Personal SOFTWARE

SPRING 1983 £1.95

MAKING THE MOST OF YOUR BBC MICRO

Get more out of your BBC Micro with our selected library of utilities, games and ideas.

BRITAIN'S BEST BUY FOR THE BBC MICRO USER

PLUS FULL USER REPORT INSIDE

FINANCIAL

BBC Model B — GAMES

THE WORLD TRAVEL GAME

1 or 2 Players,
Choice of Game,
'Exciting, Tense, Competitive and even Educational'.
XXXXX

Travel the World; Journey by air, rail and road.
Exchange currencies; Buy souvenirs; Book tickets.
Cope with hijacks, strikes, robbery and other problems inherent with travel.

Visit countries as diverse as Russia & the Falklands.

Your aim is to collect 6 souvenirs & return to LONDON intact!

— £6.95 —

GREAT BRITAIN LTD

You are P.M. and Chancellor of 'Great Britain'

You must select the Party you wish to represent and your aim is to stay in office for as long as possible. You must control inflation and unemployment, maintain th exchange rate, introduce social reforms and stay popular. The game is split into sectors: country profile, shopping basket, budget day, reform opportunities, manifesto, and most important election nights (a telling time).

A COMPLEX GAME THAT YOU WILL NOT TIRE OF IN A HURRY

— £5.95 —

INHERITANCE

Gt. Uncle Arbuthnot is dead.
You stand to inherit!!

A 2 part game. Prove your financial accumen in Part 1 by investing wisely at the stock and metal markets; if desperate try the casino or the horse races. If you are successful you will enter the world of big business in Part 2. Find the secret formula for paradise cola; manufacture and market the drink; cope with strikes, fires, frauds, cash shortages, etc. Your ultimate aim is to become a millionaire! **A MAMMOTH GAME PACKED FULL OF FEATURES**

— £5.95 —

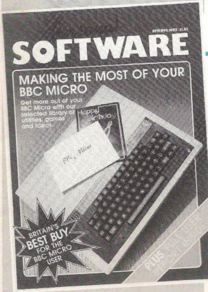
See Reviews in:
Acorn User Dec '82
Personal Computer Jan '83

Trade Enquiries Welcome
Special Deal for Schools.

Simon W. Hessel (Dept A)

15 Lytham Court, Cardwell Crescent
Sunninghill, Berks.–Tel: Ascot 25179

24hr Despatch — One Year Guarantee — Money-back if not satisfied.



CONTENTS

Editorial & Advertisement Office
145 Charing Cross Road, London WC2H OEE
Telephone: 01-437 1002 Telex: 8811896

Volume 1
No 4
Spring 1983

Editor: Henry Budgett
Assistant Editor: Wendy J Palmer
Advertisement Manager: Jeff Raggett
Advertisement Copy Control: Sonia Hunt
Managing Editor: Ron Harris BSc
Managing Director: T J Connell

Origination and design by MM Design & Print

Personal Software is a quarterly magazine appearing on the third Friday in May, August, November and February.

Distribution by: Argus Press Sales & Distribution Ltd, 12-18 Paul Street, London EC2A 4JS Tel: 01-247 8233. Printing by: Alabaster Passmore & Sons Ltd, Tovil, Maidstone, Kent.

Subscription Rates : UK £7.80 for four issues including postage. Airmail and other rates upon application to Personal Software Subscriptions, 513 London Road, Thornton Heath, Surrey CR4 6AR.

Credits: Our grateful thanks to the following individuals and organisations for their assistance in the production of this issue. Walt Disney Productions, 20th Century Fox, ITC, Charles, Kieran, Adrian, Junior, Tom, Enzo, Gary, Lesley, Chris, Walter, John and Bill.

PREFACE

Welcome to this, the fourth issue of Personal Software. Following the success of our first issue, dedicated to the BBC Micro, we have decided to repeat the formula again. However, unlike the previous offering, you will now find many completely new programs and features, not just re-written and converted software. What we have attempted to do is to take the best material available for the BBC Micro that we have published in ***Computing Today*** and to add new, specially commissioned features and programs to produce a complete package.

The material included in this issue ranges from the elementary level right up to sophisticated graphics and sound techniques and even extends to the disc user in offering an alternative to the *CAT command. So, no matter whether you have just bought your BBC Micro or are a well established user, you should find more than enough to try out. Much of the feature material is orientated towards software techniques and, rather than presenting complete listings, provides food for thought. Hopefully you, the user, will be able to take and adapt these ideas for use in your own programs. Once you have done that why not send them back to us so we can pass your knowledge on? Both ***Computing Today*** and ***Personal Software*** are always on the look out for good, well written material and we'll even pay you for anything that we use.

Quite apart from the useful software like the disassembler and the joystick calibrator we've included some more lighthearted material. As computer games go the 'lander' variants must be among the oldest but we make no apology at all for including our version. The graphics are superb and the apparent simplicity is soon found to be just that... apparent! If you've always thought that chess was rather dull why not try Ultima, our computerised alternative? The strategy is subtle and the displays really show the BBC Micro's graphics off at their best. For pure fun try rescuing the villagers from the wrath of the fire breathing dragon in St George and the Dragon. Finally, in the games section at least, may we draw your attention to Maze. Such an unassuming name for a game that presents a greater challenge to man than the Total Perspective Vortex! Once inside this three dimensional, three dimensional labyrinth you will need all your faculties just to remember which way is up, let alone find the object you seek at the heart of the complex.

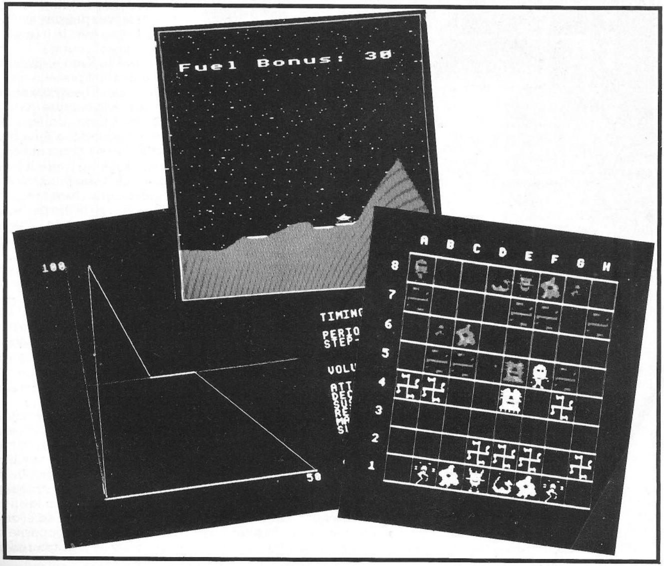
For the dabbler in BBC BASIC we have collected a whole host of hints, tips, short routines and ideas to enable you to write better, faster programs that make more use of the many and powerful facilities the BBC Micro has to offer.

For those who wish to delve yet further into the machine we have provided a complete User Report on the hardware, a list of all the various clubs and affiliated groups that are springing up around the country and a collection of books on and about the machine. So, there should be something for everybody!

Our first venture into the world of the BBC Micro was slightly fraught with problems, to put not too fine a point on it many of the listings, despite being machine generated, contained errors. In this issue we have run all the programs and listed them directly from the BBC Micro that they were running on. This should mean that they are free from errors but... If, and we sincerely hope there aren't any, errors do seem to be occurring in your program please check it ***very*** carefully before contacting us, a lot of the problems we found last time were caused by the omission of spaces next to variables which results in the now familiar No Such Variable message.

Future issues of ***Personal Software*** will tend to follow the format of this issue rather than concentrate on single subjects as we have for the past two issues. Among the machines we are currently planning special issues for are the Dragon 32, the various Commodore machines and the Apple. So, if you are a user of one of these and you have some suggestions to make why not drop us a line?

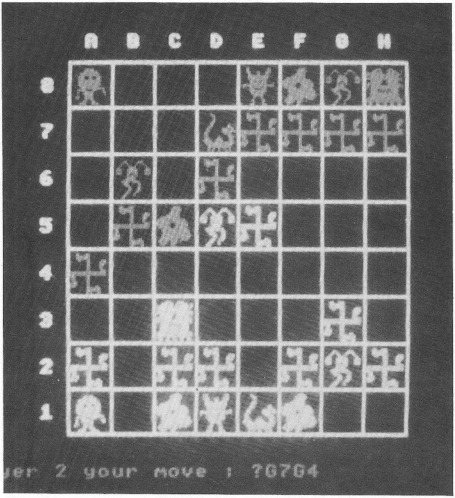
GAMES/UTILITIES



S Draper

ULTIMA

You may have played Ultima before but probably not on a micro.



Ultima is a board game played between two human players on an eight by eight board (much like a chess board). I was taught the game at college where we played it with a chess set (since there is, as far as I know, no such thing as an Ultima set). This was found to be very confusing so I decided to write a program which would use completely new pieces (to avoid confusion with the types of moves that chess pieces make), and which would check all moves for legality (no mean task as you shall see!).

The rules of the game are rather complicated and from now on I shall refer to the diagram of the starting position (Fig. 1). Starting from the square A8 and going along the 8th rank the pieces are as follows:

A8 — The Coordinator — by moving this piece to a new square any enemy piece caught on the intersection of the coordinator's new rank (horizontal line) and your King's file (vertical line) will be taken off. The coordinator moves like a Queen in chess.

B8 — The Leaper — this also moves like a Queen but can jump over enemies and in doing so takes them off. However it cannot jump over two adjacent pieces and must finish its move immediately after a jump unless there is another piece to jump immediately beyond this square. Note that all pieces jumped in any one move must lie in a straight line.

C8 — The Amoeba — this also moves like a Queen, but, as its name and shape imply it can change to meet circumstances — that is to say it takes pieces off in the way that piece would have to take another, ie it leaps Leapers, coordinates Coordinators, immobilizes Immobilizers (see later), etc.

D8 — The Withdrawer — yet another piece that moves like a Queen. This one's method of capture is entirely different however. Being timid creatures Withdrawers poison their enemies as they leave — a piece is taken off by a Withdrawer when it moves directly away from it after being next to it (enemy pieces only of course).

E8 — The King — this is the most important piece in the entire game. The whole object of the game is to capture your opponent's King. Unlike most of the other pieces the King does not move like a chess queen, but like a chess king (well, it seems logical don't you think). It also captures like a chess king, ie by moving on top of the piece to be taken off. Note that in Ultima there is no check.

H8 — The Immobilizer — this is another piece which moves like a queen. It cannot take pieces off but is none-the-less very important since it immobilizes all enemy pieces adjacent to it (including diagonally), and so holds them for capture by your other pieces. Note that an Amoeba can immobilize an Immobilizer.

The 7th file — all of these pieces are Rollers. They move like rooks in chess (ie as far as you like without jumping anything horizontally or vertically). They capture by moving vertically or horizontally adjacent to an enemy when there is one of your own pieces on the opposite side of the piece to be captured ie by sandwiching it.

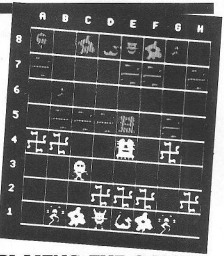
Note that in all cases only the piece that is actually moving can take an enemy off.

If you do not fully understand all the rules from the above (yes I agree — they are rather complicated) then I suggest you enter the program and play with someone — the computer won't allow you to make illegal moves and will perform all captures automatically, so you can soon learn what's going on.

PROGRAM STRUCTURE

Statement	Function	Action
Lines 10-50	Set up	Sets up arrays and data.
Lines 60-170	Title	Clears to Mode 7 and draws the title in double width coloured characters, then waits for a keypress.
Lines 180-220	Set up	Sets Mode 1, draws the initial position and gives player 1 the move.
Line 230-260	Next player	Changes whose move it is and prints the prompt in the relevant colour.
Lines 270-330	Get move	Inputs the player's move and checks that the chosen square is on the board.
Lines 340-430	Legality	Check legality of move and get another if illegal.
Lines 440-620	Leap	Check to see what piece has jumped, and goto PROCILLEGAL if the leap is not allowable, otherwise remove the leapt pieces.
Line 630	Branch	Goto a separate check depending on what piece has been moved.
Lines 640-690	Roller	Check for capture by a roller and remove any captured pieces.
Lines 700-720	Coordinate	Check for piece capture by Coordinator.
Lines 730-820	Amoeba	Check for piece capture by Amoeba.
Lines 830-850	King	Check for piece captured by King and update King's position.
Line 860	Update	Updates the board.
Line 870	Loopend	Returns to the beginning of the main loop unless someone's King has been taken.
Lines 880-900	Won	Congratulates the winner and starts another game after a short delay.
Lines 920-1100	Drawpiece	Draws a specified type and colour piece at a specified position (see text).
Lines 1110-1310	Data	Data for the graphics.
Lines 1320-1520	Board	Sets up the board in its initial position and displays it.
Lines 1530-1580	Out	Removes a piece from the display at a specified position.
Lines 1590-1620	Lines	Draws the border around a specified square.
Lines 1630-1710	Illegal	Prints a message to tell the player that he is trying to make an illegal move.
Lines 1720-1740	Cursor	Removes the flashing cursor.

Table 1. Program structure.



PLAYING THE GAME

When the program is run it will draw the board with the initial position set up on it. The colours are red and yellow with a white board since these give reasonable contrast on a black and white set. Initially there will be a yellow prompt saying 'Player 1 your move:?' at the bottom of the screen. This is to tell the player with the yellow pieces that the computer is ready for his move. When the first player has moved, by typing the numbers of the square from which he wishes to move a piece followed by the square to which he wishes to move it (eg G2G4), the yellow prompt will be replaced by a red prompt for the second player, and so on until the game is over.

