – and never find anything. Skip the rest of this section if you want to give the human the advantage in this game.

Send the program off to this routine when the 99 scores a second hit.

```
DR=HR-R
DC=HC-C
HM=2
```

DR is the Difference between the two row numbers. In figure 40, the difference is zero. DC (the Difference in Column numbers) would here be 1.

Next time the 99 has a shot, the Hit Marker will send it off to a new tactical shooting routine.

```
R=HR+DR
C=HC+DC
```

In the example given, R would be 8, and C would be 6. Bull's eye!

However, if the third square of the Cruiser had been 8,3, and not 8,6, the 99 would have missed. To cope with this situation, you will need to move the Hit Marker on one more, and try another tactical shot next time. Here it is:

```
R=HR-2*DR
C=HC-2*DC
```

This produces the numbers R=8 and C=3. Bang on, at last.

It's all very fiddly, but such is the nature of computer games. However, that extra effort has produced a program that plays like a human. You will need a further variation on these routines to handle the 4-square ship.

Add in simple counters to keep track of the scores, check lines to cover those "All Sunk?" diamonds on the flowchart, and some good sound effects, and your Battleships program

is more or less complete. The next section is included for interest only, and should not be used in your program.

The Intelligent Computer

"Intelligence" here means the same as it does in the expression "Military Intelligence". "Spying" is a more accurate name for it. There is nothing to stop the 99 from using the CALL GCHAR routine to find the player's ships. Nothing, that is, except the honesty of you, the programmer, and the fact that if it's too good a player, humans might get suspicious. Used with discretion, the occasional check ahead to see if a random shot is worth doing, or a quick scan across a line to see if anything is there, would scarcely be noticed. You could include a check line (if N$ = "HONEST SID" THEN...) to make it jump its cheating routine when playing you. It is, however, a well-known fact that 99 owners are extremely honest, and there is therefore no point in going into this any further.

8
War games 2 - Movement

A simplified flowchart for the COMMANDO game is shown in figure 41. Much of the programming is quite obvious – see the LIST in the Appendix – but it is probably worth looking a little more closely at the way in which the player's and the 99's moves are handled.

The Player's Moves

Several arrays are used in this game, and they are all brought into play at some point during the player's go. The key ones are B$(18,16) which maps the Board, U(3,2) which holds Row and Column numbers for the 3 Units (the player's pieces), and O(3,6) which can hold up to 6 Orders for the 3 Units. The orders are collected by the routine between 1200 and 1490. (See the LIST). You will notice that the first thing to do is to check that the Unit is still available.

```
1220  IF U(T,1)=0 THEN.....
```

(jump to next Unit)

The removal of a Unit is flagged by changing its Row number to 0. The orders are given in letter codes, and these are changed to simple numbers for storage in the Order array.

```
1340  CALL KEY(3,K,S)
1350  IF S=0 THEN 1340
1360  IF (K<65)+(K>75) THEN 1330
```

(beep and try again)

```
1370  O(T,N)=K-64
```

The orders are carried out by the next section, 1500–1900. The line:

```
1560  ON O(T,N) GOTO.......
```

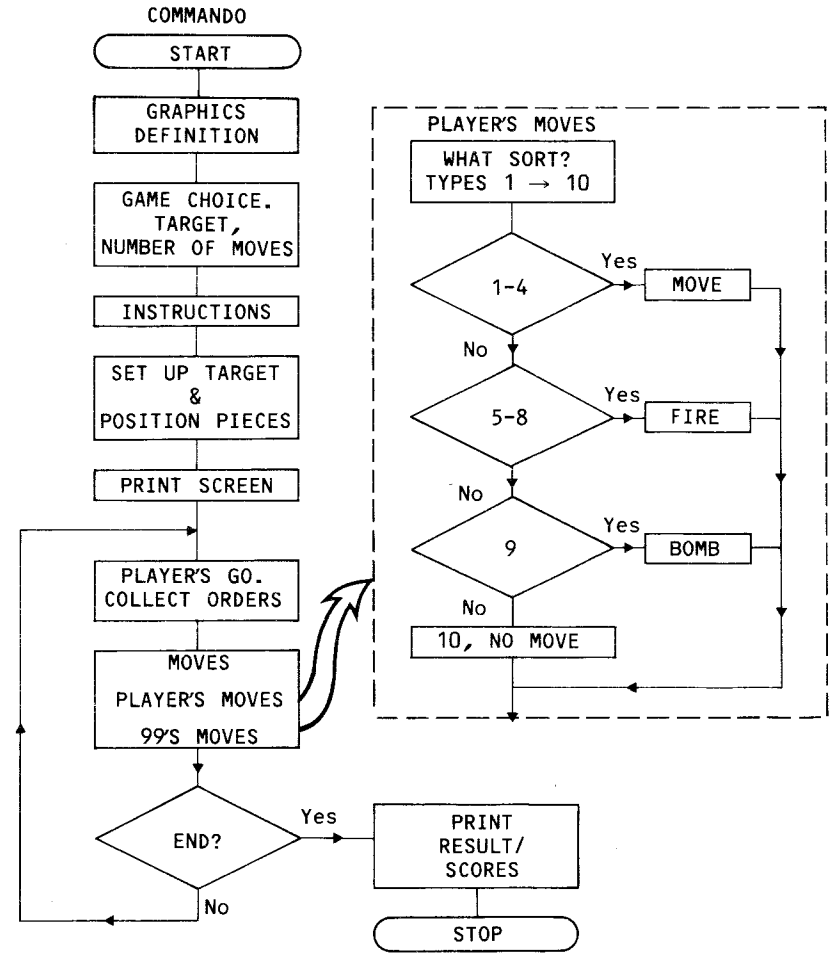


Figure 41

sends the program off to the appropriate routine. Where the order is to change position, then lines like this:

```
1570  U(T,2)=U(T,2)-1-B$(U(T,1),U(T,2)-1)
      <>CHR$(144))
```

will make the move, but cancel it again automatically if the square to be moved to is not a space. (CHR$(144) is the grid character).

Firing is managed through a sub-routine, but before going there, the Direction of fire has to be given.

```
1670  D=4
1680  GOSUB 3000
```

This Direction variable must be either 1,2,3 or 4. It would be possible to allow your fighters to fire diagonally as well. Possible, but more complicated.

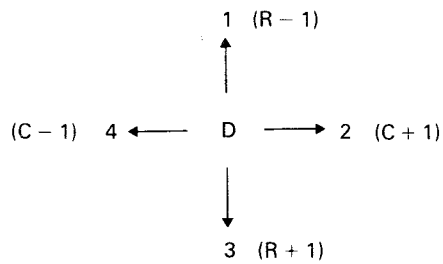


Figure 42

The firing sub-routine is used for both the player's Units and the 99's Guards. The Unit's (or the Guard's) co-ordinates are transferred to simple R and C variables, at the bullet's start point. The bullet will then travel for up to 6 squares in the direction given by D.

```
3100  FOR S=1 TO 6
3110  R=R+(D=1)-(D=3)      (up or down)
3120  C=C+(D=4)-(D=2)      (left or right)
```

There are a number of check lines that send the program off different ways according to what is on the square that the bullet is about to go through. If it's a space, then carry on (line 3130). A building, or the screen edge, will stop the bullet (3140). If it's a Unit, we must find out which Unit and knock it out. (Line 3150 and the routine from 3160 to 3270). Likewise, a Guard must be identified and removed. (3280 and the sub-routine from 4500 onwards.) In both cases, the bullet's Row and Column numbers are compared with those of each of the Units (or Guards). If they are the same, then

that square is flashed to show the hit; the same point in the array is made into a blank, and the Unit's (or Guard's) Row number is changed to 0, to indicate its removal from the board.

Bombs are handled through another array, B(10,3). This stores the co-ordinates and a timer for each of 10 bombs. When a bomb has been planted, a Bomb Signal (BS) is switched on. This tells the program to go to the "Explosion?" sub-routine each time it goes through the main order loop.

```
1840  IF BS=0 THEN 1900
1850  GOSUB 4000
```

At this sub-routine, the 99 adds onto the "timer" of each bomb that has been planted, and if the timer has reached a certain point, it jumps to an explosion routine.

```
4000  FOR Z=1 TO 10
4010  IF B(Z,1)=0 THEN 4410
                    (bomb not planted, or exploded already)
4020  B(Z,3)=B(Z,3)+1
4030  IF B(Z,3)<18 THEN 4410
                              (don't explode yet)
```

Part way through the game, the Bomb array might look like this:

Bomb Number	1(Row)	2 (Column)	3 (Timer)	
1	0	4	18	Was at 6,4. Now exploded.
2	7	12	17	At 7,12, about to blow.
3	10	4	8	At 10,4. Timer nearly half-way.
4	0	0	0	Not yet set
5	0	0	0	Not yet set.

The 99's Moves

These are handled by the lines from 1910 to 2300. These Guards are supposed to be on sentry duty, so their main task is to keep their eyes open for intruders. Fortunately for the player, they don't always do that. A line is written in so that some of the time their eyes are shut.

```
1940  IF RND>.7 THEN 2300
```

This random limit can be changed to make the game easier or harder, as you like. The search routine is run through two loops, covering all 10 guards (the T loop) and the 3 Units (the Z loop). Here's the section that checks to see if there is a Unit on the same column as a Guard, and if it is in range:

```
1960  IF G(T,2)<>U(Z,2) THEN 2020
                (not on same Column check the Row next)
1970  IF ABS(G(T,1)-U(Z,1))>6 THEN 2100
                                        (out of range)
1980  D=1     (shoot up)
1990  IF G(T,1)>U(Z,1) THEN 2080
                                (go to firing routine)
2000  D=3
2010  GOTO 2080
```

The first two lines have checked that a target is there somewhere, the question is, "is it above or below the Guard?" This is checked by line 1990. If the Unit is upscreen, then the Direction variable is left at 1, and the 99 goes to the firing sub-routine. If the Unit is downscreen, then the Direction variable needs to be reset before firing.

Having had their look around, and perhaps fired at an intruder, the Guards now continue their sentry duty and march on. The direction in which each is to move has been fixed, at random, at the start of the program. It is held in the 3rd store in the Guards' array. The Guards will march in their allotted direction until they bump up against something. (See lines 2130 to 2260). If they can go no further, then they will turn.


```
2270  G(T,3)=G(T,3)+1
2280  G(T,3)=G(T,3)+(4*(G(T,3)>4))
                        (keep D in range 1–4)
```

This produces a quarter turn clockwise. The amount of turn could be randomised instead, to make their movements less predictable.

At the moment, if a Guard find a Unit on the same Row (or Column), but out of range, it will simply ignore it. This could be altered to make the Guards more menacing. Write in a routine to make the Guard move closer to the Unit, rather than following its normal patrol route. The Guards could also be allowed to pass through buildings, if you felt they needed any extra advantages.

The 99 in this game is a good tactical player – it can always make a sensible move, but it has no real strategy – it does not plan ahead. For examples of strategic play, you could look at the listings of the programs in the Chess Pack in this series.

9
Simulations

The object of a simulation program is to get the computer to produce a copy of the real world, or at least, of one small part of it. A good simulation will present you with the same problems that you would get in reality. Flight simulators are much used in the training of civil and military aircraft pilots. It costs millions to crash a Jumbo Jet, but nothing to "crash" a computer program! War games are simulations so are business games. Simulation programs are also used in scientific research, economic planning and weather forecasting. The key point to bear in mind about a simulation program is that it will only be as good as you knowledge of the subject – which might explain why AIRSHIP will be of little value in training airship crew!