TECHNICAL DETAILS

There is really very little worth saying here — the program is long but contains nothing particularly unusual in it, save perhaps the way in which the pieces are drawn. With the 0.1 operating system it is not possible to define more than 32 user defined graphics characters and this is clearly not enough since each piece uses nine. As a result all of the graphics data is kept in DATA statements until it is needed. It is then read into just nine user defined graphics (224-232), and these nine are used for all the pieces, being redefined every time a piece is drawn. This method is slow but it has the advantage of being easy to follow unlike the faster POKEing or machine code methods.

```
>LIST
    5 DIM BOARD(9,9),TYPE(7),LEAPT(4),KINFILE(1)
   10 DATA2,1,4,5,6,4,1,3
   20 FORLOOP=0TO7
   30   READTYPE(LOOP)
   40   NEXTLOOP
   50 MODE7
   51 PRINT'''
   52 PROCREMOVECURSOR
   55 FORTITLE=1TO2
   60   PRINTCHR$(141);SPC(13);
   70   VDU131,157,132
   80   PRINT"ULTIMA  ";CHR$(156)
   90   NEXTTITLE
   92 PRINT'''"    This is the board game ULTIMA for"
   93 PRINT"two players."''
   94 VDU136,134:PRINT"Press space to begin.";CHR$(137)
   95 DUMMY=GET
  110 MODE1
  115 PROCREMOVECURSOR
  120 PROCDRAWBOARD
  130 WIN=FALSE
  140 TURN=TRUE
  150 REPEAT
  155   TURN=-TURN
  157   COLOUR(TURN+3)/2
  160   PRINTTAB(0,29);SPC(39);TAB(0,29);"Player ";(3-TUR
N)/2;" your move : ?";
  170   VDU8
  180   INPUTTURN$
  190   X1=ASC(LEFT$(TURN$,1))-64
  200   X2=ASC(MID$(TURN$,3,1))-64
  210   Y1=ASC(MID$(TURN$,2,1))-48
  220   Y2=ASC(MID$(TURN$,4,1))-48
  230   IFX1>8ORX2>8ORY1>8ORY2>8ORX1<1ORX2<1ORY1<1ORY2<1
THEN PROCILLEGAL:GOTO160
  235   BIT=BOARD(X1,Y1)
  240   IFBIT=0 OR SGN(BIT)<>SGN(TURN) THEN PROCILLEGAL:G
OTO160
  250   IFX1-X2<>0 AND Y1-Y2<>0 AND ABS(X1-X2)<>ABS(Y1-Y2
) THEN PROCILLEGAL:GOTO160
  260   IFABS(BIT)=6AND(ABS(X1-X2)>1 OR ABS(Y1-Y2)>1) THE
N PROCILLEGAL:GOTO160
  270   IFX1=X2 AND Y1=Y2 THEN PROCILLEGAL:GOTO160
  280   IF BOARD(X2,Y2)<>0 AND ABS(BIT)<>6AND(ABS(BOARD(X
2,Y2))<>6 OR ABS(BIT)<>5)THEN PROCILLEGAL:GOTO160
  281   FORDELTAX=-1TO1
  282     FORDELTAY=-1TO1
  283       IFBOARD(X1+DELTAX,Y1+DELTAY)=-SGN(BIT)*4 OR (
ABS(BIT)=4 AND BOARD(X1+DELTAX,Y1+DELTAY)=-SGN(BIT)*5)THE
NDELTAY=1:DELTAX=1:NEXTDELTAY,DELTAX:PROCILLEGAL:GOTO160
  284     NEXTDELTAY,DELTAX
  285   LEAP=TRUE:LEAPCOUNTER=0
  286   X=X1-SGN(X2-X1)
  287   Y=Y1-SGN(Y2-Y1)
  288   AFTER=0
  290   REPEAT
  291     IF AFTER THENAFTER=AFTER+1
  292     IFAFTER>3THENUNTILTRUE:PROCILLEGAL:GOTO160
  300   Y=Y+SGN(Y2-Y1)
  305   X=X+SGN(X2-X1)
  310   IFX=X1 AND Y=Y1 THEN380
  319   IF BOARD(X,Y)=0AND AFTER=3THEN PROCILLEGAL:GOTO16
0 ELSE IF BOARD(X,Y)=0 THENLEAP=TRUE:GOTO380
  330   IFABS(BIT)<>2 AND ABS(BIT)<>6 AND(ABS(BIT)<>5 OR
BOARD(X,Y)<>-SGN(BIT)*2)THENUNTILTRUE:PROCILLEGAL:GOTO160
  340 IFLEAP=FALSE THENUNTILTRUE:PROCILLEGAL:GOTO160
  350 LEAP=NOT LEAP
  355 AFTER=1
  360 LEAPT(LEAPCOUNTER)=X+8*Y
  370 LEAPCOUNTER=LEAPCOUNTER+1
  380 UNTILX=X2 AND Y=Y2
  390 IFLEAPCOUNTER<>0THEN FORLOOP=0 TO LEAPCOUNTER-1:PRO
COUT(LEAPT(LOOP) MOD8,LEAPT(LOOP) DIV8):BOARD(LEAPT(LOOP)
 MOD8,LEAPT(LOOP) DIV8)=0:NEXTLOOP
  400 ON ABS(BIT) GOTO410,470,450,470,500,700,600
  410 FORXSTEP=-1TO1 STEP2
  420   IFSGN(BOARD(X2+XSTEP,Y2))=-TURN THEN IF SGN(BOARD
(X2+2*XSTEP,Y2))=TURN THEN PROCOUT(X2+XSTEP,Y2):BOARD(X2+
XSTEP,Y2)=0
  430   NEXTXSTEP
  435 FORYSTEP=-1TO1 STEP2
  437   IFSGN(BOARD(X2,Y2+YSTEP))=-TURN THEN IF SGN(BOARD
(X2,Y2+2*YSTEP))=TURN THEN PROCOUT(X2,Y2+YSTEP):BOARD(X2,
Y2+YSTEP)=0
  438   NEXTYSTEP:GOTO800
  450 TARGET=BOARD(KINFILE((TURN+1)/2),Y2)
  460 IFSGN(TARGET)=-TURN THEN PROCOUT(KINFILE((TURN+1)/2
),Y2):BOARD(KINFILE((TURN+1)/2),Y2)=0
  470 GOTO800
  500 TARGETX=X1+SGN(X1-X2):TARGETY=Y1+SGN(Y1-Y2)
  510 IFBOARD(TARGETX,TARGETY)=-TURN*7 THENPROCOUT(TARGET
X,TARGETY):BOARD(TARGETX,TARGETY)=0
  520 FORXSTEP=-1TO1 STEP2
  530   IFBOARD(X2+XSTEP,Y2)=-TURN THEN IF SGN(BOARD(X2+2
*XSTEP,Y2))=TURN THEN PROCOUT(X2+XSTEP,Y2):BOARD(X2+XSTEP
,Y2)=0
  531   NEXTXSTEP
  533 FORYSTEP=-1TO1STEP2
  540   IFBOARD(X2,Y2+YSTEP)=-TURN THEN IF SGN(BOARD(X2,Y
2+2*YSTEP))=TURN THEN PROCOUT(X2,Y2+YSTEP):BOARD(X2,Y2+YS
TEP)=0
  550   NEXTYSTEP
  560 IFBOARD(KINFILE((TURN+1)/2),Y2)=-TURN*3 THEN PROCOU
T(KINFILE((TURN+1)/2),Y2):BOARD(KINFILE((TURN+1)/2),Y2)=0
  570 GOTO800
  600 TARGETX=X1+SGN(X1-X2):TARGETY=Y1+SGN(Y1-Y2)
  610 IFSGN(BOARD(TARGETX,TARGETY))=-TURN THENPROCOUT(TAR
GETX,TARGETY):BOARD(TARGETX,TARGETY)=0
  700 KINFILE((TURN+1)/2)=X2
  800 BOARD(X1,Y1)=0:PROCOUT(X1,Y1):PROCDRAWPIECE(X2-1,Y2
,ABS(BIT)-1,(TURN+1)/2):BOARD(X2,Y2)=BIT
  810 UNTILWIN
  820 PRINTTAB(0,29);SPC(39);TAB(0,29);"Congratulations p
layer ";(3-TURN)/2;", a brilliant"'"win!!"
  830 PROCWAIT(200)
  840 RUN
 5000 END
 8999 DEFPROCDRAWPIECE(X,Y,TYPE,PLAYER)
 9002 Y=8-Y
 9005 COLOUR PLAYER+1
 9010 RESTORE9500
 9011 IFTYPE=0THEN9020
 9012 FORDUMMYREAD=1TO72*TYPE
 9013   READ A
 9014   NEXTDUMMYREAD
 9020 FORCHARACTER=224 TO 232
 9025   VDU23,CHARACTER
 9030   FORROW=0 TO 7
 9040     READ INFORMATION
 9050     VDU INFORMATION
 9060     NEXTROW,CHARACTER
 9070   PRINTTAB(3*X+8,3*Y+3);:VDU224,227,230
 9080   PRINTTAB(3*X+8,3*Y+4);:VDU225,228,231
 9090   PRINTTAB(3*X+8,3*Y+5);:VDU226,229,232
 9096   PROCLINES(X,Y)
 9100   ENDPROC
 9500    DATA 0,0,6,7,7,3,0,0,0,0,0,63,63,56,24,24,28,60,
56,0,0,0,0,0
 9510    DATA 0,0,56,248,248,24,24,24,24,24,24,255,255,24
,24,24,24,24,24,31,31,28,0,0
 9520    DATA 0,0,0,0,0,28,60,56,48,48,28,252,252,0,0,0,0
,0,192,224,224,96,0,0
10000   DATA 0,0,0,1,2,4,8,28,20,28,0,0,0,3,1,0,0,0,0,0,1
,3,0,0
10010    DATA 0,0,0,129,66,36,60,126,126,&66,&5A,&7C,&FC,&
EF,&C1,&E0,&31,&31,&63,&C4,&84,0,0,0
10020    DATA 0,0,0,&C0,&20,&10,8,&1C,20,&1C,0,0,0,0,128,1
28,128,128,0,0,0,0,0,0
10030     DATA 0,0,0,0,0,3,3,7,7,7,7,7,3,1,7,4,4,4,12,0,0,
3,0,0
10040     DATA 0,0,0,124,255,255,255,&99,&99,255,255,255,2
55,255,&C7,252,252,124,124,&6C,&C6,&C7,0,0
10050    DATA 0,0,0,0,0,128,128,192,192,192,192,192,128,24
0,16,24,8,0,0,0,0,128,0,0
10060    DATA 0,0,1,7,47,63,15,63,15,15,63,47,47,15,15,15,
15,31,63,63,25,25,0,0
10070    DATA 0,0,&C3,&66,&E7,255,255,255,&E7,&E7,&C3,255,
255,&55,&AA,255,255,255,255,255,153,153,0,0
10080    DATA 0,0,128,&E4,244,248,240,252,244,240,248,252,
244,240,240,240,240,248,252,252,152,152,0,0
10090    DATA 0,0,0,0,0,0,1,15,31,63,63,63,31,7,7,3,7,7,15
,31,15,4,0,0
10100    DATA 0,0,30,63,62,126,254,254,255,255,&CF,&87,&CF
,255,255,255,255,255,252,189,56,48,0,0
10110    DATA 0,0,0,0,0,0,0,192,224,240,240,224,240,248,
248,252,248,240,0,0,0,0,0
10120    DATA 0,0,6,6,7,3,3,3,1,1,1,31,17,1,1,0,0,0,0,0,0,
1,0,0
10130    DATA 0,0,0,0,24,24,255,255,255,&DB,255,255,189,&C
3,255,255,126,&66,66,&C3,66,&C3,0,0
10140    DATA 0,0,96,96,224,192,192,192,128,128,136,248,12
8,128,128,0,0,0,0,0,0,128,0,0
10150    DATA 0,0,0,0,0,0,0,0,1,3,3,6,6,6,7,15,7,3,1,1,1
,0,0
10160    DATA 0,0,128,96,32,32,96,192,192,128,1,1,0,0,4,14
,&9F,255,255,255,36,68,0,0
10170    DATA 0,0,0,0,0,0,0,0,48,56,&EC,254,240,56,28,24,2
8,252,248,240,&90,&50,0,0
11000    DEFPROCDRAWBOARD
11010    FORROW=0TO8
11020      MOVE256,928-96*ROW:DRAW1024,928-96*ROW
11030      NEXTROW
11040    FORCOLUMN=0TO8
11050      MOVE256+96*COLUMN,928:DRAW256+96*COLUMN,160
11060      NEXTCOLUMN
```

```
11061    FORCOLUMNNUMBER=1TO8
11062      PRINTTAB(3*COLUMNNUMBER+6,1);CHR$(COLUMNNUMBER+
64)
11063      PRINTTAB(6,COLUMNNUMBER*3+1);CHR$(57-COLUMNNUMB
ER)
11064      NEXTCOLUMNNUMBER
11070    FORPIECE=0TO7
11080      TYPE=TYPE(PIECE)
11090      PROCDRAWPIECE(PIECE,1,TYPE,1):BOARD(PIECE+1,1)=
TYPE+1
11091      IFPIECE=3ORPIECE=4 THEN TYPE=11-TYPE
11095      PROCDRAWPIECE(PIECE,8,TYPE,0):BOARD(PIECE+1,8)=
-TYPE-1
11096      PROCDRAWPIECE(PIECE,2,0,1):BOARD(PIECE+1,2)=1
11097      PROCDRAWPIECE(PIECE,7,0,0):BOARD(PIECE+1,7)=-1
11100      NEXTPIECE
11105    KINFILE(0)=5:KINFILE(1)=4
11110    ENDPROC
15000    DEFPROCOUT(X,Y)
15001    IFABS(BOARD(X,Y))=6THENWIN=TURN
15005    Y=8-Y:X=X-1
15010    PRINTTAB(3*X+8,3*Y+3);"   ";TAB(3*X+8,3*Y+4);"
";TAB(3*X+8,3*Y+5);"   "
15015    PROCLINES(X,Y)
15020    ENDPROC
15999    DEFPROCLINES(X,Y)
16020    MOVE96*X+256,928-96*Y
16030    DRAW96*X+256,832-96*Y:DRAW96*X+352,832-96*Y:DRAW9
6*X+352,928-96*Y:DRAW96*X+256,928-96*Y
16040    ENDPROC
20000    DEFPROCILLEGAL
20010    PRINTTAB(0,29);"You can't do that!!";SPC(10)
20015    PROCWAIT(200)
20017    PRINTTAB(0,29);SPC(39)
20020    ENDPROC
30000    DEFPROCWAIT(T)
30010    TIME=0
30020    REPEATUNTILTIME=T
30030    ENDPROC
32000    DEFPROCREMOVECURSOR
32010    !&FE00=&10200A
32020    ENDPROC
```

Listing 1. The program for playing Ultima.

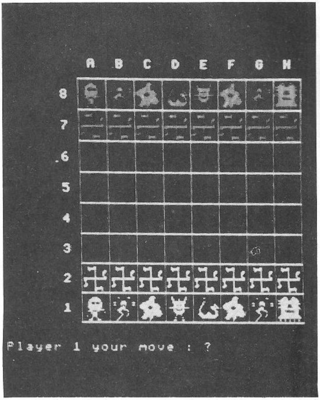


Fig. 1. The starting position for Ultima.

GEMINI PERFORMANCE

It can do a powerful job for yo

SPECIAL LIMITED OFFER

Buy just any two programs at £19.95 and take one at £19.95 FREE!

CASH BOOKS ACCOUNTS PROGRAM FOR BBC MICRO . . .£95.00

New

One of the most innovative business programs on the market. Most serious accountancy packages are written and coded by professional and competent programmers. The Gemini Cashbook Accounting program was written by practising Chartered Accountants and coded by professional and competent programmers. This is a fundamental difference.

This practical program is simple to use and will replace your manual cash and bank records and by giving you instant management information, it may even put your accountant out of job!

With exceptionally exhaustive user documentation, full technical back up and product update policy this program will increase the efficiency and profitability of your business. Take a look at the information this program will provide.

* summary of VAT information to enable you to complete your VAT returns
* cumulative receipts and payments report analysed over the standard profit and loss and balance sheet heading.
* option for departmental analysis of sales and purchases
* print out of all transactions
* journal routine for entering transfers between accounts and year end adjustment for debtors, creditors etc.
* year end trial balance
* profit and loss account and balance sheet.

These statements can be produced at what ever interval you require e.g. monthly, quarterly or annually.