AIRSHIP could be made into an effective simulation, given a fuller understanding of the various factors at work and sufficient memory space. The wind is purely random at the moment. It should correspond to likely weather conditions in this part of Europe. It should also vary with height, and the safe flying height should also vary according to whereabouts on the map you are supposed to be. To do this, you would have to build a 3-dimensional map into the program, and check the airship's position on there as it flies.

These would be significant improvements, but the major change that the program needs is the inclusion of a "real-time" element. AIRSHIP works in set units of time. You give your orders for the next hour, and that's it. It doesn't matter, either, how long it takes you to work out those orders – the airship will hang suspended, and unmoving, until you are ready. To make this change you would need to be able to alter speed, height and bearing at any time, through the keyboard. This is not really feasible in TI BASIC. The program would have to run continuously round a loop that recalculates and displays all the speed, bearing and distance information, passing through CALL KEY lines, but not stopping there. (See figure 43).

The calculations are done rapidly, but the display of the variables takes time when you have to use an HCHAR – Print anywhere – routine. As a rough guide, calculations of the type "AX= SIN(AB*.017)*AS" (line 1610) are performed at the rate of about 9 a second. Displayed with a normal PRINT command, the rate is around 5 a second. Put that display through the "Print Anywhere" routine, and you are down to 3 a second. The AIRSHIP program has 12 variables that need continual updating and display. To ask your user to catch the one brief moment of a CALL KEY line in a 4 second loop, is to ask too much. Working in TI EXTENDED BASIC, with its DISPLAY AT command, and faster working, you would be able to reduce the loop time to a little over 1 second. You could reduce it further by cutting down on the display – but is it worth it?

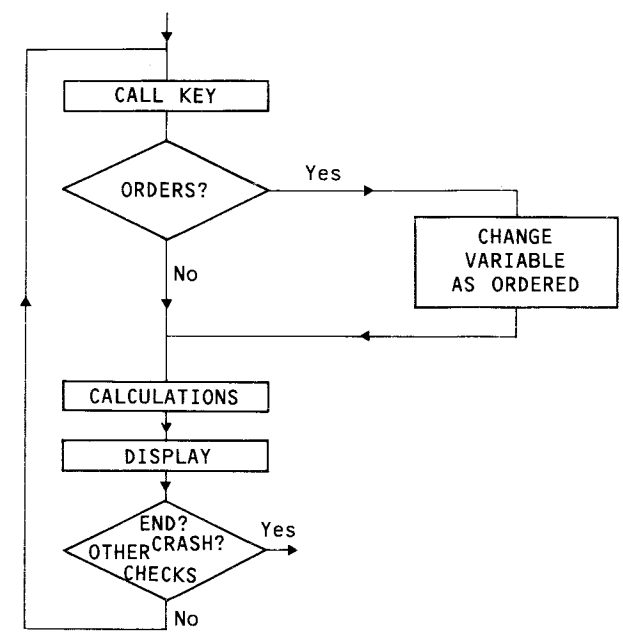


Figure 43

If you want to produce a reasonable spaceflight, or jet flight simulation, you are going to have to transfer to machine code to get the necessary speed.

Meanwhile, let's have a closer look at the AIRSHIP program, as it shows the use of some of the 99's trigonometry functions, SIN,COS and ATN.

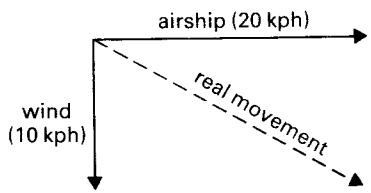


Figure 44

As you probably saw on the demonstration early in the program, if the airship is flying due East at 20 kph, and the wind is blowing due South at 10 kph, then you actually travel East South East at about 22 kph. (See figure 44). With simple speed figures, and the wind and the ship's direction at right angles to each other, this is fairly obvious. We can calculate the actual speed by using Pythagorus's theorem – "The square on the hypothenuse is equal to the sum of the squares on the other two sides."

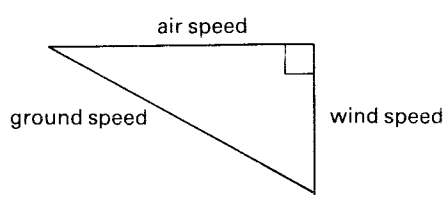


Figure 45

"Ground Speed^2 = Airspeed^2 + Windspeed^2

That translates to this equation:

```
GS = SQR(AS*AS + WS*WS)
```

In fact, you won't find the equation in the program in quite this form. Line 2130

```
D=INT(SQR(X*X+Y*Y))
```

Is a generalized form of the equation, which is used for several different calculations of the same type.

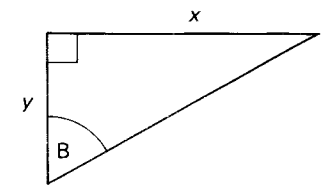


Figure 46

To find the bearing, we use the Arctangent function (ATN). This lets us work out an angle if we know the opposite and adjacent sides. In figure 46, angle B is the Arctangent of the opposite (x) divided by the adjacent (y). Of course, if we ask the 99 for the angle, it will give it to us in radians. We are trying to work in degrees, like all good navigators, so we must multiply by 57.3 to convert radians to degrees. (1 circle = 2 π radians = 360 degrees).

The final line looks like this:

```
2380  B= INT(ATN(X/Y)*57.3)
```

It is not very accurate. If you refer to your User's Guide, you will see that TI recommend that you multiply by 57.295779513079 to make the conversion. Integerizing numbers also plays havoc with your accuracy, but decimals make a mess of your screen.

There's more to this, but let's look back at the "real travel" calculations for a moment. Suppose at the start of your flight, there was a wind blowing on a bearing of 60° at 30 kph. You want to travel on a bearing of 330. Which way should you point your ship? (figure 47). You guess that a bearing of 300

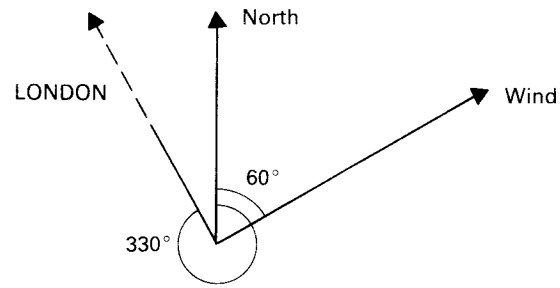


Figure 47

degrees and a speed of 100 kph should do the trick. So what happens?

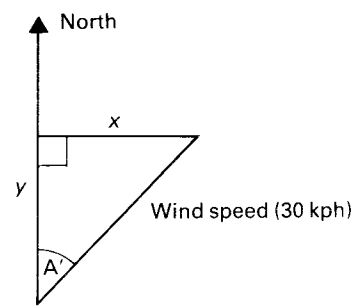


Figure 48

First of all, how far East/West and North/South will a wind of 30 kph on a bearing of 600 degrees blow you? If you were working this out on paper, you would draw in the right-angled triangle (figure 48) and find the angle A (60°).

Look what happens when the bearing is over 90 degrees. You then find X (East/West) by multiplying SIN A by 30. The 99 can work out the sines of any angle, so the line

```
X=SIN(WB*.017)*WS
```

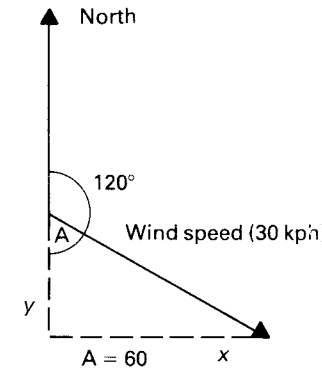



Figure 49

gives us the answer, without having to bother to draw triangles, or bring the angle down to 90 or less. In this example SIN(60*.017)*30 is 25.6. You will be blown 25.6 km East. The COSINE of the angle lets us work out the North/South (Y) movement.

```
Y=COS(WB*.017)*WS
```

Here COS(60*.017)*30 gives 15.7. That's 15.7 km North. Notice in both of the equations we have to include "*.017" to convert degrees to radians for the 99. If you want greater accuracy, use *.01745329251994.

We can perform the same calculations on our airship's speed and bearing, and find that (in still air) these would produce movements of 92.5 km West (X=−92.5) and 37 km North. (Y=37). If we combine the two lots of figures, we get the movement of the airship, in relation to the ground.

```
1650 X=AX + WX     (here, −92.5+25.6 = −66.9)
1660 Y =AY + WY    (here, 37+15.7 = 52.7)
```

The wind has cancelled some of the westward movement, but has increased the total shift to the north. We can now use the same calculations we used earlier, to find the total diagonal movement and bearing. In this case, we have a ground speed of approximately 82 kph, and have flown on a bearing of −51 degrees. −51?? That can't be right!

It isn't. While the SIN and COS functions will work right round the circle, from 0 to 360 degrees, the ATN function will always give you a result between 0 and 90, and either positive or negative. Look at the equation in simple form:

```
B= ATN(X/Y)
```

If either X or Y are negative, then the result will be negative. If both are positive, or both negative, then the result is positive. In the figure below, you can see how this varies according to the values of X and Y. In AIRSHIP the Arctangents are calculated in the routine from 2320 to 2430. Only one line performs the actual calculation, the rest are there to allow for all the different possible values that might occur.

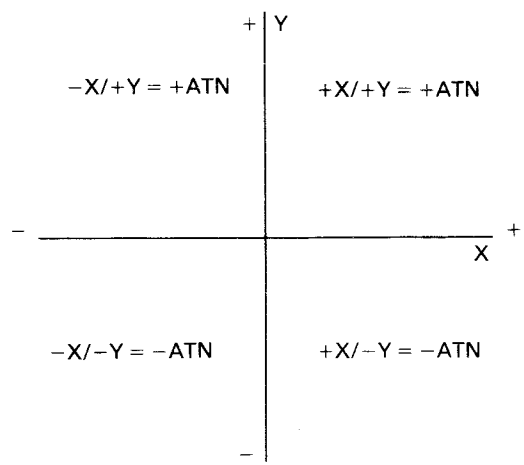


Figure 50

Dry Running

If you are a mathematical wizard, then you will be able to get equations sorted out correctly first time, and any minor typing errors you have made will show up as soon as you run the program. If, like the writer, you have only a limited grasp of higher mathematics, then "Dry Running" is essential.

Work out your equations as carefully as you can – possibly writing little programs to test the effect of various functions. Now write them out in program lines, and then take some simple figures (ones where you can work out the approximate answer on paper, or with a hand calculator) and go through your lines with those figures. Will the program you have written produce the results you expect? If not – back to the drawing board. You should have a better idea of where the problem is, if nothing else.

Test Data

Having checked your program on paper, type it in and test again. Don't use the random function at this stage. Set the variables to figures where you know the answer and run them through. Does the program give the right answer? Try with a variety of figures, ones which you will test all of the lines of the routine.

With something like the arctangent function you would want to test all the possible positive/negative combinations – comparing your results with paper sketches or calculations. It is also worth writing a temporary loop into your program so that the same calculations are performed with all but one of the factors held constant. Fix the speed and run through the bearings from 0 to 360 (in steps of 5 or 10, or you will be there all night).