Coming soon:– Integrated Sales + Purchase Ledgers

> ". . . the systems worked immaculately when tested . . ."
> "Mailist is a very professional piece of software . . ."
> (Which Micro & Software Review Feb 83)

Here's a range of software for the independent businessman that's designed to harness the power of your micro to deliver the vital information you need in all key areas of your business. A breakthrough on both price and performance, each program is fully tested and comes with all the documentation back up you need.

> "Gemini's range of software is in the vanguard of the releases for 'serious' micro users . . ."
> (Which Micro and Software Review)

SPREADSHEET ANALYSIS
BEEBCALC £19.95
DRAGONCALC £19.95

New

FOR BBC AND DRAGON 32. Spreadsheet processors have proved to be important tools for using micros in business, scientific and domestic financial applications.

Without any programming knowledge at all, you may:–

* Set up a computerised spreadsheet, with chosen row and column names.
* Specify formulae relating any row or column to any other.
* Enter your source data and have the results calculated.
* Save the results on tape (or disk – BBC) for later reloading and manipulation.
* Print the tabulated results in an elegant report format.
* Experienced users may access saved files and write their own reporting or graphics presentation programs for the results.

Some typical applications:–

* Small business accounting applications, e.g. profit and loss statements and cashflow projections, break-even analyses etc.
* Investment project appraisal – anything from double glazing to oil rigs!
* Comparing rent/lease/buy options
* Processing the results of scientific experiments or field studies
* Engineering calculation models
* In fact, anything that involves repeated re-calculation of results presented in tabular or spreadsheet format.

Program Availability Chart:–

	Database	Stock Control	Mailist	Invoices & Statements	Spread sheet Analysis	Cashbook Accounting	Word processor	Home Accounts	Commercial Accounts
Sinclair Spectrum 16k or 48k	X	X	X					X	X
Dragon 32k or 64k	X				X			X	
VIC20 (16k+)	X	X	X	X				X	X
Sinclair ZX81 (16k+)	X								
Grundy Newbrain	X								
Texas T199 4A	X								
Osborne 1	X								
Sharp MZ80A	X	X	X	X				X	X
Sharp MZ80K	X	X	X	X				X	X
Sharp MZ80B	X	X	X	X				X	X
BBC micro model A or B 32K	X	X	X	X	X	X	X	X	X

I'S NEW
CE SOFTWARE
ur business at petty cash prices.

INVOICES AND STATEMENTS . . . £19.95
Compatible with most micros. See table. Ideal for the small business. A complete suite of programs together with generated customer file for producing crisp and efficient business invoices and monthly statements on your line printer. All calculations include VAT automatically, and the program allows your own messages on the form produced. This program gives you superb presentation and saves time on one of the most tedious tasks in the office.

COMMERCIAL ACCOUNTS . . . £19.95
Compatible with most micros. See table. A gem of a program, all for cassette, with the following features:– Daily Journal. Credit Sales. Cash Sales. Credit Purchases. Purchases – other. Sales Ledger. Purchase Ledger. Bank Account. Year to date summary. A fully interactive program suitable for all businesses. Files can be saved and loaded and totals from one file carried forward to another on cassette. Particularly useful from a cash flow point of view, with an immediate accessibility to totals for debtors and creditors. Bank totally supported with entries for cheque numbers, credits and, of course, running balance.

MAILING LIST . . . £19.95
Compatible with most micros. See table. A superb dedicated database to allow for manipulations of names and addresses and other data. Gemini's unique 'searchkey' system gives you a further ten 'user-defined parameters' to make your own selections. Features include the facility to find a name or detail when only part of the detail is known, it will print labels in a variety of user specified formats.

DATABASE . . . £19.95
Compatible with most micros. See table. The program that everyone needs, the most valuable and versatile in your collection. Facilities include sort search, list print if required. Can be used in place of any card index application; once purchased you can write your own dedicated database to suit your particular needs with a limitless number of entries on separate cassettes.

STOCK CONTROL . . . £19.95
Compatible with most micros. See table. Dedicated software with all that's necessary to keep control of stock. This program will take the tedium out of stock control and save time and money. Routines include stock set up, user reference number, minimum stock level, financial summary, line print records, quick stock summary, add stock, delete/change record and more.

HOME ACCOUNTS . . . £19.95
Compatible with most micros. See table. Runs a complete home finance package for you with every facility necessary for keeping a track of regular and other expenses, bank account mortgage, H.P. etc. This program also allows you to plot graphically by Listograms your monthly outgoings.

WORD PROCESSOR . . . £19.95
Compatible with most micros. See table. This program features routines found in much larger and more expensive packages with a typical word length of 5-6 letters it allows for around 1000 words in memory at one time. Ideal for the user who requires a simple program to write letters on his computer. Features include, block delete, block insert, search and replace, edit text, display text and more.

Dealer/Trade enquiries invited – generous trade discounts for quantity

Special ACCESS card instant sales hotline for GUARANTEED despatch within 24 hours . . . Tel: 03952-5165

24 hr Ansaphone Service.

All enquiries other than credit card sales to 03952-5832

Gemini. Functional Software Specialists. 9, Salterton Road, Exmouth, Devon.

Tick the box for Program you require. Prices include V.A.T. and Package and Postage.
Please supply the following cassette software.

Database	£19.95 ☐	ZX81 16K Database	£9.95 ☐
Stock Control	£19.95 ☐	BBC Cash Book disk or tape	£95.00 ☐
Mailing List	£19.95 ☐	BBC Disks – other titles	£23.95 ☐
Invoices and Statements	£19.95 ☐	Osborne Disk Database	£23.95 ☐
Commercial Accounts	£19.95 ☐	Word processor	£19.95 ☐
Home Accounts	£19.95 ☐	Beebcalc	£19.95 ☐
		Dragoncalc	£19.95 ☐

Name____________________

Address____________________

Machine Type____________________ Memory Size____________________

I enclose____________________

Make cheques and postal orders payable to Gemini Marketing Ltd.

Diners Card Number____________________ Access Number____________________

Signature____________________

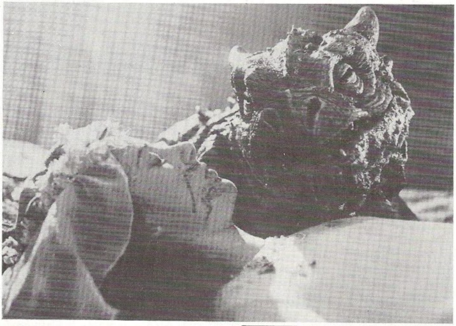
DINERS CLUB INTERNATIONAL   VISA

Gemini. Functional Software Specialists, 9 Salterton Road, Exmouth, Devon.

H Roberts

GEORGE AND THE DRAGON

Fight the dragons and save the people of Sleepy Vale as they hide in the castle, by playing George and the Dragon.



Once upon a time there was a little valley called Sleepy Vale. All the people who lived there were happy, and they carried on their daily life in peace and harmony.

Then one day a large pride (herd?) of dragons came to the valley and drove all the people out, killing and burning as they went. Some of the townsfolk managed to stay in the valley, living in the large castle owned by the lord, but they lived in terror of the dragons that now roamed their desecrated homeland.

You take the role of St George, called by the people of the valley to liberate their home from the dragon menace. You have no sword, but a trusty bow and arrow to shoot the dragons, but remember to keep away from their vicious teeth and roaring flame, as these are deadly. Once you have slain a dragon, head for the castle as the people raise the drawbridge to give you food and rest before you set out once more on your difficult task.

Use keys 'Z' and 'X' to rotate left and right respectively, 'Return' to move in the direction you are facing, and space bar to shoot an arrow. You cannot fire again until the first arrow has found a mark. Be careful at the bridge over the river as the dragons often guard it ferociously, also beware of the magical forests in which you move at quarter speed, and can be driven back if you stop.

Good luck on your difficult task, no-one can help you now.

VARIABLES

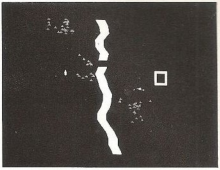
X%(D),Y%(D)	Direction arrays
FL	Counter for duration of dragon's flame
LOOP	General loop variable
TITLE	Loop variable for double height printing on title
SCORE	Score
SP	Speed of dragon's movement
A%,B%	Position of arrow
V%,W%	Direction vector of arrow
X%,Y%	Position of St. George
D%	Direction St. George is facing
H%	Temporary store for St. George's direction
F%,G%	Position of Dragon
K%	Direction Dragon is facing
I%,J%	Temporary storage for St. George's and the Dragon's positions
DR	= TRUE If Dragon is alive = FALSE If Dragon is dead
FLAG	Space bar flag, to make sure only one arrow is fired for each press of the space bar
FINISHED	A flag set by various procedures, to tell the main routine when the frame is finished
E%	ASCII code for current Dragon graphic
DX%,DY%	Position of castle door
TB	Time bonus awarded at end of frame
NTES	Number of notes to be played in the tune
P,D	Pitch and duration of current note
C$	Reply to "Another Game?"

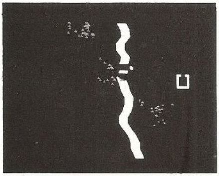
Table 1. The use of the variables in the program.

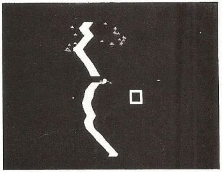
PROGRAM STRUCTURE

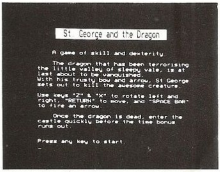
Statement	Action
10-60	Initialise arrays and envelopes
70-150	Print title and instructions
160	Set score to zero and set speed of dragon
170-220	Set up screen etc for the start of the frame
230	Initialise all variables
240-270	Place the dragon randomly on the far side of the river
280	Reset time to calculate time bonus
290-330	Main movement loop
340	Calculate time bonus
350-360	Select the correct tune and play it
370-430	Add up new score
440-490	Ask whether player wants another game
500-510	Data for tunes
520-660	Define graphics for St. George and the Dragon
670-790	MOVE ST. GEORGE
680	Save old position
690-700	Scan keyboard, and rotate St. George if necessary
710-720	Calculate new position
730	Rub out old position
740-760	Check whether or not he can move
770	Plot St. George in new position
780	Check to see if he has entered the castle
800-1020	DRAW SCREEN
830-860	Draw forests
870-950	Draw river and bridge
960-1010	Draw castle
1030-1220	MOVE ARROW
1040-1130	Check whether space bar has been pressed. If it has, and he can shoot an arrow, then initialise arrow variables
1140	Don't move the arrow if he hasn't shot it
1150-1170	Move arrow
1180-1220	Arrow has hit something — find out what, kill dragon if necessary
1230-1350	MOVE DRAGON
1240	If the dragon is breathing fire, then continue to do so
1250-1270	Rub out dragon from old position
1280	Change the direction he is facing randomly
1290	Move dragon depending on the direction he is facing, and the position of St. George
1300-1320	Check whether the dragon can move
1330	Plot the dragon in its new position
1340	Check if dragon has hit St. George
1360-1380	Procedure to plot the dragon
1390-1440	Procedure to breathe fire from the dragon's mouth
1430	Check whether St. George is engulfed in fire
1450-1490	Procedure executed when the dragon is killed
1460	Open castle gate
1470-1480	Rub out dragon and flame
1500-1520	Procedure executed when the human is killed

Table 2. Explanation of the lines of the program.





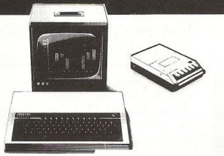




```
   10  MODE7
   20  ENVELOPE 1,1,0,0,0,0,0,0,10,0,-4,-2,100,100
   30  ENVELOPE 2,1,2,-2,2,2,4,2,100,-1,-1,-1,100,20
   40  DIM X%(8),Y%(8):FORLOOP=1 TO 8:READX%(LOOP),Y%(LOO
P):NEXTLOOP
   50  DATA 12,0,9,9,0,12,-9,9,-12,0,-9,-9,0,-12,9,-9
   55  HISC=0
   60  PROCDEFCHARS
   70  PRINT'''
   80  FORTITLE=1 TO 2
   90    PRINTTAB(3);CHR$(141);CHR$(131);CHR$(157);CHR$(1
32);"St. George and the Dragon  ";CHR$(156)
  100    NEXT TITLE
  110  PRINT''"    A game of skill and dexterity."
  120  PRINT'"    The dragon that has been terrorisingthe
 little valley of sleepy vale, is at last about to be van
quished."
  130  PRINT"With his trusty bow and arrow, St Georgesets
 out to kill the awesome creature."
  140  PRINT'"Use keys ""Z"" & ""X"" to rotate left and
 right, ""RETURN"" to move, and ""SPACE BAR""to fire an a
rrow."
  150  PRINT'"    Once the dragon is dead, enter the  cas
tle quickly before the time bonus    runs out."
  160  SCORE=0:SP=.4:FL=FALSE
  170  PRINT''"Press any key to start."
  180  *FX15,1
  190  C=GET
  200  MODE1
  210  PROCSCREEN
  220  VDU23;8202;0;0;0;
  230  A%=0:D%=1:X%=8:Y%=512:GCOL3,3:MOVEX%,Y%:VDUD%+223
  240  F%=RND(200)+800:G%=RND(600)+200:FORI%=F% TO F%+64
STEP4:FORJ%=G%-32 TO G% STEP4:IF POINT(I%,J%)=0 NEXTJ%,I%
:GOTO260
  250    I%=F%+64:J%=G%:NEXTJ%,I%:GOTO240
  260  K%=5:DR=TRUE:FINISHED=FALSE
  270  GCOL3,1:PROCPLOTDRAGON:GCOL3,3
  280  TIME=0
  290  REPEAT
  300    PROCMOVEHUMAN
  310    PROCSHOOT
  320    IF DR THEN PROCMOVEDRAGON
  330    UNTIL FINISHED
  340  TB=INT((TIME-12000)/10)*(TIME<12000)
  350  IF DR THEN RESTORE 500:NTES=11 ELSE RESTORE 510:NT
ES=13
  360  FORLOOP=1TONTES:READP,D:SOUND1,-10,P,D:SOUND1,0,0,
1:NEXTLOOP
  370  MODE7:PRINTTAB(5,10);"SCORE:";SPC(11);SCORE
  380  IF DR THEN 435
  390  PRINTTAB(5);"DRAGON KILLED:   ";INT(500*SP)
  400  PRINTTAB(5);"TIME BONUS:      ";TB
  410  SCORE=SCORE+INT(500*SP)+TB
  420  PRINT'TAB(5);"TOTAL:";SPC(11);SCORE
  430  SP=SP+.2:GOTO170
  435  IF SCORE>HISC THEN HISC=SCORE
  436  PRINTTAB(5);"HI-SCORE:";SPC(8);HISC
  440  PRINT'CHR$(129);"Too bad, you died!!!!!"
  450  PRINT'"Would you like another game?";
  460  *FX15,1
  470  C$=GET$:IFC$="Y" OR C$="y" THEN 160
  480  IF C$<>"N" AND C$<>"n" THEN 470
  490  END
  500  DATA 53,8,53,8,53,2,53,8,65,8,61,2,61,8,53,2,53,8,
53,2,53,8
  510  DATA53,2,53,2,53,4,69,2,69,2,69,4,81,2,69,2,81,2,6
9,2,53,2,53,2,53,4
  520  DEFPROCDEFCHARS
  530  VDU23,224,0,&74,&FA,&FF,&FA,&74,0,0
  540  VDU23,225,0,&1A,4,&7A,&FA,&F8,&F8,&70
  550  VDU23,226,16,&38,&54,&38,&7C,&7C,&7C,&38
  560  VDU23,227,0,&58,&20,&5E,&5F,&1F,&1F,&E
  570  VDU23,228,0,&2E,&5F,&FF,&5F,&2E,0,0
  580  VDU23,229,&E,&1F,&1F,&5F,&5E,&20,&58,0
  590  VDU23,230,&38,&7C,&7C,&7C,&38,&54,&38,16
  600  VDU23,231,&70,&F8,&F8,&FA,&7A,4,&1A,0
  610  VDU23,232,&20,&F8,&F9,&B,7,3,4,8,23,233,0,0,&82,&C
1,&E2,&F4,&58,&20
  620  VDU23,234,0,0,&41,&83,&47,&2F,&1A,4,23,235,4,&1F,&
9F,&D0,&E0,&C0,&20,&10
  630  VDU23,236,&1C,&1C,&7F,&7F,&6B,8,8,&1C
  640  VDU23,237,&28,&4A,&22,&19,&A6,&59,&22,&4,23,238,&B
0,&5E,&BF,&7E,&A8,&50,&A0,&40
  650  VDU23,239,&D,&7A,&FD,&7E,&15,&A,&5,&2,23,240,&14,&
52,&44,&98,&65,&9A,&20
  660  ENDPROC
  670  DEFPROCMOVEHUMAN
  680  H%=D%:I%=X%:J%=Y%
  690  IF INKEY(-98) D%=D%+1:IFD%=9 D%=1
  700  IF INKEY(-67) D%=D%-1:IFD%=0 D%=8
  710  IF NOT INKEY(-74) GOTO730
  720  X%=X%+X%(D%):Y%=Y%+Y%(D%)
  730  MOVEI%,J%:VDU223+H%
  740  C=POINT(X%+8,Y%+8) OR POINT(X%+24,Y%-24) OR POINT(
X%+8,Y%-24) OR POINT(X%+24,Y%+8)
  750  IF C=1 X%=X%-X%(D%)*3/4:Y%=Y%-Y%(D%)*3/4
  760  IF C>1 OR C=-1 X%=I%:Y%=J%
  770  MOVEX%,Y%:VDU223+D%
  780  IFX%>DX%-20 AND X%<DX%+40 AND Y%<DY% AND Y%>DY%-30
 FINISHED=TRUE
  790  ENDPROC
  800  DEFPROCSCREEN
  810  VDU5
  820  VDU19,1,2,0,0,0,19,2,6,0,0,0
  830  GCOL0,1
  840  FORZ=3TORND(5)+3
  850    X%=RND(1000)+100:Y%=RND(1000)+100:FORA=15TORND(3
5)+15:MOVEX%+RND(200)-100,Y%+RND(200)-100:VDU236:NEXTA
  860    NEXTZ
  870  A%=RND(400)+400
  880  D%=RND(600)+200
  890  FORB=1023 TO 0 STEP-50
  900    MOVEA%,B:MOVEA%+70,B
  910    IFB<=D% AND B>D%-50 GCOL0,0:GOTO930 ELSE GCOL0,2
  920    C%=A%+RND(100)-50:IFC%<400 OR C%>800 C%=A%
  930    PLOT85,C%,B-50:PLOT85,C%+70,B-50
  940    A%=C%
  950    NEXTB
  960  X%=RND(100)+900:Y%=RND(800)+50
  970  GCOL0,0:MOVEX%,Y%:DRAWX%+100,Y%:PLOT85,X%,Y%+100:P
LOT85,X%+100,Y%+100:GCOL0,3
  980  FORA=0 TO 12 STEP4
  990    MOVEX%+A,Y%+A:DRAWX%+100-A,Y%+A:DRAWX%+100-A,Y%+
100-A:DRAWX%+A,Y%+100-A:DRAWX%+A,Y%+A
 1000    NEXTA
 1010  DX%=X%+35:DY%=Y%+100
 1020  ENDPROC
 1030  DEFPROCSHOOT
 1040  C=INKEY(-99)
 1050  IF NOTC FLAG=FALSE:GOTO1140
 1060  IF FLAG=TRUE GOTO1140
 1070  FLAG=TRUE
 1080  IFA%<>0 GOTO1150
 1090  SOUND&11,2,40,20
 1100  A%=X%+12:B%=Y%-12
 1110  V%=X%(D%)*2:W%=Y%(D%)*2
 1120  MOVEA%,B%:PLOT1,V%,W%
 1130  GOTO1150
 1140  IF A%=0 ENDPROC
 1150  MOVEA%,B%:DRAWA%+V%,B%+W%
 1160  A%=A%+V%:B%=B%+W%
 1170  IFPOINT(A%,B%)=0 MOVEA%,B%:PLOT1,V%,W%:ENDPROC
 1180  IFPOINT(A%,B%)<>1 THEN 1210
 1190  IF A%<F% OR A%>F%+64 OR B%<G%-32 OR B%>G% GOTO1210
 1200  PROCDRAGONDEAD
 1210  A%=0
 1220  ENDPROC
 1230  DEFPROCMOVEDRAGON
 1240  IF FL<>0 PROCFLAME:GCOL3,3:ENDPROC
 1250  GCOL3,1
 1260  I%=F%:J%=G%
 1270  PROCPLOTDRAGON
 1280  K%=(K%+RND(3)+5)MOD8+1
 1290  F%=F%+(X%(K%)+5*SGN(X%-F%))*SP:G%=G%+(Y%(K%)+5*SGN
(Y%-G%))*SP
 1300  C=POINT(F%,G%) OR POINT(F%+64,G%) OR POINT(F%,G%-3
2) OR POINT(F%+64,G%-32)
 1310  IF C=-1 OR C>1 F%=I%:G%=J%
 1320  IF C=1 OR RND(10)=1 PROCFLAME:GCOL3,1
 1330  PROCPLOTDRAGON
 1340  IF X%>F%-20 AND X%<F%+52 AND Y%>G%-20 AND Y%<G%+20
PROCYOUDEAD:ENDPROC
 1350  GCOL3,3:ENDPROC
 1360  DEFPROCPLOTDRAGON
 1370  IFK%>3 AND K%<7 E%=232 ELSE E%=234
 1380  MOVEF%,G%:VDUE%,E%+1:ENDPROC
 1390  DEFPROCFLAME
 1400  FL=FL+1:IF FL=1 GCOL0,3:SOUND16,1,6,10:GOTO1420
 1410  IF FL=5 GCOL0,0:FL=0 ELSE ENDPROC
 1420  IFK%>3 AND K%<7 L%=F%-64:E%=237 ELSE L%=F%+64:E%=2
39
 1430  IF X%>L%-20 AND X%<L%+52 AND Y%>G%-20 AND Y%<G%+20
PROCYOUDEAD
 1440  MOVEL%,G%:VDUE%,E%+1:ENDPROC
 1450  DEFPROCDRAGONDEAD
 1460  GCOL0,0:MOVEDX%,DY%:MOVEDX%+30,DY%:PLOT85,DX%,DY%-
12:PLOT85,DX%+30,DY%-12
 1470  PROCPLOTDRAGON
 1480  REPEAT PROCFLAME:UNTIL FL=FALSE
 1490  DR=FALSE:GCOL3,3:ENDPROC
 1500  DEFPROCYOUDEAD
 1510  FINISHED=TRUE
 1520  ENDPROC
```

BBC
It's a Winner!

G.T.M.
BBC Microcomputer System A & B



BBC Machines
Model 'A' £299
16K RAM 32K ROM. Full colour, high-resolution graphics (+£7 p+p)
Model 'B' 32K RAM 32K ROM £399
16 Colour graphics (+£7 p+p)
14" RGB Colour Monitor (as used in the BBC Computer programme) £279 (+£9.50 courier)
12" Green Monitor £95 (+£6.50 courier)
Cassette Player. Includes DIN to DIN lead £28
Battery mains option (+£2 p+p)
BBC Joystick £13 p.pair (+£1 p+p)

BBC Software
Snapper, Planetoid, Monster, Graphs, Charts and Creative Graphics. All at £9.95
Arcade Action £11.90

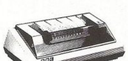
Atoms
Atom kit £135 (+£3.50 p+p)
Colour Atom £199
Complete with 4 software cassettes (+£3 p+p)
Atom Disk Pack £335 (+£2.50 p+p)

Atom Software
All the latest Acornsoft software in stock. Atomcalc, electronic spread sheet. FORTH, LISP, Adventure etc.

Accessories for BBC Computers
Cassettes per 10 £4 (+£1 p+p)
Verbatim single sided double density disks, box of 10 £19.99 (+£1 p+p)
GP80 and GP100 ribbon £4.75 (+£1 p+p)
Printer Cable (parallel) £15 (+£1 p+p)
6522 buffers £4.75 (+£1 p+p)
DIN to Jack cassette leads £3.50 (+£1 p+p)
BBC Machine dust covers £3.95 (+£1 p+p)

Books
BBC 30 hour BASIC £5.50 (+£1 p+p)
Practical Programs for Atom & BBC £5.95 (+£1 p+p)
BASIC Programming on the BBC Micro £5.95 (+£1 p+p)
Programming the 6502 £11.75 (+£1 p+p)

All the products are the official versions, beware of imitations, they will invalidate your guarantee.



We've also got the complete range of Acorn BBC and Atom products in stock including:

Printers
Acorn GP 80A Printer £199
Lowest ever price! (+£4.50 p+p)
Acorn GP 100A Printer £228 (+£4.50 p+p)

Lynq Wordprocessor
An FDS Product for professional use. Full support and service.

* **NEW SHOWROOM** *

ALL PRICES INCLUDE VAT. FOR FURTHER DETAILS AND MAIL ORDER LIST SEND LARGE S.A.E.

G.T.M.
864 York Road, Leeds.
TEL: 0532 865118

SOFTWARE FROM ASP

THE WHITE BARROWS Program approximately 8K
Somewhere amid this maze of burial chambers lurks an Evil Sorcerer whom you need to trap. Trouble is, he's protected by Trolls, Dwarves, Serpents and the occasional Dragon or two! Your magic staff will block the tunnel to prevent him escaping...unless, that is, he outwits you.

A real brain twister, White Barrows requires both brains and brawn from its players. It's no good just hacking your way through the Barrows and hoping to fall over the Sorcerer. Eventually you'll meet a Dragon -- and they don't hack easily! You'll need all your cunning and strength to survive this one for long.

CONQUERING EVEREST Program approximately 11K
So, you think climbing mountains is all about scrambling over rocks? This superb piece of programming will soon change all that!

You are in charge of an expedition comprising 18 climbers, 34 Sherpas and 40 Porters. There is food, tents and equipment for all, including the oxygen you'll need as you near the summit. Trouble is, it all starts at the BOTTOM of the mountain and you have to get it all to the TOP! Each route upward must be forced and any camp you make must be properly supplied, otherwise the expedition members will retreat down the mountain.

The monsters in this game are avalanches, starvation, storms and, worst of all, bad planning! A real, thinking man's adventure, Everest will test your skill and forward planning to the limit.

TWO MASSIVE ADVENTURES FOR JUST £11.45 ALL INCLUSIVE!

Fill in the coupon and return it to:

ASP Software, ASP Ltd, 145 Charing Cross Road, London WC2H 0EE

Computing Today Software

Please send me . . . tape(s) of White Barrows and Everest.
PET ☐ VIC-20 ☐
Sharp MZ-80K ☐ MZ-80A ☐
TRS-80 ☐ BBC ☐
At £11.45 inclusive.

I enclose my cheque/Postal Order/International Money Order (delete as necessary) for:
£ . . . . . (Made payable to ASP Ltd)
or Debit my Access/Barclaycard (delete as necessary)

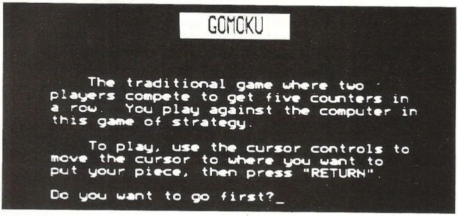
☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Please use BLOCK CAPITALS
NAME(Mr/Mrs/Miss) . . . .
ADDRESS . . . .
. . . . POSTCODE . . . .
Signature . . . . Date . . . .

H Roberts

GOMOKU

The traditional Chinese board game makes the transition to the small screen of your BBC Micro.



This is the traditional Chinese game where two players compete to make a line of five stones horizontally, vertically or diagonally. It is normally played on a 19 by 19 GO board, but has been known under the name of 'connect five' or 'five in a row' on an infinite board.

The method used to determine the computer's move is a simple but effective one. Each possible line of five stones is given a bias depending on the number of stones of each type in the line. A line including stones of both types is worthless, because it is impossible for either player to win using that line. Lines with three or four stones of one type are important, and thus have a high bias, and lines with only one or two stones have very small biases. Each square on the board is given a bias which is the sum of the biases of all the lines running through it, and the square with the highest bias is the one on which the computer plays.

Calculating the bias for every square on a 19 by 19 board would be very time consuming so a running total of the biases for every square is stored in the table BIAS%. The entries in this table are altered after every move by the procedure PROCUPDATEBOARD, but only the entries in the immediate vicinity of the move need to be altered, so this is quicker.

This method has also been used effectively in programs for three dimensional noughts and crosses, and 'connect four'.

To make your move, use the cursor controls to position the cursor where you want to play your piece, then press Return. The computer takes about 10 seconds to make its move, and plays reasonably well.