TRACE in cases of difficulty. It ruins the screen display, but at least you can tell if the 99 is going through the program lines in the way that you think it should. If you have a printer, then you can follow the trace numbers on the print out. If not, then you will have to do it the other way. Write down the trace sequence for that section that worries you, then compare that with the LIST on the screen later.

Other Simulations

Why not write your own business simulation? A good one need not be complicated, and they can be fun games to play. At the simplest, you would have only one product, and you

would take only the cost of production and of advertising into account.

Key factors would be how much cash you have to start with; how much it costs to produce 1 whatever-it-is; how far price affects sales (and profits); the effect and cost of advertising. You can refine the program later to bring other factors into play. Do you have any competitors, and what are they doing? What's happening to the price of raw materials? If your product is something like lemonade or ice cream, then the weather is important.

Figure 51 shows, as a diagram and not as a flowchart, the interplay of some of the factors in a business game.

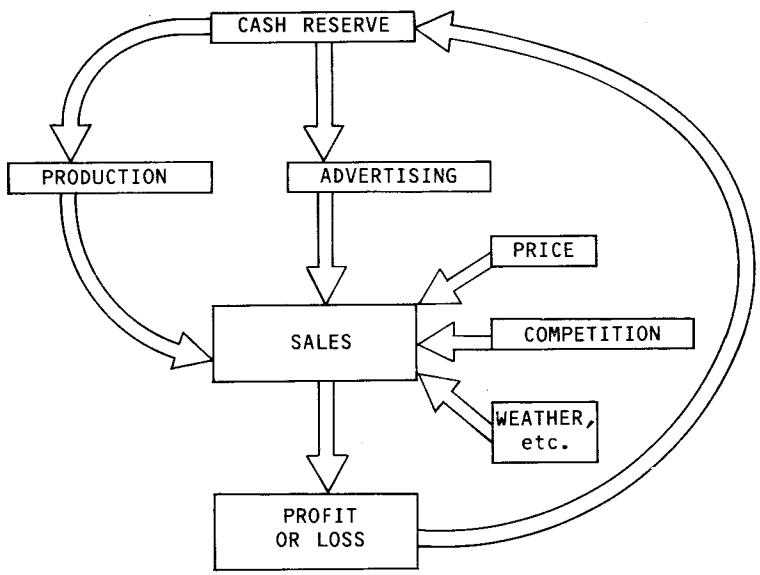


Figure 51

This type of game may be best played by several people at once, each in competition with the other.

The best simulations to write are those where you understand the subject, and where you can give a straightforward number value to each factor. It is very difficult to write a good simulation that has to take humans into account.

Appendix
Program lists

DICERACE

```
10 REM  DICERACE
20 REM  MACBRIDE 1983
30 CALL SCREEN(8)
40 CALL CLEAR
50 OPTION BASE 1
59 REM  n$(4)= players names
60 DIM N$(4)
69 REM  p(4,3)=counters row, col
umn and character
70 DIM P(4,3)
79 REM  d$(6,3)=dice graphics

80 DIM D$(6,3)
90 PRINT ::"       DICERACE"::
100 PRINT " THIS IS A GAME FOR M
E AND"::" UP TO 3 HUMAN PLAYERS.
"::
110 PRINT ::" DEFINING GRAPHICS"
::" PLEASE WAIT A MOMENT."::
120 FOR N=1 TO 7
130 READ X,G$
140 CALL CHAR(X,G$)
150 NEXT N
159 REM  dice and edges
160 DATA 120,00183C7E7E3C1800,12
1,FF99A5DBDBA599FF,122,0
164 REM  counters
165 DATA 128,C381000000081C3,13
6,C3810000000081C3,144,C38100000
00081C3,152,C381000000081C3
170 FOR N=12 TO 16
180 READ F,B
190 CALL COLOR(N,F,B)
200 NEXT N
210 DATA 2,16,8,5,8,7,8,3,8,14
220 INPUT "HOW MANY PLAYERS 1-3
?":PN
225 IF PN>3 THEN 220
230 FOR N=2 TO PN+1
240 PRINT "PLAYER NUMBER ";N-1
250 INPUT "NAME PLEASE ":N$(N)
259 REM  make sure name fits
260 IF LEN(N$(N))<12 THEN 280
270 N$(N)=SEG$(N$(N),1,11)
280 NEXT N
289 REM  i'm the first player

290 N$(1)="ME"
295 PN=PN+1
300 PRINT " PRESS ANY KEY TO STA
RT "
310 CALL KEY(3,K,S)
320 IF S=0 THEN 310
325 RANDOMIZE
330 CALL CLEAR
339 REM  the board
340 FOR R=1 TO 22 STEP 3
350 CALL HCHAR(R,2,121,31)
360 NEXT R
370 FOR C=2 TO 32 STEP 3
380 CALL VCHAR(1,C,121,22)
390 NEXT C
400 FOR R=5 TO 18
410 CALL HCHAR(R,6,32,23)
420 NEXT R
425 REM  dice graphics
429 REM  row of blanks
430 E$=CHR$(122)&CHR$(122)&CHR$(
122)
439 REM  spot in centre
440 F$=CHR$(122)&CHR$(120)&CHR$(
122)
449 REM  spot on right
450 G$=CHR$(122)&CHR$(122)&CHR$(
120)
459 REM  spot on left
460 H$=CHR$(120)&CHR$(122)&CHR$(
122)
469 REM  spots both sides
470 I$=CHR$(120)&CHR$(122)&CHR$(
120)
480 D$(1,1)=E$
490 D$(1,2)=F$
500 D$(1,3)=E$
510 D$(2,1)=G$
520 D$(2,2)=E$
530 D$(2,3)=H$
540 D$(3,1)=G$
550 D$(3,2)=F$
560 D$(3,3)=H$
570 D$(4,1)=I$
580 D$(4,2)=E$
590 D$(4,3)=I$
600 D$(5,1)=I$
610 D$(5,2)=F$
620 D$(5,3)=I$
630 D$(6,1)=I$
640 D$(6,2)=I$
650 D$(6,3)=I$
659 REM  start positions
660 FOR N=1 TO 4
670 READ P(N,1),P(N,2),P(N,3)
680 NEXT N
690 DATA 21,3,128,20,4,136,20,3,
144,21,4,152
700 FOR N=1 TO PN
710 CALL HCHAR(P(N,1),P(N,2),P(N
,3))
720 NEXT N
729 REM  main game loop
730 FOR N=1 TO PN
740 W$="PLAYER - "&N$(N)
750 R=7
760 C=7
770 GOSUB 6000
779 REM  random dice number
780 Z=INT(RND*6)+1
790 GOSUB 4500
799 REM  i don't press
800 IF N=1 THEN 910
840 W$="PRESS ANY KEY TO ROLL DI
CE"
850 R=23
860 C=2
870 GOSUB 6000
880 CALL SOUND(250,550,1)
890 CALL KEY(3,K,S)
900 IF S=0 THEN 890
905 CALL HCHAR(23,3,32,26)
909 REM  rolling dice
910 FOR Z=1 TO 6
920 CALL SOUND(100,110*Z,1)
930 GOSUB 4500
940 NEXT Z
```

```
949 REM  random dice number
950 Z=INT(RND*6)+1
960 GOSUB 4500
970 FOR T=1 TO Z
980 CALL HCHAR(P(N,1),P(N,2),32)
989 REM  right-hand side
990 IF (P(N,2)>29)*(P(N,1)>4)THE
N 1040
999 REM  top
1000 IF (P(N,1)<4)*(P(N,2)>4)THE
N 1060
1009 REM  left-hand side
1010 IF (P(N,2)<5)*(P(N,1)<20)TH
EN 1080
1019 REM  bottom - go right
1020 P(N,2)=P(N,2)+3
1030 GOTO 1090
1039 REM  right - go up
1040 P(N,1)=P(N,1)-3
1050 GOTO 1090
1059 REM  top - go left
1060 P(N,2)=P(N,2)-3
1070 GOTO 1090
1079 REM  left - go down
1080 P(N,1)=P(N,1)+3
1090 CALL HCHAR(P(N,1),P(N,2),P(
N,3))
1094 REM  check for end
1095 GOSUB 2000
1100 NEXT T
1110 CALL HCHAR(7,8,32,20)
1119 REM  incident routines
1120 GOSUB 3000
1130 NEXT N
1140 GOTO 730
1999 REM  reached end?
2000 IF (P(N,1)>19)*(P(N,2)<5)TH
EN 2100
2090 RETURN
2100 M$=N$(N)&" HAS WON!"
2110 R=23
2120 C=5
2130 GOSUB 6000
2140 M$=" ANOTHER GAME ?(Y/N)"
2150 R=24
2160 C=3
2170 GOSUB 6000
2180 CALL SOUND(500,250,1)
2190 CALL KEY(3,K,S)
2200 IF S=0 THEN 2190
2230 IF K=89 THEN 300
2240 IF K<>78 THEN 2180
2250 CALL SCREEN(8)
2260 CALL CLEAR
2270 PRINT TAB(7);"PROGRAM INDEX
":
2280 PRINT "  SET UP ARRAYS.....
....50":
2290 PRINT "  DEFINE GRAPHICS...
...120":
2300 PRINT "  COLLECT NAMES.....
...220":
2310 PRINT "  BOARD AND DICE....
...330":
2320 PRINT "  GAME STARTS HERE..
...659":
2330 PRINT "  DICE ROUTINES.....
...779":
2340 PRINT "  MOVE-WHICH WAY ?..
...970":
2350 PRINT "  END ? INCIDENT ?..
..1094":
2360 PRINT "  DISPLAY ROUTINES..
..4499"
2370 END
3000 REM  write your own
3010 REM  incident routines
3020 REM  here.
3990 RETURN
4499 REM  wipe out dice face
4500 FOR T=1 TO 3
4510 CALL HCHAR(9+T,16,122,3)
4520 NEXT T
4999 REM  draw new dice
5000 FOR T=1 TO 3
5010 M$=D$(Z,T)
5020 R=9+T
5030 C=15
5040 GOSUB 6000
5050 NEXT T
5060 RETURN
5999 REM  writing routine
6000 FOR D=1 TO LEN(M$)
6010 X=ASC(SEG$(M$,D,1))
6020 CALL HCHAR(R,D+C,X)
6030 NEXT D
6040 RETURN
```

CROSSES

```
10 REM  CROSSES
20 REM  MACBRIDE 1983
30 CALL SCREEN(16)
40 CALL CLEAR
50 PRINT TAB(12);"CROSSES":::
60 PRINT "       ONE MOMENT PLEAS
E"
70 RESTORE 5000
80 GOSUB 5000
90 PRINT
100 REM  graphics defined
110 REM  ready to start
120 INPUT " PRESS ENTER TO START
.":A$
130 OPTION BASE 1
135 REM  screen array 3x3
140 DIM S(3,3)
145 REM  working array 8x3
150 DIM W(8,3)
160 CALL CLEAR
170 PRINT :::"    ME      DRAW";
TAB(22);"YOU"
180 PRINT ::TAB(5);I;TAB(14);D;T
AB(22);Y
185 REM  this draws the
    lines of the board
190 B1$=" "&CHR$(128)&" "&CHR$(1
28)
200 B2$=CHR$(130)&CHR$(129)&CHR$
(130)&CHR$(129)&CHR$(130)
210 PRINT ::::TAB(13);B1$
220 PRINT TAB(13);B2$
230 PRINT TAB(13);B1$
240 PRINT TAB(13);B2$
250 PRINT TAB(13);B1$
260 PRINT ::::::::
270 FOR R=1 TO 3
280 FOR C=1 TO 3
285 REM  puts letters on
    board-small caps
 to allow coloring
290 CALL HCHAR(9+R*2,13+C*2,93+3
*R+C)
300 NEXT C
310 NEXT R
320 M$="YOUR MOVE - WHICH SQUARE
 ?"
330 R=20
340 C=3
350 GOSUB 6000
360 CALL SOUND(250,1397,1)
370 CALL KEY(3,K,Z)
380 IF Z=0 THEN 370
385 REM  input check

390 IF (K<65)+(K>73)THEN 360
395 REM  convert ASCII code
    to co-ordinates
400 R=INT((K-62)/3)
410 C=K-(61+3*R)
420 IF S(R,C)>0 THEN 360
425 CALL HCHAR(20,4,32,26)
430 CALL HCHAR(9+2*R,13+2*C,88)
435 REM  mark move on array

440 S(R,C)=1
445 REM  P=number of plays

450 P=P+1
490 REM  screen array data
    to working array
500 FOR R=1 TO 3
510 FOR C=1 TO 3
520 W(R,C)=S(R,C)
530 W(3+R,C)=S(C,R)
540 NEXT C
550 W(7,R)=S(R,R)
560 W(8,R)=S(4-R,R)
570 NEXT R
580 IF P>1 THEN 700
590 REM  first move?

600 IF W(2,2)=1 THEN 640
610 IF RND>.9 THEN 700
620 W(2,2)=4
630 GOTO 1000
640 W(1,1)=4
650 GOTO 1000
690 REM  line checks start

700 M=0
705 REM  has player won ?

710 FOR R=1 TO 8
720 T=W(R,1)+W(R,2)+W(R,3)
730 IF T<>3 THEN 770
740 WIN=1
750 M=1
760 GOTO 1000
770 NEXT R
775 IF P>4 THEN 3400
778 REM  can i win ?

780 FOR R=1 TO 8
790 T=W(R,1)+W(R,2)+W(R,3)
800 IF T<>8 THEN 830
810 F=1
820 GOSUB 2000
825 GOTO 1000
830 NEXT R
835 REM  has player  got
      2 in a line ?
840 FOR R=1 TO 8
850 T=W(R,1)+W(R,2)+W(R,3)
860 IF T<>2 THEN 880
870 GOSUB 2000
880 NEXT R
890 IF M THEN 1000
895 REM  trap spotter

900 IF (RND>.9)*(W(1,1)=1)*(W(3,
3)=1)THEN 940
910 IF (RND>.9)*(W(1,3)=1)*(W(3,
1)=1)THEN 970
915 REM  still not moved ?
920 GOSUB 2500
930 GOTO 1000
940 W(2,1)=40
950 M=1
960 GOTO 1000
970 W(3,1)=4
980 M=1
990 REM  transfers move
   back to screen array
1000 FOR R=1 TO 3
1010 FOR C=1 TO 3
1020 IF W(R,C)=0 THEN 1040
1030 S(R,C)=W(R,C)
1040 IF W(3+R,C)=0 THEN 1060
1050 S(C,R)=W(3+R,C)
1060 NEXT C
1070 IF W(7,R)=0 THEN 1090
1080 S(R,R)=W(7,R)
1090 IF W(8,R)=0 THEN 1110
1100 S(4-R,R)=W(8,R)
1110 NEXT R
1120 IF WIN THEN 3000
1125 REM  puts move on screen

1130 FOR R=1 TO 3
1140 FOR C=1 TO 3
1150 IF S(R,C)<>1 THEN 1170
1160 CALL HCHAR(9+2*R,13+2*C,88)
1170 IF S(R,C)<>4 THEN 1190
1180 CALL HCHAR(9+2*R,13+2*C,79)
1190 NEXT C
1200 NEXT R
1210 IF F THEN 3200
1215 REM  game must end
     after 5th move
1220 IF P>4 THEN 3400
1230 GOTO 320
1990 REM  finds empty square
      after 2 in a line
   check.
2000 FOR C=1 TO 3
2010 IF W(R,C)>0 THEN 2060
2020 W(R,C)=4
2030 M=1
2040 C=3
2050 R=8
```

```
2060 NEXT C
2070 RETURN
2490 REM  finds first empty
      square if 99 has
   no better move
2500 IF W(1,3)=0 THEN 2600
2510 IF W(3,2)=0 THEN 2620
2520 FOR R=1 TO 3
2530 FOR C=1 TO 3
2540 IF W(R,C)<>0 THEN 2570
2550 W(R,C)=4
2560 GOTO 2590
2570 NEXT C
2580 NEXT R
2590 RETURN
2600 W(1,3)=4
2610 RETURN
2620 W(3,2)=4
2630 RETURN
3000 W$=" YOU WIN "
3010 R=20
3020 C=12
3030 GOSUB 6000
3040 CALL SOUND(1000,110,1,112,1
,114,1,-4,1)
3050 Y=Y+1
3060 GOTO 3600
3200 W$=" I WIN "
3210 R=20
3220 C=12
3230 GOSUB 6000
3240 CALL SOUND(1000,220,1,277,1
,330,1)
3250 I=I+1
3260 GOTO 3600
3400 W$=" DRAWN GAME "
3410 R=20
3420 C=11
3430 GOSUB 6000
3440 CALL SOUND(500,440,1)
3445 CALL SOUND(500,330,1)
3450 D=D+1
3600 W$=" ANOTHER GAME ? (Y/N)"
3610 R=22
3620 C=3
3630 GOSUB 6000
3640 CALL SOUND(250,1397,1)
3650 CALL KEY(3,K,Z)
3660 IF Z=0 THEN 3650
3670 IF K=89 THEN 3700
3680 IF K=78 THEN 3800
3690 GOTO 3640
3700 FOR R=1 TO 3
3710 FOR C=1 TO 3
3720 S(R,C)=0
3730 NEXT C
3740 NEXT R
3745 REM  clear screen array
      & game variables
3750 WIN=0
3760 F=0
3770 P=0
3780 GOTO 160
3800 CALL SCREEN(8)
3810 CALL CLEAR
3820 PRINT TAB(7);"PROGRAM INDEX
":::
3830 PRINT " SET UP ARRAYS.....
...130"::
3840 PRINT " DRAW SCREEN.......
...160"::
3860 PRINT " PLAYER'S MOVE.....
...370"::
3870 PRINT " S.O TO W.O TRANSFER
...490"::
3880 PRINT " FIRST MOVE.......
...590"::
3890 PRINT " LINE CHECKS.......
...690"::
3900 PRINT " W.O TO S.O TRANSFER
...990"::
3910 PRINT " UPDATE DISPLAY....
..1125"::
3920 PRINT " MOVE FINDERS....199
0&2490"::
3930 PRINT " RESET FOR NEW GAME.
..3700"::
3940 FOR D=1 TO 5000
3950 NEXT D
3960 END
5000 FOR N=128 TO 130
5010 READ G$
5020 CALL CHAR(N,G$)
5030 NEXT N
5040 DATA 1818181818181818,18181
8FFFF181818,000000FFFF000000
5050 CALL COLOR(9,5,1)
5060 CALL COLOR(10,5,1)
5070 RETURN
6000 FOR Q=1 TO LEN(W$)
6010 X=ASC(SEG$(W$,Q,1))
6020 CALL HCHAR(R,Q+C,X)
6030 NEXT Q
6040 RETURN
```

CARDS

```
10 REM  CARDS
20 REM  MACBRIDE 1983
30 OPTION BASE 1
40 DIM C$(4,13)
50 DIM H$(4,13)
60 CALL SCREEN(8)
70 CALL CLEAR
80 PRINT TAB(13);"CARDS":::
90 PRINT " THIS PROGRAM SHUFFLES
,DEALS":" AND SORTS A PACK OF C
ARDS.":::
95 REM   define graphics
100 GOSUB 5000
120 CALL KEY(3,K,S)
130 INPUT " PRESS ENTER TO BEGIN
 ":A$
140 CALL CLEAR
150 PRINT "     HERE'S THE PACK":
:
160 PRINT " C$(SUIT,VALUE)    CA
RD"
165 REM  sets up pack
170 GOSUB 3000
175 REM  how many in each
     hand ?
180 N1=13
185 REM  how many hands ?
190 T1=3
195 REM  shuffle and deal
200 GOSUB 3200
205 REM  fourth hand
210 GOSUB 3400
220 PRINT " HERE ARE THE HANDS."
230 FOR N=1 TO 4
240 PRINT ::" HAND "!N::
250 FOR T=1 TO 13
260 PRINT H$(N,T);
270 NEXT T
280 NEXT N
290 PRINT ::" THEY NEED SORTING.
":::" PRESS ANY KEY WHEN READY.":
:
300 CALL KEY(3,K,S)
310 IF S=0 THEN 300
315 REM  4 hands
320 FOR N=1 TO 4
325 REM  12 runs through
330 FOR T=1 TO 12
335 REM  12 'bubbles'
340 FOR I=1 TO 12
350 IF H$(N,I)>H$(N,I+1)THEN 400
360 X$=H$(N,I)
370 H$(N,I)=H$(N,I+1)
380 H$(N,I+1)=X$
390 GOSUB 3550
400 NEXT I
405 NEXT T
440 NEXT N
450 PRINT " PRESS ANY KEY TO GO
ON"
460 CALL KEY(3,K,S)
470 IF S=0 THEN 460
480 CALL CLEAR
490 PRINT TAB(7);"PROGRAM INDEX"
:::
500 PRINT "  DEMONSTRATION.....
...80"::
510 PRINT "  SET UP PACK.......
.3000"::
520 PRINT "  SHUFFLE AND DEAL...
.3200"::
530 PRINT "  FILL FOURTH HAND...
.3400"::
540 PRINT "  DISPLAY SUBROUTINE.
.3550"::
550 PRINT "  DEFINE GRAPHICS....
.5000"::::
560 END
3000 REM  set up pack
3010 RESTORE 3000
3020 FOR N=1 TO 4
3025 REM  sets suit graphic
3030 READ X
3040 G$=CHR$(X)
3050 FOR T=1 TO 13
3055 REM  value graphic
3060 P$=CHR$(49+T)
3070 IF T<9 THEN 3120
3080 IF (N=2)+(N=3)THEN 3110
3085 REM  high value cards
3090 P$=CHR$(112+T)
3100 GOTO 3120
3110 P$=CHR$(105+T)
3120 C$(N,T)=G$&P$
3125 PRINT TAB(6);N;TAB(11);T;TA
B(21);C$(N,T)
3130 NEXT T
3140 NEXT N
3150 DATA 120,113,112,104
3160 RETURN
3200 REM  deals hands
3201 PRINT ::" SHUFFLING AND DEA
LING."
3210 FOR T=1 TO T1
3220 FOR N=1 TO N1
3230 X=INT(RND*4)+1
3240 Y=INT(RND*13)+1
3245 REM  dealt already?
3250 IF C$(X,Y)="" THEN 3230
3260 H$(T,N)=C$(X,Y)
3265 REM  card now dealt
3270 C$(X,Y)=""
3280 NEXT N
3290 NEXT T
3300 RETURN
3400 X=1
3410 Y=1
3415 REM  fill fourth hand
     with undealt cards
3420 FOR N=1 TO 13
3430 IF C$(X,Y)="" THEN 3470
3440 H$(4,N)=C$(X,Y)
3450 C$(X,Y)=""
3460 GOTO 3520
3470 Y=Y+1
3480 IF Y<14 THEN 3510
3490 Y=1
3500 X=X+1
3510 GOTO 3430
3520 NEXT N
3530 RETURN
3540 REM  sorting display
3550 FOR J=0 TO 1
3570 D$=H$(N,I+J)
3600 X=ASC(SEG$(D$,1,1))
3610 Y=ASC(SEG$(D$,2,1))
3620 CALL HCHAR(2+N*4,1+(I+J)*2,
X)
3630 CALL HCHAR(2+N*4,2+(I+J)*2,
Y)
3640 NEXT J
3650 RETURN
5000 PRINT " GRAPHICS BEING DEFI
NED"::" PLEASE WAIT.":::
5005 RESTORE 5000
5020 FOR N=1 TO 14
5030 READ X,G$
5040 CALL CHAR(X,G$)
5050 NEXT N
5070 DATA 113,006CFEFEFE7C3810,1
12,0010387CFE7C3810,114,4C525252
5252524C,115,7E08080808084830
5080 DATA 116,182442424241A,1
17,4448506050484442,118,18244242
7E424242
5090 DATA 120,10387CFEFE541038,1
04,383810D6FED61038,121,4C525252
5252524C,122,7E08080808084830
5100 DATA 123,182442424241A,1
24,4448506050484442,125,18244242
7E424242
5200 CALL COLOR(11,9,1)
5220 RETURN
```