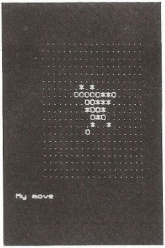
VARIABLES

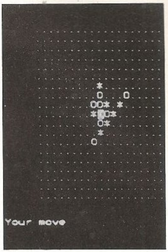
BOARD%(X,Y)	Board array: 0 = EMPTY 1 = HUMAN'S PIECE —1 = COMPUTER'S PIECE
BIAS%(X,Y)	Running total of bias on each square
H%(N)	Bias for N human pieces in a row
C%(N)	Bias for N computer pieces in a row
X%(A),Y%(A)	Direction vectors for each possible line through a square
LOOP	General loop variable
HUMAN	human = 1 for readability
COMPUTER	computer = —1 for readability
X%,Y%	Current square on the board
TURN	= 1 during the human's turn = —1 during the computer's turn
FINISHED	Is set true when the game is over
GO	Holds the number of the present turn
C$	Single character reply to various questions
C	ASCII character input during human's turn
BIG%	The biggest bias currently found on the board
I%,J%	The square with the highest bias (The one the computer thinks is best)
START	The start of the current line
XL%,YL%	The current position in that line
LONE%	Counter (1-5) through that line
I%,J%	Direction vector for that line
H,C	The number of human and computer counters in that line
BIAS%	The bias to be added to the squares in that line
DISP	Loop for flashing the computer's winning line

PROGRAM STRUCTURE

Statement	Action
10	Dimension arrays
20-40	Display the title and instructions
50-100	Read in date and clear board arrays
110-160	Determine who has the first turn
170-200	Set up variables and display for main program loop
210	Make computer's first turn
220-260	Main loop
270-320	Ask human whether he wants another game
330-350	Plot board
360-510	HUMAN'S TURN
370-390	Set up cursor
400-470	Move cursor
480-500	Check the square is empty and update the board
520-590	COMPUTER'S TURN
530	"My move"
540-570	Find the square with the highest bias
580	Play on that square
600-910	UPDATE THE BOARD
610-670	Flash square and place counter there
680-700	For each line running through the square where the counter was played:
710-790	Find out how many human and computer counters it contains

800-810	Check whether it is a winning line for either player
820-830	Calculate the difference in bias caused by this counter being played
840-880	And adjust the bias on each square in the line
890-900	Do this for all the lines running through the square
920-940	Returns the bias from a line of H human counters and C computer counters
950	Check whether X%,Y% is off the board
960-990	Human has won
1000-1060	Computer has won, so flash winning line
1070-1090	Display condescending message
1100-1110	Wait for TI|00 seconds
1120-1160	The game is a draw (somewhat unlikely)






```
  10  DIM BOARD%(19,19),BIAS%(19,19),H%(5),C%(5),X%(4),Y
%(4)
  20  MODE7:PRINT'''':FORLOOP=1 TO 2:PRINTTAB(13);CHR$(1
31);CHR$(157);CHR$(132);CHR$(141);"GOMOKU   ";CHR$(156):N
EXTLOOP
  30  PRINT'''"    The traditional game where two      P
layers compete to get five counters in a row.  You play a
gainst the computer inthis game of strategy."
  40  PRINT'"    To play, use the cursor controls to mov
e the cursor to where you want to    put your piece, then
 press ""RETURN""."
  50  HUMAN=1:COMPUTER=-1
  60  FORLOOP=0 TO 5:READ H%(LOOP),C%(LOOP):NEXTLOOP
  70  FORLOOP=1 TO 4:READ X%(LOOP),Y%(LOOP):NEXTLOOP
  80  DATA 0,0,1,4,8,10,30,200,1000,5000,0,0
  90  DATA 1,0,1,1,0,1,-1,1
 100  FORX%=1 TO 19:FORY%=1 TO 19:BOARD%(X%,Y%)=0:BIAS%(
X%,Y%)=0:NEXTY%:NEXTX%
 110  *FX4,1
 120  PRINT'"Do you want to go first?";
 130  TURN=HUMAN
 140  C$=GET$:IFC$="N" THEN TURN=COMPUTER:GOTO170
 150  IFC$<>"Y" THEN 140
 160  IF RND(2)=2 THEN PRINT"Y"'"Well you can't 'cause I
 m going to!!":PROCWAIT(400):TURN=COMPUTER
 170  VDU23;8202;0;0;0;
 180  PROCPLOTBOARD
 190  FINISHED=FALSE
 200  GO=1
 210  IF TURN=COMPUTER THEN PROCUPDATEBOARD(10,10,COMPUT
ER):GO=2:TURN=HUMAN
 220  REPEAT
 230    IF TURN=HUMAN PROCHUMANMOVE ELSE PROCCOMPUTERMOV
E
 240    GO=GO+1:IF GO=370 PROCDRAW
 250    TURN=-TURN
 260    UNTIL FINISHED
 270  PRINT'"Do you want another game?";
 280  *FX15,1
 290  C$=GET$:IFC$="Y" THEN RUN
 300  IF C$<>"N" THEN 290
 310  PRINT''''
 320  END
 330  DEFPROCPLOTBOARD
 335  CLS:PRINT
 340  FORLOOP=1 TO 19:PRINTTAB(10);STRING$(19,"."):NEXT
 350  ENDPROC
 360  DEFPROCHUMANMOVE
 370  PRINTTAB(5,22);"Your move"
 380  X%=10:Y%=10
 390  VDU23;10,64,0;0;0;
 400  REPEAT:VDU7
 410    REPEAT PRINTTAB(X%+9,Y%);
 420      C=GET
 430      IF C=136 X%=X%-1:IFX%=0 X%=19
 440      IF C=137 X%=X%+1:IFX%=20 X%=1
 450      IF C=138 Y%=Y%+1:IFY%=20 Y%=1
 460      IF C=139 Y%=Y%-1:IFY%=0 Y%=19
 470      UNTIL C=13
 480    UNTIL BOARD%(X%,Y%)=0
 490  VDU23;8202;0;0;0;
 500  PROCUPDATEBOARD(X%,Y%,TURN)
 510  ENDPROC
 520  DEFPROCCOMPUTERMOVE
 530  PRINTTAB(5,22);"My move   "
 540  BIG%=0
 550  FORX%=1 TO 19:FORY%=1 TO 19
 560      IF BIAS%(X%,Y%)>BIG% THEN IF BOARD%(X%,Y%)=0 T
HEN I%=X%:J%=Y%:BIG%=BIAS%(X%,Y%)
 570      NEXTY%:NEXTX%
 580  PROCUPDATEBOARD(I%,J%,TURN)
 590  ENDPROC
 600  DEFPROCUPDATEBOARD(X%,Y%,TURN)
 610  BOARD%(X%,Y%)=TURN
 620  FORLOOP=1 TO 15
 630    PRINTTAB(X%+9,Y%);"."
 640    PROCWAIT(4)
 650    IF TURN=HUMAN PRINTTAB(X%+9,Y%);"*" ELSE PRINTTA
B(X%+9,Y%);"O"
 660    PROCWAIT(4)
 670    NEXTLOOP
 680  FOR LOOP=1 TO 4
 690    I%=X%(LOOP):J%=Y%(LOOP)
 700    FOR START=0 TO 4
 710      XL%=X%-I%*START:YL%=Y%-J%*START
 720      H=0:C=0
 730      FOR LONE%=1 TO 5
 740        IF FNOFB(XL%,YL%) THEN LONE%=5:NEXTLONE%:GOT
O890
 750      C%=BOARD%(XL%,YL%)
 760      IF C%=HUMAN THEN H=H+1
 770      IF C%=COMPUTER THEN C=C+1
 780      XL%=XL%+I%:YL%=YL%+J%
 790      NEXT LONE%
 800    IF TURN=HUMAN AND H=5 THENPROCHUMANWON:ENDPROC
 810    IF TURN=COMPUTER AND C=5 THENPROCCOMPUTERWON:END
PROC
 820    BIAS%=FNBIAS(H,C)
 830    IF TURN=HUMAN THEN BIAS%=BIAS%-FNBIAS(H-1,C) ELS
E BIAS%=BIAS%-FNBIAS(H,C-1)
 840    XL%=X%-I%*START:YL%=Y%-J%*START
 850    FOR LONE%=1 TO 5
 860      BIAS%(XL%,YL%)=BIAS%(XL%,YL%)+BIAS%
 870      XL%=XL%+I%:YL%=YL%+J%
 880      NEXT LONE%
 890    NEXT START
 900  NEXT LOOP
 910  ENDPROC
 920  DEFFNBIAS(H,C)
 930  IF H<>0 AND C<>0 =0
 940  =H%(H)+C%(C)
 950  DEFFNOFB(X%,Y%)=X%>19OR X%<1OR Y%>19OR Y%<1
 960  DEFPROCHUMANWON
 970  CLS:PRINTTAB(10,5);"Oh blow. You beat me."
 980  PROCWAIT(500):PRINT'TAB(5);"I never liked this gam
e anyway!!"
 990  FINISHED=TRUE:ENDPROC
1000  DEFPROCCOMPUTERWON
1005  G$="."
1010  FOR DISP=1 TO 100
1020    XL%=X%-I%*START:YL%=Y%-J%*START
1030    FORLONE%=1TO5
1040      PRINTTAB(XL%+9,YL%);G$
1050      XL%=XL%+I%:YL%=YL%+J%:NEXTLONE%
1055    IFG$="." G$="O" ELSE G$="."
1060    NEXT DISP
1070  CLS:PRINTTAB(10,5)
1080  FORLOOP=1TO2:PRINTTAB(8);CHR$(141);"HA. HA. I BEAT
YOU!!":NEXT
1090  FINISHED=TRUE:ENDPROC
1100  DEFPROCWAIT(TI)
1110  TIME=0:REPEATUNTILTIME=TI:ENDPROC
1120  DEFPROCDRAW
1130  REM **THIS IS RATHER UNLIKELY**
1140  CLS:PRINTTAB(10,5);"I don't believe it,"
1150  PRINT'TAB(12);"IT'S A DRAW!!!!!"
1160  FINISHED=TRUE:ENDPROC
```

S Draper

MARS-LANDER

Landing your spaceship on Mars can be tricky, but you can prepare for it by using this flight simulation program.

Mars-Lander is a flight simulation program in which you must land your craft under control at one of three Mars bases. Unfortunately you only have a limited amount of fuel which soon goes on the higher gravities. Points are scored for a safe landing in the least possible time. There is also a bonus available depending on the amount of fuel you have left over after landing.

The player may choose what gravity he wishes to land under (0-10), and higher points are scored for high gravity landings (if you manage to pull it off). The program contains instructions, and so is easy to use without further explanation — just beware the horizontal drift.

TECHNICAL DETAILS

In order to make Mars-Lander fit in a model A I have split it up into two programs, the first of which sets up the sound envelopes and prints out the instructions whilst the second is the game itself. The first program contains nothing special but note the use of double quotes in line 220 — this is called a quote image and is the way we make the computer print A '"' in a print statement.

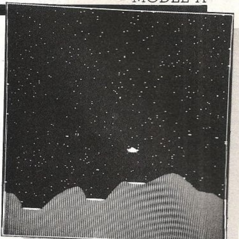
In the main program I have used XOR graphics for the lander (see previously), and OR graphics for the stars. The reason for using OR graphics here was to stop stars appearing over the landscape (the land is colour 3, the stars colour 1).

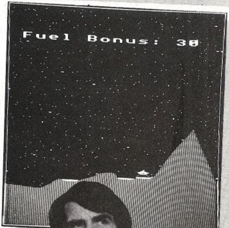
Most landscape creation programs produce their landscapes by randomly altering the angle of the slope as they go along. However this technique tends to give angular and jerky terrain which does not look very realistic, and so I have adopted a slightly different technique — that of modifying the rate of change of the angle of the slope. This gives smoother curves and, I think, looks far more realistic.

Mars-Lander will just fit in a model A.

PROGRAM STRUCTURE

Statement	Function	Action
Lines 10-50	Set up	Sets up arrays and constants.
Lines 60-90	Bases	Produces the positions of the three bases.
Lines 120-130	Input	Finds out what gravity to use.
Lines 150-190	Initialize	Selects the correct palete, removes the cursor and sets a few variables.
Lines 200-260	Landscape	Draws in the planet's surface.
Line 270	Border	Draws the border around the landscape.
Line 277	Stars	Draws in the stars.
Line 360	Instruments 1	Sets up a text window and prints the instrument headings.
Line 370	Preflight	Sets up altitude, speed, etc.
Line 395	Loop	Starts timing your landing and commences the flight.
Lines 420-440	Controls	Gets the keyboard controls.
Lines 445-510	Action	Acts on the controls.
Lines 520, 530	Loopend	Updates your instruments and goes back to the start of the main loop if you haven't landed or crashed.
Lines 535-630	Results	Prints out the results of your effort and the current high score then finds out if you want another go.
Lines 640-770	Instruments 2	Defines a procedure to draw your instruments.
Lines 780-810	Lander	Plots your vehicle on the screen.
Lines 820-910	Bonus	Works out your landing and fuel bonuses with appropriate effects.
Lines 920-960	Crash	Performs all the special effects when you crash.
Lines 970-990	Flat	Defines a function to check whether or not you have landed on one of the landing pads.
Line 1000,1010	Wait	Delays for a specified time.





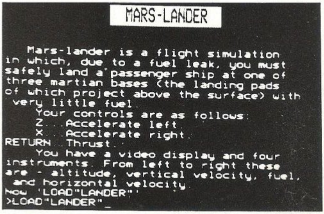
```
 10 ENVELOPE1,1,1,1,1,100,100,100,30,0,0,-5,110,110
 12 ENVELOPE2,2,0,0,-1,1,1,255,0,0,0,0,0,0
 13 ENVELOPE3,3,0,0,0,0,0,0,126,-2,-1,-1,126,100
 20 MODE7
 30 PRINT'''
 40 FORI=1TO2
 50   PRINTSPC(12);:VDU131,141,157,132
 60   PRINT"MARS-LANDER  ";CHR$(156)
 70   NEXT
 80 PRINT''"   Mars-lander is a flight simulation"
 90 PRINT"in which, due to a fuel leak, you must"
100 PRINT"safely land a passenger ship at one of"
110 PRINT"three martian bases (the landing pads"
115 PRINT"of which project above the surface) with very
 little fuel."
120 PRINT"    Your controls are as follows:"
130 PRINT"    Z...Accelerate left."
140 PRINT"    X...Accelerate right."
150 PRINT"RETURN..Thrust."
160 PRINT"    You have a video display and four"
170 PRINT"instruments. From left to right these"
180 PRINT"are - altitude, vertical velocity, fuel, and
horizontal velocity."
200 VDU23,224,0,24,60,255,&66,60,66,129,23,225,0,34,&52
,&52,&F9,&89,&88,0
210 VDU23,226,0,&2F,&28,&28,&4F,&48,&88,0,23,227,0,&A2,
&22,&22,&3E,&22,&22,0
220 PRINT"Now 'LOAD""LANDER""'"
```

Listing 1. Setting up sound envelopes and printing instructions.

```
 10 DIMH(3),L(3)
 20 L(3)=10000:H(3)=L(3):HIGH=0
 30 MODE7
 50 VDU28,0,14,39,11
 60 X=192:P=0:FORI=0TO2
 80   L(I)=X+RND(300):IFL(I)>900THEN80
 90   H(I)=L(I)+80:X=H(I):NEXT
120 CLS:INPUT'"    What gravity would you like (0-10)",
G
130 IFABS(G-5)>5THEN120
150 MODE5:VDU19,3,1;0,0,19,1,3;0,0,19,2,6,0;0
170 !&FE00=&10200A:GCOL0,3
190 H=200:HR=-1:OH=2000:OX=OH:T=0:OK=TRUE
200 FORI=192TO1272STEP8:MOVEI,0:DRAWI,H
210   IFI>L(P)ANDI<H(P)THENGCOL0,2:PLOT69,I,H:GCOL0,3:N
EXT
220 IFI>L(P)-40THENHR=HR-3*RND(1)
230 IFI>=H(P)THENP=P+1:HR=RND(1)*12
240 HR=HR+RND(1)*4-2:H=H+HR:IFH<40THENHR=1ELSEIFH>500HR
=-HR
260 NEXT:H=200
270 GCOL0,2:MOVE192,0:DRAW192,1020:DRAW1272,1020:DRAW12
72,0:DRAW192,0
277 GCOL1,1:COLOUR1:FORI=0TO500:PLOT69,200+RND(1070),RN
D(1010)+8:NEXT
360 VDU28,0,30,2,0,10,225,226,227,5
370 HV=0:HT=1000:VV=0:F=900:CRASH=FALSE:X=200+RND(1000)
380 FORI=1TO4:MOVE48*I-32,512:DRAW48*I-16,512:NEXT
395 TIME=0:REPEATPROCLANDER
420   IFINKEY(-98)THENHV=HV-.8
```