LOGICOL

```
10 REM  LOGICOL
20 REM  MACBRIDE 1983
30 CALL SCREEN(16)
40 CALL CLEAR
50 PRINT TAB(10);"LOGICOL":::
60 PRINT " I WILL THINK OF 4 DIF
FERENT"::"COLOURS.":::
70 PRINT " YOU MUST TRY TO  GUES
S WHAT"::"THEY ARE.":::
80 PRINT " I'LL TELL YOU HOW MAN
Y ARE"::"IN THE RIGHT PLACE-(P)
AND"::"HOW MANY ARE RIGHT COLOUR
S":::
90 PRINT "BUT IN THE WRONG PLACE
 (C).":::
100 PRINT "  ONE MOMENT PLEASE."
::
110 FOR N=96 TO 136 STEP 8
120 CALL CHAR(N,"0")
130 NEXT N
140 CALL CHAR(144,"1824040810100
010")
150 FOR N=9 TO 15
160 READ B
170 CALL COLOR(N,2,B)
180 NEXT N
190 DATA 5,8,3,14,9,12,16
200 OPTION BASE 1
210 DIM P$(4)
220 DIM C$(4)
230 DIM L$(6)
240 FOR N=1 TO 6
250 READ L$(N)
260 NEXT N
270 DATA B,C,G,M,R,Y
280 PRINT " PRESS ANY KEY TO BEG
IN.":::
290 CALL SOUND(500,500,1)
300 CALL KEY(3,K,S)
310 IF S=0 THEN 300
320 CALL CLEAR
325 MYGO=0
330 PRINT TAB(2);CHR$(144);" ";C
HR$(144);" ";CHR$(144);" ";CHR$(
144);"  P  C     COLOUR CODE"
340 PRINT ::TAB(17);CHR$(96);" B
LUE....B"::
350 PRINT TAB(17);CHR$(104);" CY
AN....C"::
360 PRINT TAB(17);CHR$(112);" GR
EEN...G"::
370 PRINT TAB(17);CHR$(120);" MA
GENTA.M"::
380 PRINT TAB(17);CHR$(128);" RE
D.....R"::
390 PRINT TAB(17);CHR$(136);" YE
LLOW..Y":::::
400 PRINT TAB(2);"PICK 4 DIFFERE
NT COLOURS":::::
410 FOR N=1 TO 4
420 X=INT(RND*6)+1
430 IF N=1 THEN 470
440 FOR T=1 TO N-1
450 IF L$(X)=C$(T)THEN 420
460 NEXT T
470 C$(N)=L$(X)
480 NEXT N
490 PR=4
500 RP=0
510 RC=0
520 FOR N=1 TO 4
530 W$="COLOUR "&STR$(N)
540 R=22
550 C=3
560 GOSUB 6000
570 CALL SOUND(500,500,1)
580 CALL KEY(3,K,S)
590 IF S=0 THEN 580
600 CALL HCHAR(24,1,32,32)
610 FOR T=1 TO 6
620 IF K<>ASC(L$(T))THEN 640
630 GOTO 700
640 NEXT T
650 W$="PLEASE PRESS INITIAL LET
TER"
660 R=24
670 C=1
680 GOSUB 6000
690 GOTO 570
700 IF N=1 THEN 800
710 FOR T=1 TO N-1
720 IF CHR$(K)=P$(T)THEN 750
730 NEXT T
740 GOTO 800
750 W$="DIFFERENT COLOURS PLEASE
"
760 R=24
770 C=2
780 GOSUB 6000
790 GOTO 570
800 P$(N)=CHR$(K)
810 FOR T=1 TO 6
820 IF P$(N)<>L$(T)THEN 840
830 CN=T*8+88
840 NEXT T
850 CALL HCHAR(PR,N*2+2,CN)
860 NEXT N
870 IF MYGO THEN 2020
900 FOR T=1 TO 4
910 FOR N=1 TO 4
920 IF P$(T)=C$(T)THEN 950
930 IF P$(T)=C$(N)THEN 970
940 GOTO 990
950 RP=RP+1
960 GOTO 1000
970 RC=RC+1
980 GOTO 1000
990 NEXT N
1000 NEXT T
1010 W$=STR$(RP)
1020 R=PR
1030 C=12
1040 GOSUB 6000
1050 W$=STR$(RC)
1060 C=14
1070 GOSUB 6000
1080 PR=PR+2
1090 IF PR>18 THEN 1200
1100 IF RP=4 THEN 1400
1110 GOTO 500
1200 W$="ENOUGH !!IT WAS "
1210 FOR N=1 TO 4
1220 W$=W$&C$(N)
1230 W$=W$&" "
1240 NEXT N
1250 R=22
1260 C=3
1270 GOSUB 6000
1280 GOTO 1600
1400 W$="RIGHT IN "&STR$(PR-4)/
2)
1410 W$=W$&" GOES."
1420 R=22
1430 C=3
1440 CALL SOUND(1000,500,1)
1450 GOSUB 6000
1460 GOTO 1600
1600 W$="CAN I HAVE A GO ?(Y/N)
"
1610 R=24
1620 C=5
1630 GOSUB 6000
1640 CALL SOUND(1000,500,1)
1650 CALL KEY(3,K,S)
1660 IF S=0 THEN 1650
1670 IF K=89 THEN 1850
1680 IF K=78 THEN 1700
1690 GOTO 1640
1700 W$="DO YOU WANT ANOTHER GO?
(Y/N)"
1710 R=24
1720 C=2
1730 GOSUB 6000
1740 CALL SOUND(500,500,1)
1750 CALL KEY(3,K,S)
1760 IF S=0 THEN 1750
1770 IF K=89 THEN 320
1780 IF K=78 THEN 1800
1790 GOTO 1740
1800 STOP
1850 FOR N=2 TO 18 STEP 2
1860 CALL HCHAR(N,4,32,12)
1870 NEXT N
1880 CR=4
1885 CC=4
1890 CG=0
1900 CALL HCHAR(19,1,32,192)
1910 W$="PLEASE TELL ME YOUR COL
OURS"
1920 R=19
1930 C=2
1940 GOSUB 6000
1950 W$="I PROMISE NOT TO REMEMB
ER!"
1960 R=20
1970 GOSUB 6000
1980 CALL SOUND(500,500,1)
1990 MYGO=1
2000 PR=2
2010 GOTO 520
2020 CALL HCHAR(19,1,32,192)
2030 W$="I CANNOT GUESS PLACES."
2040 R=20
2050 C=3
2060 GOSUB 6000
2070 G$="BCGM"
2080 GOSUB 5000
2090 ON K GOTO 2100,2150,2500,30
00
2100 GOSUB 3500
2110 GOTO 2090
2150 G$="BCRY"
2160 GOSUB 5000
2170 ON K GOTO 2180,2200,2250,30
00
2180 GOSUB 3500
2190 GOTO 2170
2200 G$="GMRY"
2210 GOSUB 5000
2220 ON K GOTO 2230,2230,2230,30
00
2230 GOSUB 3500
2240 GOTO 2220
2250 G$="BGRY"
2260 GOSUB 5000
2270 ON K GOTO 2280,2300,2350,30
00
2280 GOSUB 3500
2290 GOTO 2270
2300 G$="CMRY"
2310 GOSUB 5000
2320 ON K GOTO 2330,2330,2330,30
00
2330 GOSUB 3500
2340 GOTO 2320
2350 G$="BMRY"
2360 GOSUB 5000
2370 ON K GOTO 2380,2400,2380,30
00
2380 GOSUB 3500
2390 GOTO 2370
2400 G$="CGRY"
2410 GOSUB 5000
2420 ON K GOTO 2430,2430,2430,30
00
2430 GOSUB 3500
2440 GOTO 2420
2500 G$="CGMR"
2510 GOSUB 5000
2520 ON K GOTO 2530,2550,2700,30
00
2530 GOSUB 3500
2540 GOTO 2520
2550 G$="BCGY"
2560 GOSUB 5000
2570 ON K GOTO 2580,2580,2600,30
00
2580 GOSUB 3500
2590 GOTO 2570
2600 G$="BCMY"
2610 GOSUB 5000
2620 ON K GOTO 2630,2630,2650,30
00
2630 GOSUB 3500
2640 GOTO 2620
2650 G$="BGMY"
2660 GOSUB 5000
2670 ON K GOTO 2680,2680,2680,30
00
2680 GOSUB 3500
2690 GOTO 2670
2700 G$="CGMY"
2710 GOSUB 5000
2720 ON K GOTO 2730,2750,2730,30
00
2730 GOSUB 3500
2740 GOTO 2720
2750 G$="BCGR"
2760 GOSUB 5000
2770 ON K GOTO 2780,2780,2800,30
00
2780 GOSUB 3500
2790 GOTO 2770
2800 G$="BCMR"
2810 GOSUB 5000
2820 ON K GOTO 2830,2830,2850,30
00
2830 GOSUB 3500
2840 GOTO 2820
2850 G$="BGMR"
2860 GOSUB 5000
2870 ON K GOTO 2880,2880,2880,30
00
2880 GOSUB 3500
2890 GOTO 2870
3000 W$="HOW'S THAT THEN?"
3010 R=22
3020 C=5
3030 GOSUB 6000
3040 GOTO 1700
3500 W$="PLEASE CHECK THAT!"
3510 R=23
3520 C=3
3530 GOSUB 6000
3540 CALL SOUND(500,500,1)
3550 CG=CG-1
3560 GOSUB 5100
3570 RETURN
5000 FOR N=1 TO 4
5010 FOR T=1 TO 6
5020 IF SEG$(G$,N,1)<>L$(T)THEN
5060
5030 CN=T*8+88
5040 CALL HCHAR(CR,CC,CN)
5050 CC=CC+2
5060 NEXT T
5070 NEXT N
5080 CR=CR+2
5090 CC=4
5100 W$="HOW MANY RIGHT COLOURS?
"
5110 R=24
5120 C=2
5130 GOSUB 6000
5140 CALL SOUND(500,500,1)
5150 CALL KEY(3,K,S)
5160 IF S=0 THEN 5150
5170 IF (K<49)+(K>52)THEN 5140
5180 CALL HCHAR(24,26,K)
5190 K=K-48
5200 CG=CG+1
5210 CALL HCHAR(23,1,32,64)
5220 RETURN
6000 FOR Q=1 TO LEN(W$)
6010 X=ASC(SEG$(W$,Q,1))
6020 CALL HCHAR(R,Q+C,X)
6030 NEXT Q
6040 RETURN
```

COMMANDO

```
10 REM  COMMANDO
20 REM  MACBRIDE 1983
21 REM
22 REM  ** please note **
23 REM     rems removed from
    cassette programs
24 REM  to save memory

30 CALL CLEAR
40 PRINT TAB(10);"COMMANDO":::
50 PRINT "GRAPHICS BEING DEFINED
.":::
60 PRINT "   PLEASE WAIT.":::
70 FOR N=1 TO 12
80 READ X,G$
90 CALL CHAR(X,G$)
100 NEXT N
105 REM   see 190 and
     290-310 for key

110 DATA 128,"0818080808081C",12
9,"1C2220408103E"
120 DATA 130,"38440418044438",13
1,"FF9999FFFFE7E7E7"
130 DATA 136,"00101898FE981810",
137,"0030389CFF9C3830"
140 DATA 138,"7878FCFFFC787800",
139,"FFBDDBE7E7DBBDFF",152,"0000
80FFF8C8C0C0"
150 DATA 144,"0100010001000188",
145,"FFFFFFFFFFFFFFFF",146,"0000
001818"
160 CALL COLOR(13,2,8)
170 CALL COLOR(14,9,12)
180 CALL COLOR(15,3,16)
185 CALL COLOR(16,2,16)
190 PRINT " IN THIS GAME YOU WIL
L GIVE":"THE ORDERS TO 3 UNITS
":CHR$(128);" ";CHR$(129);" ";CH
R$(130)::
200 PRINT "IN THEIR RAID ON AN E
NEMY":"INSTALLATION.":::
210 PRINT " THERE ARE THREE DIFF
ERENT":"TARGETS -1 THE FUEL DUM
P":"        -2 THE TANK DEPOT
::
220 PRINT "         -3 THE AIRPOR
T.":::
230 PRINT "1 HAS FEW GUARDS 3 HA
S MOST.":::
240 INPUT "WHICH TARGET ,1 2 OR
3 ?":TN
250 IF TN>3 THEN 240
260 PRINT ::" YOU CAN GIVE YOUR
UNITS ":"ORDERS FOR UP TO 6 MOV
ES":"AHEAD.":::
270 INPUT "HOW MANY MOVES AT A T
IME ?    (1 TO 6) ? ":M
280 IF M>6 THEN 270
290 PRINT ::" THESE SYMBOLS ARE
USED:":::" ";CHR$(136);"  ";CHR$(
137);"  PLANES"::
300 PRINT " ";CHR$(138);"   TANK
";CHR$(139);"  FUEL DUMP"::" ";C
HR$(131);"  BUILDING",
310 PRINT CHR$(152);"   GUARD"::
" YOU SCORE BY BLOWING UP ":"PL
ANES,TANKS AND FUEL DUMPS"::
320 PRINT " OR BY SHOOTING GUARD
S.":::
330 PRINT "   THE LOSS OF A UNIT
 WILL":::" REDUCE YOUR SCORE.":::
340 PRINT " ** ONE MOMENT PLEASE
 **"::
345 REM  set up arrays

350 OPTION BASE 1
355 RANDOMIZE
359 REM             Board
360 DIM B$(18,16)
364 REM             Bombs
365 DIM B(10,3)
369 REM             Guards
370 DIM G(10,3)
379 REM             Units
380 DIM U(3,2)
384 REM  shot guard count
385 GN=0
389 REM             Orders
390 DIM O(3,6)
394 REM  live unit count
395 UN=3
399 REM         basic board

400 FOR R=2 TO 17
410 FOR C=2 TO 15
420 B$(R,C)=CHR$(144)
430 NEXT C
440 NEXT R
450 ON TN GOTO 460,570,680
459 REM         fuel dump

460 FOR N=1 TO 8
470 C=INT(RND*13)+3
480 R=INT(RND*6)+6
490 B$(R,C)=CHR$(131)
500 NEXT N
510 FOR R=3 TO 5 STEP 2
520 B$(R,3)=CHR$(139)
530 B$(R,5)=CHR$(139)
540 NEXT R
550 GOTO 780
560 REM
569 REM         tank depot

570 FOR N=1 TO 6
580 R=INT(RND*5)+8
590 C=INT(RND*13)+3
600 B$(R,C)=CHR$(131)
610 NEXT N
620 FOR N=1 TO 10
630 R=INT(RND*7)+2
640 C=INT(RND*13)+3
650 B$(R,C)=CHR$(138)
660 NEXT N
670 GOTO 780
679 REM         airport

680 FOR N=1 TO 10
690 R=INT(RND*14)+1
700 C=16-R-(RND>.5)
710 B$(R,C)=CHR$(131)
720 X=136-(RND>.5)
730 R=INT(RND*14)+2
740 C=INT(RND*13)+3
750 IF R+C)15 THEN 730
760 B$(R,C)=CHR$(X)
770 NEXT N
779 REM         board edges

780 FOR R=1 TO 18
790 B$(R,2)=CHR$(145)
800 B$(R,16)=CHR$(145)
810 IF R)16 THEN 845
830 B$(1,R)=CHR$(145)
840 B$(18,R)=CHR$(145)
845 B$(R,1)=CHR$(32)
850 NEXT R
859 REM       position guards

860 FOR N=1 TO 1+TN*3
870 R=INT(RND*10)+2
880 C=INT(RND*13)+3
890 IF B$(R,C)<>CHR$(144)THEN 87
0
900 B$(R,C)=CHR$(152)
910 G(N,1)=R
920 G(N,2)=C
930 G(N,3)=INT(RND*4)+1
940 NEXT N
945 REM       position units

950 C=INT(RND*8)+4
955 FOR N=1 TO 3
960 B$(17,C+N)=CHR$(127+N)
970 U(N,1)=17
980 U(N,2)=C+N
985 NEXT N
990 INPUT "PRESS ENTER TO START
GAME ":A$
999 REM   initial screen

1000 CALL SCREEN(12)
1010 CALL CLEAR
1020 FOR R=1 TO 18
1030 FOR C=1 TO 16
1040 X=ASC(B$(R,C))
1050 CALL HCHAR(R,C,X)
1060 NEXT C
1070 NEXT R
1080 C=18
1090 FOR R=1 TO 18
1100 READ W$
1110 GOSUB 6000
1120 NEXT R
1130 DATA ORDER CODES," ",GO LEF
T    A,GO RIGHT   B,GO UP      C
,GO DOWN    D
1140 DATA " ",FIRE LEFT  E,FIRE
RIGHT F,FIRE UP    G,FIRE DOWN
H
1150 DATA " ",SET BOMB    I," ",N
O MOVE    J," ",CHANGE LAST,MOVE
    K
1160 W$="GAME LEVEL "&STR$(TN*6+
M)
1170 R=20
1180 C=3
1190 GOSUB 6000
1199 REM   main loop start

1200 CALL HCHAR(21,1,32,128)
1210 FOR T=1 TO 3
1220 IF U(T,1)=0 THEN 1430
1230 W$="ORDERS FOR UNIT "&STR$(
T)
1240 R=23
1250 C=3
1260 GOSUB 6000
1270 CALL HCHAR(21,(T-1)*10+2,12
7+T)
1280 FOR N=1 TO M
1290 W$="MOVE "&STR$(N)
1300 R=24
1310 C=5
1320 GOSUB 6000
1330 CALL SOUND(500,500,1)
1340 CALL KEY(3,K,S)
1350 IF S=0 THEN 1340
1360 IF (K<65)+(K)7)THEN 1330
1370 O(T,N)=K-64
1380 CALL HCHAR(21,(T-1)*10+3+N,
K)
1390 IF K<75 THEN 1420
1400 N=N-1-(N=1)
1410 GOTO 1290
1420 NEXT N
1430 NEXT T
1440 FOR T=1 TO 5
1450 FOR P=300 TO 600 STEP 60
1460 CALL SOUND(20,P,1)
1470 NEXT P
1480 NEXT T
1490 CALL HCHAR(23,1,32,64)
1499 REM   carry out orders

1500 FOR N=1 TO M
1510 FOR T=1 TO 3
1520 IF U(T,1)=0 THEN 1900
1530 IF O(T,N)>4 THEN 1560
1540 CALL HCHAR(U(T,1),U(T,2),14
4)
1550 B$(U(T,1),U(T,2))=CHR$(144)
1560 ON O(T,N)GOTO 1570,1590,161
0,1630,1670,1700,1730,1760,1790,
1850
1570 U(T,2)=U(T,2)-1-(B$(U(T,1),