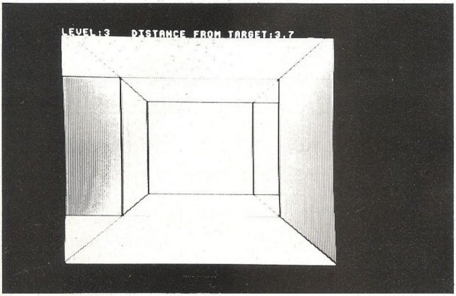
```
430   IFINKEY(-67)THENHV=HV+.8
440   IFINKEY(-74)ANDF THENT=T+(5-T)*.2ELSET=0
445   IFABS(HV)>30HV=30*SGN(HV)
460   X=X+HV:IFX<200THENX=1210
470   IFX>1210THENX=200
480   IFT>5THENT=5
490   VV=VV-G/2.5+T:HT=HT+VV:F=F-T:IFVV<-50THENVV=-50EL
SEIFVV>50THENVV=50
500   IFF<0THENF=0
510   SOUND16,-3*T,5,20
520   PROCINSTRUMENTS
530   UNTILCRASH
535 T=TIME+500:BONUS=0:S2=0
540 IFVV>-4ANDFNFLAT THENPROCBONUS ELSEPROCCRASH
550 MODE7
555 S1=50*((2E5/T)DIV50)
560 PRINT''"Time Bonus + Landing Bonus + Fuel Bonus"
570 PRINT"  =";CHR$131;CHR$136;S1;TAB(18);S2;TAB(33);BO
NUS
580 ADD=S1+S2+BONUS
585 PRINT'"Total score: ";ADD
586 IFADD>HIGH THENHIGH=ADD
587 PRINT'"High = ";CHR$134;CHR$136;HIGH
590 *FX15,0
600 PRINTTAB(0,18);"Would you like to try again?";:A=GE
T
610 IFA=78END
620 IFA<>89THEN600
630 GOTO30
640 DEFPROCINSTRUMENTS
650 MOVE24,900:GCOL0,0:DRAW24,512
660 IFPOINT(X+32,H)=3THENREPEATH=H+4:UNTILPOINT(X+32,H)
=0 ELSEREPEATH=H-4:UNTILPOINT(X+32,H)=3
670 AL=HT-H:IFAL>350THENAL=350
680 GCOL0,1:DRAW24,512+AL
690 IFAL<32THENCRASH=TRUE
700 GCOL0,0:MOVE72,200:DRAW72,900
710 GCOL0,1:PLOT69,72,512:DRAW72,512+VV*5
720 GCOL0,0:MOVE120,900:DRAW120,512
730 GCOL0,1:DRAW120,512+F*2/5
740 GCOL0,0:MOVE168,200:DRAW168,900
750 GCOL0,1:PLOT69,168,512:DRAW168,512+HV*10
760 IFF<100THENGCOL3,2:MOVE64,70:VDU70:SOUND2,1,50,10
770 ENDPROC
780 DEFPROCLANDER
790 GCOL3,2:MOVEX,HT:VDU224:MOVEOX,OH:VDU224
800 OX=X:OH=HT
810 ENDPROC
820 DEFPROCBONUS
825 S2=100*((600-HT+G*100)DIV100)
830 VDU4,28,4,5,18,4,23;8202;0;0;0:CLS
870 IFF=0THENENDPROC
875 PRINT"Fuel Bonus:";
880 REPEATSOUND1,-10,150,2:SOUND2,-8,198,2:SOUND3,-6,24
6,2:F=F-50
890   BONUS=BONUS+5:PRINTTAB(12,0);BONUS
900   PROCWAIT(30):UNTILF<50
910 ENDPROC
920 DEFPROCCRASH
930 SOUND17,2,200,10:SOUND16,3,7,10
940 FORI=7TO0STEP-1:VDU19,0,I,0;0:PROCWAIT(2):NEXT
950 PROCWAIT(200)
960 ENDPROC
970 DEFFNFLAT
980 FORI%=0TO2:IFX>L(I%)ANDX<H(I%)-48THENI%=2:NEXT:=TRU
E
990 NEXT:=FALSE
1000 DEFPROCWAIT(t)
1010 TIME=0:REPEATUNTILTIME=t:ENDPROC
```

Listing 2. The program for the game itself.



MAZE

It has to be an amazing game when you are trying to reach a target in a three-dimensional maze!



Supermaze is a 3-D maze game with a difference. Most 3-D maze games simply create an ordinary maze and display it to you in three dimensions as if you were inside it. Supermaze creates a three-dimensional 3-D maze and displays that to you as if you were inside it. This means that not only can you go forwards, backwards left and right, but also up and down!

PLAYING THE GAME

When the program is run the first thing to do is choose the size of maze you want — size 2 is a two by two by two maze, size 3 a three by three by three, and so on. When you have entered your choice the program will create the maze — the amount of time this takes depends on the size of the maze: from a few seconds for a size two up to a few minutes for a size eight.

When the maze has been made your location will be displayed on the screen. Floors and ceilings are shown in white and walls in yellow — this is important to enable you to keep track of where you are since if you are moving up a vertical passage and then turn right into a side tunnel you will find not the floor, but a wall under your feet with the floor and ceiling to the sides!

Your controls are as follows:

↑ . . .Move in the direction you are facing — note that this is the only control that actually moves you, all the others just turn you to face in a different direction.
→ . . . Turn to your right.
← . . . Turn to your left.
U . . . Turn to face upwards.
D . . . Turn to face down.
↓ . . . Turn right around.

The object of the game is to reach a target hidden deep within the maze — this is a huge globe and you'll know it when you see it!

To aid you in your search you are given two instruments — one gives a continuous read out of your height within the maze (ie — the floor you're on), the other tells you how far you are from the target.

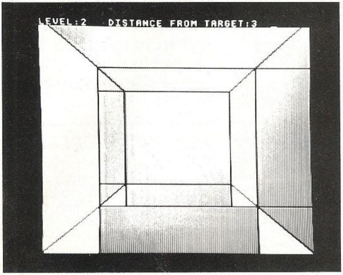
Good luck — you'll need it!

TECHNICAL DETAILS

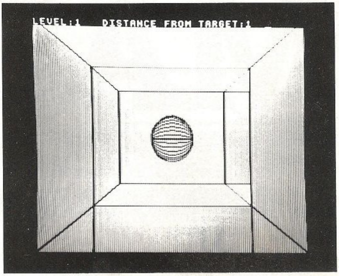
There is nothing very special in the actual programming of Supermaze, but the algorithm used is of interest. Firstly the way cells are represented in the computer. Each cell can have six exits — one to each adjacent cell — north, south, east, west, up, and down. Since each of these exits can either be open or closed I have used one bit to represent the state of each exit. Hence the maze is kept in one large array (MAZE%) with each entry giving the states of all six entrances to the corresponding cell. Notice that I have used an integer array here in order to save memory (Supermaze just fits in a model B!).

The other point of interest is the way in which the maze is created. This is done by starting at the target point (to ensure that the target lies within the maze) and lengthening the tunnel from there in a random direction until there is nowhere new to go (ie — the present cell is totally surrounded by cells which we have already visited). At this point the program simply backtracks by one cell and tries again. The process finishes when we have backtracked all the way back to the target. The byte vectors (see manual for an explanation of this term) LOCL and LOCH are used to store the positions of the cells which we have visited (so that we can tell where to backtrack to). It would have been simpler to have replaced both of these with one array, but that would use twice as much memory (even with an integer array) which explains why I decided to do it this way.

On the whole the program is self-explanatory since I have used descriptive procedure names and long variable names where possible. Anyway as a further aid, here is a table showing what does what:



Ah, the world's on its side again!



Dead ahead but mind the shaft!

Table 1. Explanation of the lines of the maze program.

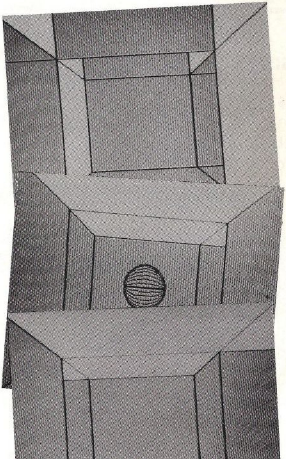
Statement	Function	Action
Lines 10-180	Set up and get size	Set up constants and input size of maze — FX4, 1 on line ★ 40 allows the cursor control keys to be used.
Lines 190-330	Create maze	Sets up the maze in the array MAZE% in the manner described above.
Lines 340-370	Place player	Sets up the player's direction and places him in an empty cell.
Line 380,390	Set screen	Chooses Mode1 and defines text and graphics windows.
Line 440	Draw start	Draws the view from your initial position.
Lines 410-480	Controls	Inputs from keyboard and turns you to face in the desired direction.
Lines 490-530	Move	Moves you and checks the legality of the move.
Line 540	Drawview	Draws what you can see.
Line 550	Loopend	Goes back to the beginning of the loop unless you have reached the target.
Lines 590-690	Draw	Draws view from position X,Y,Z — starts with the nearest cell and works away from you until a blank wall is encountered.
Lines 700-720	Instruments	Updates your instruments and draws the target if in view.
Lines 740-810	Distance	Works out the screen co-ordinates of the corners of the cell taking perspective into account.
Lines 820-960	Drawcell	Draws in the walls, floor and ceiling of the cell together with any entrances.
Lines 980-1030	Target	Draws in the target globe taking perspective into account.
Lines 1050-1110	Block	Defines a function of position which returns the value 'true' if there are no new exits to be made and 'false' otherwise.
Lines 1120-1140	Cellchange	Defines a procedure to alter one bit of a cell's entry in MAZE% in order to create a new exit.
Lines 1150-1190	Backtrack	Defines a procedure to backtrack along the path created (see text).
Lines 1200-1250	Data	Data for making turns (see below).