U(T,2)-1)<>CHR$(144))
1580 GOTO 1640
1590 U(T,2)=U(T,2)+1+(B$(U(T,1),
U(T,2)+1)<>CHR$(144))
1600 GOTO 1640
1610 U(T,1)=U(T,1)-1-(B$(U(T,1)-
1,U(T,2))<>CHR$(144))
1620 GOTO 1640
1630 U(T,1)=U(T,1)+1+(B$(U(T,1)+
1,U(T,2))<>CHR$(144))
1640 B$(U(T,1),U(T,2))=CHR$(127+
T)
1650 CALL HCHAR(U(T,1),U(T,2),12
7+T)
1660 GOTO 1850
1670 D=4
1680 GOSUB 3000
1690 GOTO 1850
1700 D=2
1710 GOSUB 3000
1720 GOTO 1850
1730 D=1
1740 GOSUB 3000
1750 GOTO 1850
1760 D=3
1770 GOSUB 3000
1780 GOTO 1850
1790 IF BN=10 THEN 1810
1800 GOSUB 3500
1805 GOTO 1850
1810 W$=" YOU HAVE RUN OUT OF BO
MBS"
1820 R=23
1830 C=2
1840 GOSUB 6000
1850 IF BS=0 THEN 1900
1860 GOSUB 4000
1900 NEXT T
1909 REM       guards move

1910 FOR T=1 TO 10
1920 IF G(T,1)=0 THEN 2300
1929 REM    look for enemy

1930 FOR Z=1 TO 3
1935 IF U(Z,1)=0 THEN 2100
1940 IF RND).7 THEN 2100
1950 IF G(T,2)<>U(Z,2)THEN 2020
1970 IF ABS(G(T,1)-U(Z,1))>6 THE
N 2100
1980 D=1
1990 IF G(T,1)>U(Z,1)THEN 2080
2000 D=3
2010 GOTO 2080
2020 IF G(T,1)<>U(Z,1)THEN 2100
2040 IF ABS(G(T,2)-U(Z,2))>6 THE
N 2100
2050 D=4
2060 IF G(T,2)>U(Z,2)THEN 2080
2070 D=2
2080 GOSUB 3030
2090 Z=3
2100 NEXT Z
2109 REM        patrolling

2110 B$(G(T,1),G(T,2))=CHR$(144)
2120 CALL HCHAR(G(T,1),G(T,2),14
4)
2130 ON G(T,3)GOTO 2140,2170,220
0,2230
2140 IF B$(G(T,1)-1,G(T,2))<>CHR
$(144)THEN 2270
2150 G(T,1)=G(T,1)-1
2160 GOTO 2290
2170 IF B$(G(T,1),G(T,2)+1)<>CHR
$(144)THEN 2270
2180 G(T,2)=G(T,2)+1
2190 GOTO 2290
2200 IF B$(G(T,1)+1,G(T,2))<>CHR
$(144)THEN 2270
2210 G(T,1)=G(T,1)+1
2220 GOTO 2290
2230 IF B$(G(T,1),G(T,2)-1)<>CHR
$(144)THEN 2270
2240 G(T,2)=G(T,2)-1
2260 GOTO 2290
2270 G(T,3)=G(T,3)+1
2280 G(T,3)=G(T,3)+(4*(G(T,3)>4)
)
2290 B$(G(T,1),G(T,2))=CHR$(152)
2295 CALL HCHAR(G(T,1),G(T,2),15
2)
2300 NEXT T
2310 NEXT N
2320 IF UN THEN 2340
2330 GOTO 5000
```

```
2340 W$="PRESS 0 TO QUIT,6 TO GO
 ON"
2350 R=24
2360 C=1
2370 GOSUB 6000
2380 CALL KEY(3,K,S)
2390 IF K=81 THEN 5000
2400 IF K=71 THEN 1200
2410 GOTO 2380
2999 REM          shooting

3000 R=U(T,1)
3010 C=U(T,2)
3020 GOTO 3100
3030 R=6(T,1)
3040 C=6(T,2)
3100 FOR S=1 TO 6
3110 R=R+(D=1)-(D=3)
3120 C=C+(D=4)-(D=2)
3130 IF B$(R,C)=CHR$(144)THEN 33
00
3140 IF (B$(R,C)=CHR$(131))+(B$(
R,C)=CHR$(145))THEN 3350
3150 IF (B$(R,C)<CHR$(128))+(B$(
R,C)>CHR$(130))THEN 3280
3155 REM          unit hit

3160 FOR A=1 TO 3
3170 IF (U(A,1)=R)*(U(A,2)=C)THE
N 3190
3180 GOTO 3270
3190 U(A,1)=0
3200 FOR X=1 TO 5
3210 CALL HCHAR(R,C,127+A)
3220 CALL SOUND(50,200*X,1)
3230 CALL HCHAR(R,C,144)
3240 CALL SOUND(50,110*X,1)
3250 NEXT X
3255 B$(R,C)=CHR$(144)
3260 UN=UN-1
3265 IF UN=0 THEN 5000
3270 NEXT A
3280 IF B$(R,C)<>CHR$(152)THEN 3
350
3290 GOSUB 4500
3295 GOTO 3350
3299 REM          bullet
3300 CALL HCHAR(R,C,146)
3310 CALL SOUND(50,-1,1)
3320 CALL SOUND(20,-1,1)
3330 CALL HCHAR(R,C,144)
3340 GOTO 3400
3350 S=6
3400 NEXT S
3410 RETURN
3499 REM             bombs

3500 BN=BN+1
3510 B(BN,1)=U(T,1)
3520 B(BN,2)=U(T,2)
3530 W$="CHARGE SET.TIMER STARTE
D"
3540 R=23
3550 C=2
3560 GOSUB 6000
3570 CALL SOUND(500,760,1)
3580 B$=B$+1
3590 RETURN
4000 FOR Z=1 TO 10
4010 IF B(Z,1)=0 THEN 4410
4020 B(Z,3)=B(Z,3)+1
4030 IF B(Z,3)<18 THEN 4410
4040 REM          explosion

4050 FOR R=B(Z,1)-1 TO B(Z,1)+1

4060 FOR C=B(Z,2)-1 TO B(Z,2)+1
4070 IF (R(2)+(R)17)THEN 4360
4080 IF (C(2)+(C)15)THEN 4360
4090 CALL GCHAR(R,C,GC)
4100 CALL SOUND(50,-1,1)
4110 CALL HCHAR(R,C,145)
4120 CALL SOUND(50,-5,1)
4130 CALL HCHAR(R,C,32)
4140 CALL SOUND(50,-8,1)
4150 CALL HCHAR(R,C,144)
4160 IF GC<>152 THEN 4240
4165 REM   any guards there?
4170 FOR W=1 TO 10
4180 IF (G(W,1)=R)*(G(W,2)=C)THE
N 4200
4190 GOTO 4230
4200 G(W,1)=0
4210 GN=GN+1
4220 B$(R,C)=CHR$(144)
4230 NEXT W
4240 IF (GC>135)*(GC<140)THEN 42
60
4250 GOTO 4280
4260 B$(R,C)=CHR$(144)
4270 TS=TS+1
4280 IF (GC<128)+(GC>130)THEN 43
50
4285 REM  any units there?

4290 FOR W=1 TO 3
4300 IF (U(W,1)=R)*(U(W,2)=C)THE
N 4320
4310 GOTO 4340
4320 U(W,1)=0
4330 UN=UN-1
4335 IF UN=0 THEN 5000
4340 NEXT W
4350 B$(R,C)=CHR$(144)
4360 NEXT C
4370 NEXT R
4390 BS=BS-1
4400 B(Z,1)=0
4410 NEXT Z
4420 RETURN
4499 REM            guard shot

4500 FOR A=1 TO 10
4510 IF (G(A,1)=R)*(G(A,2)=C)THE
N 4530
4520 GOTO 4620
4530 G(A,1)=0
4540 FOR X=1 TO 6
4550 CALL HCHAR(R,C,152)
4560 CALL SOUND(50,-X,1)
4570 CALL HCHAR(R,C,144)
4580 CALL SOUND(50,110*X,1)
4590 NEXT X
4600 GN=GN+1
4610 B$(R,C)=CHR$(144)
4620 NEXT A
4630 RETURN
4999 REM          end of game

5000 IF UN THEN 5040
5010 CALL SOUND(1000,220,1)
5020 CALL SOUND(1000,110,1)
5030 GOTO 5050
5040 CALL SOUND(2000,294,1,370,1
,440,1)
5045 CALL HCHAR(22,1,32,96)
5050 W$="  FINAL SCORE "&STR$(TN
*M*(TS*10)*(GN*5))
5060 R=23
5070 C=2
5080 GOSUB 6000
5090 W$="ANOTHER GAME ?(Y/N) "
5100 R=24
5110 GOSUB 6000
5120 CALL SOUND(150,1397,1)
5130 CALL KEY(3,K,S)
5140 IF S=0 THEN 5130
5150 IF K=89 THEN 190
5160 IF K=78 THEN 5180
5170 GOTO 5130
5180 END

6000 FOR Q=1 TO LEN(W$)
6010 X=ASC(SEG$(W$,Q,1))
6020 CALL HCHAR(R,Q+C,X)
6030 NEXT Q
6040 RETURN
```