On examining the program you may wonder why I have not used a simple formula to work out what direction you are facing in after turning right or left. Unlike a 2-D maze the answer is that there isn't one — there are 24 different orientations you can be in within a cell — and these are connected by no simple formula! As a result five arrays are used — LEFT, RIGHT, UP, DOWN, and REVERSE. These give the new direction in which you are facing after turning from the previous one, eg, suppose you turn left from facing in direction 10, your new direction will then be LEFT?10 — hence all the data at the end of the program.

```
>LIST
    5 @%=0
   10 WON=FALSE
   15 FOUND=FALSE
   20 *FX4,1
   30 BACK=FALSE
   40 MODE7
   50 PRINT''''
   60 INPUT"WHAT SIZE MAZE WOULD YOU LIKE(1-8)",N
   70 IFN<1ORN>8THEN40
   80 DIMMAZE%(7,7,7),LOCL 350,LOCH 350,XINC(5),YINC(5),Z
INC(5),SWAP(5),UP 23
   90 DIMDOWN 23,LEFT 23,RIGHT 23,REVERSE 23,COL(5)
  100 TX=RND(N)-1:TY=RND(N)-1:TZ=RND(N)-1
  110 FORLOOP=0TO5:READXINC(LOOP),YINC(LOOP),ZINC(LOOP),S
WAP(LOOP),COL(LOOP):NEXT
  120 FORI=0TO23
  130   READRIGHT?I,LEFT?I,UP?I,DOWN?I,REVERSE?I
  140   NEXT
  150 POINTER=0:X=TX:Y=TY:Z=TZ
  160 FINISHED=FALSE
  170 REPEATLOCL?POINTER=Z:LOCH?POINTER=8*Y+X
  180   POINTER=POINTER+1
  190   PROCCELLCHANGE(X,Y,Z,128)
  200   IFFINISHED THEN310
  210   IFFNBLOCK(X,Y,Z)=0THENPROCBACKTRACK:GOTO200
  220   ROT=RND(6)-1:X1=X+XINC(ROT):Y1=Y+YINC(ROT):Z1=Z+Z
INC(ROT)
  230   IFX1<0 OR X1>=N OR Y1<0 OR Y1>=N OR Z1<0 OR Z1>=N
 THEN220
  240   IFROT>3AND(MAZE%(X1,Y1,Z1)AND48 ORMAZE%(X,Y,Z)AND
48) THENPROCBACKTRACK:GOTO200
  250   IF(MAZE%(X1,Y1,Z1)AND128)=0THEN270
  260   IFRND(2)=1ORROT=5THEN220ELSEBACK=TRUE
```

```
  270    PROCCELLCHANGE(X,Y,Z,2^ROT+128)
  280    PROCCELLCHANGE(X1,Y1,Z1,2^SWAP(ROT)+128)
  290    IFBACK THENBACK=FALSE:GOTO220
  300    X=X1:Y=Y1:Z=Z1
  310    UNTILFINISHED
  320 DIR1=0:DIR=0
  330 REPEATX=RND(N)-1:Y=RND(N)-1:Z=RND(N)-1
  340    UNTIL MAZE%(X,Y,Z)AND128
  350 XSTART=X:YSTART=Y:ZSTART=Z
  360 MODE1
  362 VDU28,0,0,39,0,24,0;0;1279;991;
  370 PROCDRAWVIEW(X,Y,Z)
  380 REPEAT
  390    CON=GET
  400    IFCON=82THENDIR1=0:Z=STARTZ:X=STARTX:Y=STARTY:PRO
CDRAWVIEW(X,Y,Z):UNTILFALSE
  410 IFCON=136THENDIR1=LEFT?DIR1
  420 IFCON=137THENDIR1=RIGHT?DIR1
  430 IFCON=138THENDIR1=REVERSE?DIR1
  440 IFCON=68THENDIR1=DOWN?DIR1
  450 IFCON=85THENDIR1=UP?DIR1
  460 DIR=DIR1 MOD6
  470 IFCON=139THENX1=X+XINC(DIR):Y1=Y+YINC(DIR):Z1=Z+ZIN
C(DIR)ELSEX1=X:Y1=Y:Z1=Z
  475 IF(CON=139)AND(MAZE%(X,Y,Z)AND(2^DIR))=0THEN390
  480 IFX1<0 OR X1>=N OR Y1<0 OR Y1>=N OR Z1<0 OR Z1>=N T
HENUNTILFALSE
  500 X=X1:Y=Y1:Z=Z1
  510 PROCDRAWVIEW(X,Y,Z)
  520 UNTILX=TX ANDY=TY ANDZ=TZ
  522 MODE7
  524 PRINT''''"CONGRATULATIONS - YOU'VE DONE IT!!"
  530 END
  540 DEFPROCDRAWVIEW(X,Y,Z)
  550 CLG
  560 D=-1
  570 X1=X:Y1=Y:Z1=Z
  580 REPEAT
  590    D=D+1
  600    PROCDRAWCELL(X1,Y1,Z1,D)
  610    IF(MAZE%(X1,Y1,Z1)AND 2^DIR)=0THENUNTILTRUE:GOTO6
40
  620 X1=X1+XINC(DIR):Y1=Y1+YINC(DIR):Z1=Z1+ZINC(DIR)
  630 IFX1<0 OR X1>=N OR Y1<0 OR Y1>=N OR Z1<0 OR Z1>=N T
HENUNTILTRUE ELSEUNTILMAZE%(X1,Y1,Z1)=0
  640 GCOL0,COL(C5):MOVEPX4+4,PY4-4:MOVEPX5-4,PY5-4:PLOT8
5,PX7+4,PY7+4:PLOT85,PX6-4,PY6+4
  646 PRINT'"LEVEL:";Z;TAB(10)"DISTANCE FROM TARGET:";INT
(10*SQR((Z-TZ)^2+(Y-TY)^2+(X-TX)^2))/10'"  ";
  649 IFFOUND THENPROCTARGET(D1)
  650 ENDPROC
  660 DEFPROCDRAWCELL(A,B,C,DI)
  670 PX0=640-750/(DI+1):PY0=512+600/(DI+1)
  680 PX1=640+750/(DI+1):PY1=512+600/(DI+1)
  690 PX2=640+750/(DI+1):PY2=512-600/(DI+1)
  700 PX3=640-750/(DI+1):PY3=512-600/(DI+1)
  710 PX4=640-750/(DI+2):PY4=512+600/(DI+2)
  720 PX5=640+750/(DI+2):PY5=512+600/(DI+2)
  730 PX6=640+750/(DI+2):PY6=512-600/(DI+2)
  740 PX7=640-750/(DI+2):PY7=512-600/(DI+2)
  750 C1=RIGHT?DIR1 MOD6:C2=LEFT?DIR1 MOD6:C3=UP?DIR1 MOD
6:C4=DOWN?DIR1 MOD6
  760 C5=REVERSE?DIR1 MOD6
  770 GCOL0,COL(C1):MOVEPX5,PY5:MOVEPX6,PY6:PLOT85,PX1,PY
1:PLOT85,PX2,PY2
  780 MOVEPX7,PY7:MOVEPX4,PY4:PLOT85,PX3,PY3:PLOT85,PX0,P
Y0
  790 GCOL0,COL(C3):MOVEPX4,PY4:MOVEPX5,PY5:PLOT85,PX0,PY
0:PLOT85,PX1,PY1
  800 MOVEPX6,PY6:MOVEPX7,PY7:PLOT85,PX2,PY2:PLOT85,PX3,P
Y3
  810 IFMAZE%(A,B,C)AND(2^C1)THENGCOL0,COL(C5):MOVEPX5,PY
5:MOVEPX1,PY5:PLOT85,PX6,PY6:PLOT85,PX1,PY6:GCOL0,COL(C3)
:PLOT85,PX2,PY2:MOVEPX1,PY1:MOVEPX1,PY5:PLOT85,PX5,PY5:GC
OL0,0:DRAWPX1,PY5:MOVEPX6,PY6:DRAWPX2,PY6
  820 IFMAZE%(A,B,C)AND(2^C4)THENGCOL0,COL(C5):MOVEPX6,PY
6:MOVEPX6,PY2:PLOT85,PX7,PY7:PLOT85,PX7,PY3:GCOL0,COL(C1)
:PLOT85,PX3,PY3:MOVEPX2,PY2:MOVEPX6,PY2:PLOT85,PX6,PY6:GC
OL0,0:DRAWPX6,PY2:MOVEPX7,PY7:DRAWPX7,PY2
  830 IFMAZE%(A,B,C)AND(2^C2)THENGCOL0,COL(C5):MOVEPX4,PY
4:MOVEPX0,PY4:PLOT85,PX7,PY7:PLOT85,PX0,PY7:GCOL0,COL(C4)
:PLOT85,PX3,PY3:MOVEPX0,PY0:MOVEPX0,PY4:PLOT85,PX4,PY4:GC
OL0,0:DRAWPX0,PY4:MOVEPX7,PY7:DRAWPX0,PY7
  840 IFMAZE%(A,B,C)AND(2^C3)THENMOVEPX5,PY5:GCOL0;COL(C5
):MOVEPX5,PY1:PLOT85,PX4,PY4:PLOT85,PX4,PY0:GCOL0,COL(C1)
:PLOT85,PX0,PY0:MOVEPX1,PY1:MOVEPX5,PY5:PLOT85,PX5,PY0:GC
OL0,0:DRAWPX5,PY5:MOVEPX4,PY4:DRAWPX4,PY0
  850 GCOL0,0:MOVEPX0,PY0:DRAWPX4,PY4:DRAWPX7,PY7:DRAWPX6
 PY6:DRAWPX5,PY5
  860 DRAWPX1,PY1:DRAWPX2,PY2:DRAWPX3,PY3:DRAWPX0,PY0:MOV
EPX2,PY2:DRAWPX6,PY6
  870 MOVEPX3,PY3:DRAWPX7,PY7:MOVEPX4,PY4:DRAWPX5,PY5:MOV
EPX0,PY0:DRAWPX1,PY1
  880 IFTX=A ANDTY=B ANDTZ=C THENFOUND=TRUE:D1=D
  885 ENDPROC
  886 DEFPROCTARGET(DEPTH)
  890 GCOL0,1:XD=200/(DEPTH+1)
  900 FORANG=0 TO 1.8 STEP.2
  910    MOVE640,512+XD*COSANG
  920    FORROUND=0 TO 7STEP.3
  930       DRAW640+XD*SINROUND,512+XD*COSANG*COSROUND
  940       NEXTROUND,ANG
  950    ENDPROC
  960    DEFFNBLOCK(A,B,C)
  970    COUNTER=0
  980    FORLOOP=0TO5
  990       IFA+XINC(LOOP)<0 OR A+XINC(LOOP)>=N OR B+YINC(L
OOP)<0 OR B+YINC(LOOP)>=N OR C+ZINC(LOOP)<0 OR C+ZINC(LOO
P)>=N THEN1010
 1000       IF(MAZE%(A+XINC(LOOP),B+YINC(LOOP),C+ZINC(LOOP)
)AND128)=0 THENCOUNTER=COUNTER+1
 1010       NEXTLOOP
 1020    =COUNTER
 1030    DEFPROCCELLCHANGE(X,Y,Z,BIT)
 1040    MAZE%(X,Y,Z)=MAZE%(X,Y,Z)OR BIT
 1050    ENDPROC
 1060    DEFPROCBACKTRACK
 1070    Z=LOCL?POINTER:Y=(LOCH?POINTER)DIV8:X=LOCH?POINTE
R MOD8
 1080    POINTER=POINTER-1
 1090    IFPOINTER<0THENFINISHED=TRUE
 1100    ENDPROC
 1110    DATA1,0,0,2,2,0,-1,0,3,2,-1,0,0,0,2,0,1,0,1,2,0,0
,-1,5,3,0,0,1,4,3
 1220    DATA1,3,5,4,2,2,0,11,10,3,3,1;17,16,0,0,2,23,22,1
,13,21,0,8,17
 1230     DATA7,15,8,0,16,10,23,7,21,14,16,5,20,6,15,19,9,
4,5,12,8,12,10,11,19
 1240    DATA14,6,1,9,23,20,18,9,1,22,9,19,16,17,8,17,4,18
,14,21,23,10,13,15,6
 1250    DATA5,16,14,18,7,15,7,2,12,5,21,13,12,2,4,11,22,1
5,13,20,12,8,22,23,9
 1260    DATA22,11,21,7,18,4,17,6,20,13,18,20,3,19,11,6,14
,19,3,10
```

Listing 1. The program for the game of Supermaze.



Jeremy Ruston

DISASSEMBLER #1

A disassembler can be an invaluable aid to unravelling the BBC Micro's ROMs.

A disassembler is a valuable tool (in the right hands) since it allows users of a computer to decode the mysteries of the ROMs which at first seem forbidden territory. This article describes my Rustonian disassembler, together with a tutorial on using it.

PROGRAM DESCRIPTION

The complete program appears in Listing 1, which may give you cause to wonder why similar programs are sold for anything up to £7 on cassette.

The operation of the program revolves around the DATA statements towards the beginning. These contain an entry for each of the 256 possible 6502 instructions (even though some of these are in fact illegal). Each entry takes the form of the number of bytes in the instruction, the instruction mnemonic and the addressing mode employed. The addressing mode is encoded in the following way:

Code & Mode	Example
0 — immediate	LDA #14
1 — absolute	LDA &358
2 — implied	PLA
3 — accumulator	ROL A
4 — pre-indexed indirect	LDA (&103,X)
5 — post-indexed indirect	LDA (&80),Y
6 — X indexed	LDA &7C00,X
7 — Y indexed	LDA &C000,Y
8 — indirect	JMP (&20A)
9 — relative	BNE LOOP

Thus this table does virtually all the work involved in the program.

All the program need do is interrogate the current location, ensure it contains a legal op-code (line 570), print the mnemonic (line 550) and add the addressing mode information (lines 600 to 700 — line 620 is not an error).

The program behaves correctly with the BRK instruction (used for fault handling).

To use the program, type RUN, enter the start address for disassembly and take out a notebook and pen if you have no printer. After each instruction is printed, the computer will await a keypress before continuing — paged mode is not terribly useful here. Line 710 follows each mnemonic with a series of underline characters, which make it easier to add notes to a printed listing. If you are disassembling to the screen, you may prefer to replace this line with 'PRINT'.

Typing in the program is rather difficult, thanks to the repetitive nature of the DATA statements, but they have been arranged to make the job easier. Once you have the program completed, check it very carefully before committing it to cassette.

USING THE PROGRAM

(The following is only directly applicable to OS 1.00, because that happens to be the MOS my computer is fitted with at the moment, users of other operating systems will have to employ a little more thought.)

For the sake of illustration, we will disassemble the part of the operating system concerned with handling *FX calls — this is illuminating from the point of view of demonstrating the program, and it reveals several calls not detailed in the **User Guide.**

At the simplest level, *FX follows the general form:

*FX A,X,Y

If the X or Y parameters are omitted they are assumed to be zero, which is why statements like *FX4 are legal.

If you wish to use *FX calls from assembly language, you load the 6502 registers indicated above and do a JSR to &FFF4. Thus the first place to start disassembling is &FFF4. The instruction at &FFF4 is JMP (&20A). Locations &20A and &20B normally contain the address &E786, so disassembly must be switched to this address.

A commented disassembly from address &E786 appears in Fig. 1. With my comments, the listing should be self explanatory for most readers.

The *FX routine revolves around a jump table at &E56E which contains the start address of each routine associated with each *FX call. However, as some calls are illegal, the table has had to be coded in an unusual way. For various values of 'A' (the call identifier), the routine's action is shown in Fig. 2 opposite.

This information is a combination of the contents of the jump table and the disassembly.

Thus we can now write a program to print out the start address of any *FX call we wish and listing to achieve this is given over the page in listing 2.

```
50 DATA 1brk2,2ora4,0***0,0***0,0***0,2ora1,2asl1,0***0
60 DATA 1php2,2ora0,1asl3,0***0,0***0,3ora1,3asl1,0***0
70 DATA 2bpl9,2ora5,0***0,0***0,0***0,2ora6,2asl6,0***0
80 DATA 1clc2,3ora7,0***0,0***0,0***0,3ora6,3asl6,0***0
90 DATA 3jsr1,2and4,0***0,0***0,2bit1,2and1,2rol1,0***0
100 DATA 1plp2,2and0,1rol3,0***0,3bit1,3and1,3rol1,0***0
110 DATA 2bmi9,2and5,0***0,0***0,0***0,2and6,2rol6,0***0
120 DATA 1sec2,3and7,0***0,0***0,0***0,3and6,3rol6,0***0
130 DATA 1rti2,2eor4,0***0,0***0,0***0,2eor1,2lsr1,0***0
140 DATA 1pha2,2eor0,1lsr3,0***0,3jmp1,3eor1,3lsr1,0***0
150 DATA 2bvc9,2eor5,0***0,0***0,0***0,2eor6,2lsr6,0***0
160 DATA 1cli2,3eor7,0***0,0***0,0***0,3eor6,3lsr6,0***0
170 DATA 1rts2,2adc4,0***0,0***0,0***0,2adc1,2ror1,0***0
180 DATA 1pla2,2adc0,1ror3,0***0,3jmp8,3adc1,3ror1,0***0
190 DATA 2bvs9,2adc5,0***0,0***0,0***0,2adc6,2ror6,0***0
200 DATA 1sei2,3adc7,0***0,0***0,0***0,3adc6,3ror6,0***0
210 DATA 0***0,2sta4,0***0,0***0,2sty1,2sta1,2stx1,0***0
220 DATA 1dey2,0***0,1txa2,0***0,3sty1,3sta1,3stx1,0***0
230 DATA 2bcc9,2sta5,0***0,0***0,2sty6,2sta6,2stx7,0***0
240 DATA 1tya2,3sta7,1txs2,0***0,0***0,3sta6,0***0,0***0
250 DATA 2ldy0,2lda4,2ldx0,0***0,2ldy1,2lda1,2ldx1,0***0
260 DATA 1tay2,2lda0,1tax2,0***0,3ldy1,3lda1,3ldx1,0***0
270 DATA 2bcs9,2lda5,0***0,0***0,2ldy6,2lda6,2ldx7,0***0
280 DATA 1clv2,3lda7,1tsx2,0***0,3ldy6,3lda6,3ldx7,0***0
290 DATA 2cpy0,2cmp4,0***0,0***0,2cpy1,2cmp1,2dec1,0***0
300 DATA 1iny2,2cmp0,1dex2,0***0,3cpy1,3cmp1,3dex1,0***0
310 DATA 2bne9,2cmp5,0***0,0***0,0***0,2cmp6,2dec6,0***0
320 DATA 1cld2,3cmp7,0***0,0***0,0***0,3cmp6,3dec6,0***0
330 DATA 2cpx0,2sbc4,0***0,0***0,2cpx1,2sbc1,2inc1,0***0
340 DATA 1inx2,2sbc0,1nop2,0***0,3cpx1,3sbc1,3inc1,0***0
350 DATA 2beq9,2sbc5,0***0,0***0,0***0,2sbc6,2inc6,0***0
360 DATA 1sed2,3sbc7,0***0,0***0,0***0,3sbc6,3inc6,0***0
370
380 DIM A%255,B%255,C%255,D%255,E%255
390
400 FOR T%=0 TO 255
410 READ A$
420 A%?T%=VAL(MID$(A$,1,1))
430 B%?T%=ASC(MID$(A$,2))
440 C%?T%=ASC(MID$(A$,3))
450 D%?T%=ASC(MID$(A$,4))
460 E%?T%=VAL(MID$(A$,5,1))
470 NEXT T%
480
490 INPUT "Enter start address:"A$
500 P%=EVAL(A$)
510
520 REPEAT
530 I%=?P%
540 L%=A%?I%
550 PRINT ;~P%;":";CHR$(B%?I%);CHR$(C%?I%)
    ;CHR$(D%?I%);" ";
560 M%=E%?I%
570 IF L%=0 THEN PRINT '"Unknown instruction":END
580 IF L%=2 THEN F%=P%?1
590 IF L%=3 THEN F%=P%!1 AND &FFFF
600 IF M%=0 THEN PRINT "#&";~F%;
610 IF M%=1 THEN PRINT "&";~F%;
620 IF M%=2 THEN
630 IF M%=3 THEN PRINT "A";
640 IF M%=4 THEN PRINT "(&";~F%;",X)";
650 IF M%=5 THEN PRINT "(&";~F%;"),Y";
660 IF M%=6 THEN PRINT "&";~F%;",X";
670 IF M%=7 THEN PRINT "&";~F%;",Y";
680 IF M%=8 THEN PRINT "(&";~F%;")";
690 IF M%=9 AND F%<128 THEN PRINT "&";~P%+F%+2;
700 IF M%=9 AND F%>127 THEN PRINT "&";~P%-254+F%;
710 PRINT " ";STRING$(69-POS,"_")
720 IF I%=0 THEN PROCbreak ELSE P%=P%+L%
730 L%=GET:*FX 15,1
740 UNTIL FALSE
750
760 DEF PROCbreak
770 PRINT "Error number:";P%?1
780 PRINT "Error message:";
790 P%=P%+2
800 REPEAT
810 VDU ?P%
820 P%=P%+1
830 UNTIL ?(P%-1)=0
840 PRINT
850 ENDPROC
```