AIRSHIP

```
10 REM  AIRSHIP
20 REM  MACBRIDE 1983
30 CALL SCREEN(8)
40 CALL CLEAR
50 PRINT TAB(10);"AIRSHIP":::
60 PRINT "    YOU ARE THE CAPTAI
N OF":::"    AN AIRSHIP,FLYING FR
OM "::
70 PRINT "    PARIS TO LONDON.":
:::::
75 REM  graphics definition

80 GOSUB 2690
90 CALL KEY(3,K,S)
100 INPUT " DO YOU KNOW HOW TO F
LY AN  AIRSHIP? (Y/N)":A$
110 IF A$="Y" THEN 690
120 IF A$="N" THEN 140
130 GOTO 100
140 CALL SCREEN(8)
150 CALL CLEAR
155 REM  instructions

160 PRINT " KEEP A CLOSE WATCH O
N WIND":::" SPEED AND DIRECTION."
::
170 PRINT " IT WILL TEND TO BLOW
 YOU    OFF COURSE.":::
180 PRINT " E.G.- AIRSPEED 20 K.
P.H:::"    BEARING 90 DEGREES"::
190 PRINT " WIND - 10 K.P.H.":::"
   BEARING 180 DEGREES"::
200 CALL CHAR(136,"87DFFFFFDF8F0
203")
210 CALL CHAR(137,"E0F8FEFEF8E04
0C0")
220 PRINT "PRESS ANY KEY TO GO O
N.":::
230 CALL COLOR(13,14,1)
240 FOR C=3 TO 27 STEP 3
250 PRINT TAB(C);CHR$(128);
260 NEXT C
270 PRINT "     WIND (10 K.P.H)"
280 CALL COLOR(14,5,1)
290 PRINT ::CHR$(129);" ";CHR$(1
36);CHR$(137);" AIRSPEED 20 K.P.
H."
300 CALL KEY(3,K,S)
310 IF S=0 THEN 300
320 FOR C=1 TO 23 STEP 2
330 CALL SOUND(500,110,25)
340 PRINT TAB(C);CHR$(129);" ";C
HR$(136);CHR$(137)
350 CALL SOUND(1,-1,30)
360 IF C>21 THEN 380
370 CALL HCHAR(23,C+2,32,6)
380 NEXT C
390 PRINT " REAL BEARING 122 DEG
REES":::" GROUND SPEED 22.5 K.P.H
.":::
400 INPUT "DO YOU UNDERSTAND BEA
RINGS?":A$
410 IF A$="Y" THEN 600
420 IF A$="N" THEN 440
430 GOTO 400
435 REM  bearings display

440 PRINT TAB(13);"NORTH":TAB(14
);0::TAB(6);315;TAB(20);45:::::
450 PRINT " WEST";TAB(23);"EAST"
:TAB(4);270;TAB(22);90::::::TAB(
5);225;TAB(21);135:::
460 PRINT TAB(13);180:TAB(12);"S
OUTH":::" PRESS ANY KEY TO GO ON"
470 CALL VCHAR(5,17,130,7)
480 CALL VCHAR(13,17,128,7)
490 CALL HCHAR(12,11,131,6)
500 CALL HCHAR(12,18,129,6)
510 FOR N=1 TO 5
520 CALL HCHAR(12-N,17+N,132)
530 CALL HCHAR(12+N,17+N,133)
540 CALL HCHAR(12+N,17-N,134)
550 CALL HCHAR(12-N,17-N,135)
560 NEXT N
570 CALL SOUND(500,550,1)
580 CALL KEY(3,K,S)
590 IF S=0 THEN 580
595 REM  how to give your
     commands
600 PRINT TAB(2);"COMMANDING YOU
R AIRSHIP":::
610 PRINT "  WHEN THE 99 IS READ
Y..":::
620 PRINT " GIVE YOUR COMMANDS F
OR THE";" NEXT HOUR'S FLYING.":::
" MAX. SPEED 100 KPH.":::
630 PRINT " MAX. HEIGHT 1000 M."
:::" BEARINGS IN WHOLE NUMBERS":::
" USE >S< TO DELETE ERRORS.":::
640 PRINT " >ENTER< ALL COMMANDS
.":::" YOU WILL BE PULLED IN WHEN
  WITHIN 10 KM. OF AIRPORT.":::
650 PRINT ">> PRESS ANY KEY TO B
EGIN <<"
660 CALL SOUND(500,1397,1)
670 CALL KEY(3,K,S)
680 IF S=0 THEN 670
685 REM  initial values
     see below (810-)
690 RESTORE 700
700 READ GN,T,AS,AB,GS,GB,LD,LB,
PD,PB,H
710 DATA 1,-1,0,330,0,0,400,330,
0,0,0
720 RANDOMIZE
730 WD=INT(RND*5)*5
740 WB=INT(RND*72)*5
750 CALL CLEAR
760 PRINT "WIND:" bearing     s
peed     kph":::"AIRSHIP":" beari
ng      speed     kph"::
770 PRINT "ACTUAL TRAVEL":" bear
ing     speed     kph"::
780 PRINT "HEIGHT       metres"::
:TAB(11);"AIRPORTS"::"PARIS":"be
aring      distance      km"::
790 PRINT "LONDON":"bearing
distance      km":::"FLIGHT TIME
   hours":::
800 C=11
810 W$=STR$(WB)
815 REM  Wind Bearing

820 R=2
830 GOSUB 2810
840 W$=STR$(AB)
845 REM  Airship Bearing

850 R=5
860 GOSUB 2810
870 W$=STR$(GB)
875 REM  Ground Bearing -
the way you really go
880 R=8
890 GOSUB 2810
900 C=21
910 W$=STR$(WS)
915 REM  Wind Speed

920 R=2
930 GOSUB 2810
940 W$=STR$(AS)
945 REM  Air Speed
950 R=5
960 GOSUB 2810
970 W$=STR$(GS)
975 REM  Ground Speed

980 R=8
990 GOSUB 2810
1000 R=10
1010 C=9
1020 W$=STR$(H)
1025 REM  Height

1030 GOSUB 2810
1040 C=10
1050 W$=STR$(PB)
1055 REM  Paris Bearing

1060 R=16
1070 GOSUB 2810
```

```
1080 W$=STR$(LB)
1085 REM  London Bearing

1090 R=19
1100 GOSUB 2810
1110 T=T+1
1115 REM  Time-flying hours

1120 W$=STR$(T)
1130 R=21
1140 C=14
1150 GOSUB 2810
1160 C=23
1170 W$=STR$(PD)
1175 REM  Paris Distance

1180 R=16
1190 GOSUB 2810
1200 W$=STR$(LD)
1205 REM  London Distance

1210 R=19
1220 GOSUB 2810
1225 REM  Game Number
      1 to London
  2 to Paris
1230 IF (LD<11)*(GN=1)THEN 2050
1235 REM  close enough to
  catch the guy ropes?
1240 IF (PD<11)*(GN=-1)THEN 2050
1250 CALL SOUND(250,1397,1)
1255 REM  give orders

1260 W$="* READY FOR YOUR COMMAN
DS *"
1270 R=24
1280 C=1
1290 GOSUB 2810
1300 W$="BEARING ?"
1310 R=23
1320 C=3
1330 GOSUB 2810
1340 GOSUB 2460
1350 AB=I
1360 IF (AB>360)+(AB<0)THEN 1300
1370 W$="AIRSPEED ?"
1380 GOSUB 2810
1390 GOSUB 2460
1400 AS=I
1410 IF (AS>100)+(AS<0)THEN 1370
1420 W$="HEIGHT ?  "
1430 GOSUB 2810
1440 GOSUB 2460
1450 H=I
1460 IF H>1000 THEN 1430
1465 REM  crash??

1470 IF (H<20)*(AS>20)THEN 1890
1475 REM  too low??

1480 IF H<50 THEN 1500
1490 GOTO 1570
1500 W$="!! DANGER - TOO LOW !!"
1510 R=24
1520 GOSUB 2810
1530 FOR N=1 TO 6
1540 CALL SOUND(50,-1,1)
1550 NEXT N
1560 GOTO 1420
1570 W$=" WAIT - NAVIGATOR WORKI
NG "
1575 REM  calculation time

1580 R=24
1590 C=2
1600 GOSUB 2810
1605 REM  X=East-West shift
      Y=North-South
1610 AX=SIN(AB*.017)*AS
1620 AY=COS(AB*.017)*AS
1630 WX=SIN(WB*.017)*WS
1640 WY=COS(WB*.017)*WS
1645 REM  overall E/W,N/S
  movement - find speed       an
d bearing
1650 X=AX+WX
1660 Y=AY+WY
1670 GOSUB 2310
1680 GX=X
1690 GY=Y
1695 REM  Ground Bearing
      Ground Speed
1700 GB=B
1710 GS=D
1715 REM  find Distance and
 Bearing of London
1720 B=LB
1730 D=LD
1740 GOSUB 2270
1750 LD=D
1760 LB=B
1765 REM  now for Paris

1770 B=PB
1780 D=PD
1790 GOSUB 2270
1800 PB=B
1810 PD=D
1815 REM  the wind keeps
      changing
1820 WS=WS+(INT(RND*3)*5)-5
1830 IF WS>=0 THEN 1850
1840 WS=0
1850 WB=WB+(INT(RND*3)*10)-10
1860 IF WB>=0 THEN 1880
1870 WB=WB+360
1875 REM  back to display
1880 GOTO 800
1890 W$="!!TOO LOW - TOO FAST- C
RASH!!"
1900 FOR V=30 TO 1 STEP -1
1910 CALL SOUND(200,200+5*V,V,25
0+10*V,V,300+10*V,V,-8,V/2)
1920 NEXT V
1930 R=23
1940 C=1
1950 GOSUB 2820
1960 CALL SOUND(3000,110,1,115,1
,500,1,-8,1)
1970 W$="DO YOU WANT TO TRY AGAI
N?(Y/N)"
1980 R=24
1990 GOSUB 2810
2000 CALL SOUND(150,1397,1)
2010 CALL KEY(3,K,S)
2020 IF K=89 THEN 690
2030 IF K=78 THEN 2860
2040 GOTO 2010
2045 REM  in range of
      airport
2050 IF H<110 THEN 2110
2060 W$="OVER AIRPORT - BUT TOO
HIGH"
2070 R=24
2080 C=1
2090 GOSUB 2810
2100 GOTO 800
2110 W$="THAT'S CLOSE ENOUGH!"
2120 R=23
2130 C=2
2140 GOSUB 2810
2150 W$="DO YOU WANT TO FLY BACK
?(Y/N)"
2160 R=24
2170 GOSUB 2810
2180 CALL SOUND(150,1397,1)
2190 CALL KEY(3,K,S)
2200 IF K=89 THEN 2230
2210 IF K=78 THEN 2860
2220 GOTO 2190
2230 RESTORE 2240
2235 REM  data for London
      to Paris trip
2240 READ GN,T,AS,AB,GS,GB,LD,LB
,PD,PB,H
2250 DATA -1,-1,0,0,0,0,0,0,400,
150,0
2260 GOTO 750
2265 REM  finds how far E/W
      N/S of airports
2270 X=SIN(B*.017)*D
2280 Y=COS(B*.017)*D
2285 REM  adjusts for actual
      movement
2290 X=X-GX
2300 Y=Y-GY
2305 REM  common subroutine
      finds bearing and
 speed/distance from         E/W,
N/S figures
2310 D=INT(SQR(X*X+Y*Y))
2320 IF Y<>0 THEN 2350
2330 B=90
2340 GOTO 2440
2350 IF X<>0 THEN 2380
2360 B=0
2370 GOTO 2440
2380 B=INT(ATN(X/Y)*57.3)
2390 IF (Y>0)*(X>0)THEN 2440
2400 IF (Y>0)*(X<0)THEN 2430
2410 B=180+B
2420 GOTO 2440
2430 B=360+B
2440 REM  at last!! an end
 to those awful sums.
2450 RETURN
2455 REM  call key/input

2460 I$=""
2470 C1=14
2480 CALL SOUND(150,1397,1)
2490 CALL HCHAR(23,C1,144)
2500 CALL KEY(3,K,S)
2510 IF S=0 THEN 2490
2520 IF K=13 THEN 2610
2530 IF (K=83)*(I$<>"")THEN 2640
2540 IF (K<48)+(K>57)THEN 2500
2550 I$=I$&CHR$(K)
2560 CALL SOUND(100,-1,10)
2570 CALL HCHAR(23,C1,K)
2580 CALL SOUND(10,-1,10)
2590 C1=C1+1
2600 GOTO 2500
2610 I=VAL(I$)
2620 CALL HCHAR(23,14,32,LEN(I$)
+1)
2630 RETURN
2640 C1=C1-1
2650 L=LEN(I$)
2660 I$=SEG$(I$,1,L-1)
2670 CALL HCHAR(23,C1+1,32)
2680 GOTO 2490
2685 REM  arrow graphics
2690 RESTORE 2690
2700 FOR N=128 TO 135
2710 READ G$
2720 CALL CHAR(N,G$)
2730 NEXT N
2740 DATA 382828AAEE7C3810,00180
CFE87FE0C18,081C3E775514141C,183
07FE17F301800
2750 DATA 3F1F0F172B51A140,40A15
12B170F1F3F,02858AD4E8F0F8FC,FCF
8F0E8D48A8502
2760 CALL CHAR(144,"003C3C3C3C3C
3C")
2770 FOR S=9 TO 12
2780 CALL COLOR(S,2,16)
2790 NEXT S
2800 RETURN
2805 REM  print anywhere

2810 W$=W$&" "
2820 FOR Q=1 TO LEN(W$)
2830 CALL HCHAR(R,C+Q,ASC(SEG$(W
$,Q,1)))
2840 NEXT Q
2850 RETURN
2860 CALL SCREEN(16)
2870 CALL CLEAR
2880 PRINT TAB(8);"PROGRAM INDEX
":::
2890 PRINT "  INSTRUCTIONS.....
...155"
2900 PRINT "  BEARINGS DISPLAY..
...435"
2910 PRINT "  HOW TO COMMAND....
...595"
2920 PRINT "  INITIAL VALUES....
...685"
2930 PRINT "  PRINT VARIABLES...
...800"
2940 PRINT "  CHECK FOR END.....
..1225"
2950 PRINT "  COMMAND TIME......
..1260"
2960 PRINT "  CHECK-SAFE HEIGHT?
..1465"
2970 PRINT "  NAVIGATOR WORKING.
..1570"
2980 PRINT "  REAL FLIGHT-PATH..
..1605"
2990 PRINT "  AIRPORT FIGURES...
..1715"
3000 PRINT "  CHANGING WINDS....
..1815"
3010 PRINT "  CRASH!!...........
..1885"
3020 PRINT "  ARRIVAL...........
..2045"
3030 PRINT "  RESET FOR PARIS...
..2235"
3040 PRINT "  MORE CALCULATIONS.
..2265"
3050 PRINT "  INPUT SUB-ROUTINE.
..2455"
3060 PRINT "  GRAPHICS..........
..2685"
3070 PRINT "  PRINT ANYWHERE....
..2810"
3080 FOR D=1 TO 5000
3090 NEXT D
3100 END
```