Listing 1. Complete program for a disassembler.

```
E7B1:jsr &E539   Call &E539
E7B4:bvs &E7D0   Branch to &E7D0 if overflow flag set
E7B6:lda &E56F,Y } retrieve MSB from jump table
E7B9:sta &FB     }
E7BB:lda &E56E,Y } Retrieve LSB from jump table
E7BE:sta &FA     }
E7C0:lda &EF     Retrieve original contents of 'A'
E7C2:ldy &F1     Retrieve original contents of 'Y'
E7C4:bcs &E7CA   Branch to &E7CA if carry set
E7C6:ldy #&0     } Load contents of address in &F0/&F1 into 'A'
E7C8:lda (&F0),Y }
E7CA:sec         Set carry flag
E7CB:ldx &F0     Retrieve 'X'
E7CD:jsr &F06A   Call &F06A - does indirect jump to &FA/&FB
E7D0:ror A
E7D1:plp
E7D2:rol A
E7D3:pla
E7D4:clv
E7D5:rts
E7D6:ldy #&0     Zero 'Y'
E7D8:cmp #&17    } Go back to &E7A5 if 'A' < &17   (&17 = 23)
E7DA:bcc &E7A5   }
E7DC:php
E7DD:php
E7DE:pla
E7DF:pla
E7E0:jsr &F178

>RUN
Enter start address:&E786
E786:pha         Save 'A' on stack
E787:php         Save status register on stack
E788:sei         Disable interrupts
E789:sta &EF     Save 'A' in page zero
E78B:stx &F0     Save 'X' in page zero
E78D:sty &F1     Save 'Y' in page zero
E78F:ldx #&7     Put '7' in 'X'
E791:cmp #&74    } Branch to &E7D6 if 'A' < &74   (&74 = 116)
E793:bcc &E7D6   }
E795:cmp #&A0    } Branch to &E7A2 if 'A' < &A0   (&A0 = 160)
E797:bcc &E7A2   }
E799:cmp #&A6    } Branch to &E7DC if 'A' < &A6   (&A6 = 166)
E79B:bcc &E7DC   }
E79D:clc         Clear carry flag, ready for addition
E79E:lda #&A0    Put &A0 (160) in 'A'
E7A0:adc #&0     Add 0 to 'A'
E7A2:sec         Set carry flag, ready for subtraction
E7A3:sbc #&5D    Subtract &5D (93) from 'A'
E7A5:asl A       Arithmetic shift left on 'A' (= multiply by 2)
E7A6:sec         Set carry flag
E7A7:sty &F1     Save 'Y' in zero page
E7A9:tay         Transfer 'A' to 'Y'
E7AA:bit &25E    Adjust flags for an 'AND' with contents of &25E
E7AD:bpl &E7B6   Branch if bit 7 not set to &E7B6
E7AF:txa         Transfer 'X' to 'A'
E7B0:clv         Clear overflow flag
```

Fig. 1. A commented disassembly from address &E786.

```
A<&17                JSR ?(&E56E+A*2)+(?&E56F+A*2)*256
A>=&17 and A<&74     Bad command
A>=&74 and A<&A0     JSR ?(&E56E+(A-93)*2)+?(&E56F+(A-93)*2)*256
A>=&A0 and A<&A6     Bad command
A>=&A6 and A<&FF     JSR &E9AF
```

Fig. 2. The operation of the jump table.

When RUN, the program produces results like those shown in Fig. 3.

You may like to try disassembling from &F0CB, since that particular section of code is quite opaque.

```
10 REM *FX start address
20
30 REM (c) 1982 Jeremy Ruston
40
50 REM [Only tested on OS 1.00 - may work on others]
60
70 FOR T%=0 TO 159
80 IF T%<23 THEN PRINT "*FX ";T%;"--->&";~!(&E56E+T%*2) AND &FFFF
90 IF T%>=116 THEN PRINT "*FX ";T%;"--->&";~!(&E56E+(T%-93)*2) AND &FFFF
100 NEXT T%
110 PRINT "*FX 166 to *FX 255--->&E9AF"
```

Listing 2. A program to print out the start address of any FX call wanted.

```
>RUN
*FX 0--->&F0CB
*FX 1--->&E99B
*FX 2--->&E693
*FX 3--->&E9AA
*FX 4--->&E9AA
*FX 5--->&E989
*FX 6--->&E99B
*FX 7--->&E648
*FX 8--->&E646
*FX 9--->&E674
*FX 10--->&E66C
*FX 11--->&E9A8
*FX 12--->&E99F
*FX 13--->&E6B9
*FX 14--->&E6BA
*FX 15--->&F0BA
*FX 16--->&E719
*FX 17--->&DE80
*FX 18--->&E9DF
*FX 19--->&E9D4
*FX 20--->&CD08
*FX 21--->&F0C6
*FX 22--->&FAFB
*FX 116--->&FB1D
*FX 117--->&E875
*FX 118--->&E9F0
*FX 119--->&E6C7
*FX 120--->&F057
*FX 121--->&F0ED
*FX 122--->&F0EB
*FX 123--->&E18B
*FX 124--->&E630
*FX 125--->&E631
*FX 126--->&E619
*FX 127--->&E029
*FX 128--->&E762
*FX 129--->&E726
*FX 130--->&E73C
*FX 131--->&F097
*FX 132--->&D924
*FX 133--->&D927
*FX 134--->&D648
*FX 135--->&D7C3
*FX 136--->&E614
*FX 137--->&E63C
*FX 138--->&E474
*FX 139--->&E028
*FX 140--->&E9C9
*FX 141--->&E9C9
*FX 142--->&DBED
*FX 143--->&F178
*FX 144--->&EAFA
*FX 145--->&E426
*FX 146--->&FFA7
*FX 147--->&FFAB
*FX 148--->&FFB0
*FX 149--->&FFB4
*FX 150--->&EB0B
*FX 151--->&EB0F
*FX 152--->&E421
*FX 153--->&E4B6
*FX 154--->&EA16
*FX 155--->&EA27
*FX 156--->&E170
*FX 157--->&FFB9
*FX 158--->&EE79
*FX 159--->&EE8B
*FX 166 to *FX 255--->&E9AF
```

Fig. 3. The result of running the program in Listing 2.

CONCLUSION

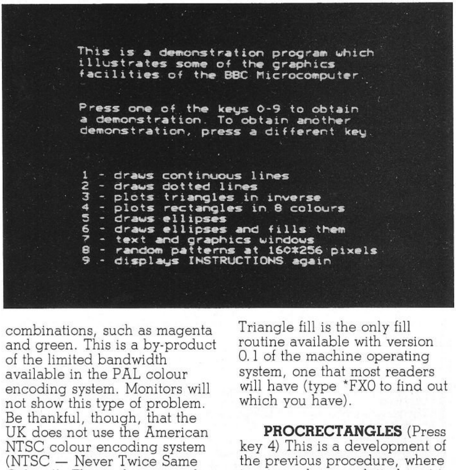
Disassembling the ROM is not a task for the beginner, but anyone with a modicum of knowledge of 6502 assembly language should have no trouble using this program to gain a useful insight into the ROMs of the BBC Micro.

Ian Nicholls

MULTIPLE GRAPHICS DEMO

If you want to amaze your friends with the graphics capabilities of your BBC Micro just read on.

One of the things the proud owner of a BBC Microcomputer often wants to do is to impress his or her friends with the graphics capabilities of the new machine. We have produced Listing 1 to provide you with a ready-made demonstration for just such an occasion. Careful scrutiny of the procedures contained in it will also yield a number of useful ideas which you can put into your file of valuable tips. The program will only run on a Model B or 32K Model A. It is written as a series of calls to procedures which are selected by pressing a numeric key 1 to 9. The program starts with a menu display in Mode 7 of the demonstrations available; during any demonstration pressing a numeric key from 1 to 8 will change to another demonstration, pressing key 9 will return to the menu (as will pressing the Escape key). The various procedures are as follows:

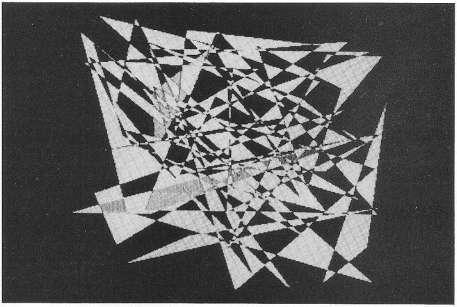


PROCINSTR (Press key 9) This is the procedure which displays the menu of available choices.

PROCDRAW(Z%) (Press keys 1 or 2) This procedure is used by two demonstrations, 1 — which draws a moiré-type pattern using solid straight lines, and 2 — which does the same using dotted straight lines. The second demonstration is a good test of your TV monitor's ability to show fine detail. Most TVs will show very bad interference with certain colour combinations, such as magenta and green. This is a by-product of the limited bandwidth available in the PAL colour encoding system. Monitors will not show this type of problem. Be thankful, though, that the UK does not use the American NTSC colour encoding system (NTSC — Never Twice Same Colour!). This is the reason for the familiar yellowy-green features of American politicians on satellite TV broadcasts.

PROCFILL (Press key 3) This is a demonstration of triangle-filling in black and white. The unusual feature is that the triangles are plotted in inverse mode (using PLOT 86,x,y). After a few seconds the picture is filled with a random pattern of black and white dots, and the triangles disappear as soon as they are drawn. Triangle fill is the only fill routine available with version 0.1 of the machine operating system, one that most readers will have (type *FX0 to find out which you have).

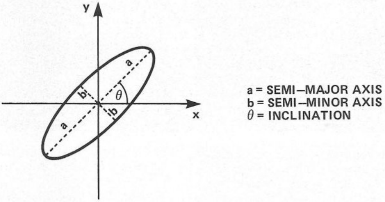
PROCRECTANGLES (Press key 4) This is a development of the previous procedure, where two triangles are plotted next to each other to give a rectangle. It uses Mode 2 to give eight colours plus eight flashing colours. It also draws the rectangles using the first three actions available in the first GCOL parameter, normal plotting, OR,AND. Use of EXOR and inverse plotting causes diagonal lines in the rectangles where the two triangles overlap. This is a result of the particular algorithm used by Acorn's BASIC programmers.



The inverse triangle drawing routine, Demo 3.

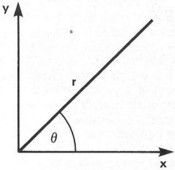
PROCELLIPSE(FILL%)

(Press key 5 or 6) The automatic drawing of curves is a feature hinted at by Acorn in a yet to be released graphics extension ROM. Until details of this are available we must make do with BASIC (or assembler) routines to draw curves. This procedure draws ellipses with random values for semi-major and semi-minor axes, inclination, and co-ordinates of the centre



NB If a = b, then we have a circle, and the value of θ has no effect.

The particular routine shown is of interest because it avoids the repetitive calculation of sine and cosine values. These are very slow processes in BASIC, since they involve a substantial number of multiplications using polynomial expansions. Most circle and ellipse drawing routines use the polar co-ordinate system:

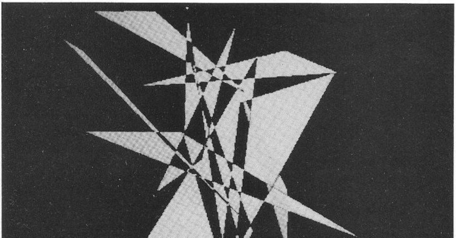


POLAR COORDINATES – r,θ
CARTESIAN COORDINATES – x,y

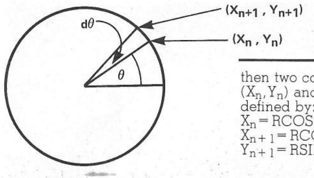
and vary the angle θ from 0 to 360 degrees in a number of steps, usually 20 or more, calculating a new point on the curve at each step, then joining it to the previous point with a straight line. At each step the cosine and sine of the angle θ have to be calculated.

The routine given uses recursion to determine each point by manipulating the co-ordinates of the previous point. The method is more easily described for a circle. We first of all decide how many points we want to use to define the circle. Since they will be joined by straight lines, the more points we use the smoother will be the outline of the circle, but the longer it will take to draw. Suppose we decide on 50 points, then the co-ordinates of each point will be (X_1,X_1), (X_2,Y_2) . . . . (X_{50},Y_{50}).

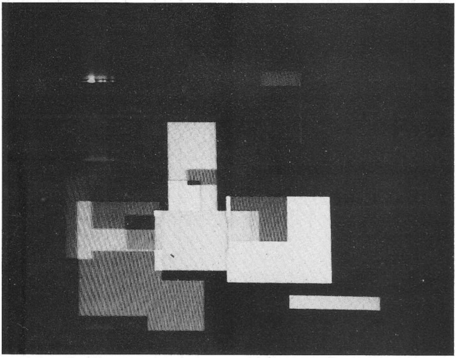
If dθ is the angle that we move through to get from one point on the circle to the next: (in our case this will be dθ= 360°/50)



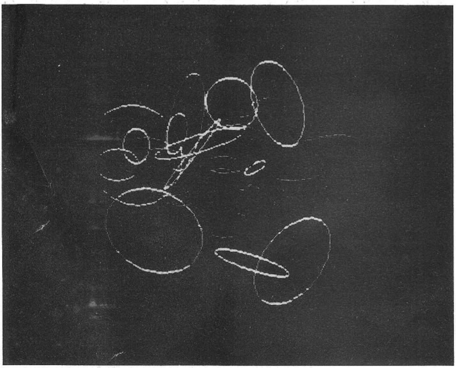
More inverse triangles produced by Demo 3.



then two consecutive points (X_n, Y_n) and (X_{n+1}, Y_{n+1}) are defined by:
$X_n = RCOS\theta : Y_n = RSIN\theta$
$X_{n+1} = RCOS(\theta + d\theta)$:
$Y_{n+1} = RSIN(\theta + d\theta)$



Demo 4 draws solid rectangles in Mode 2 giving all eight colours.



Demo 5 produces ellipses, see the text for more detail on this.

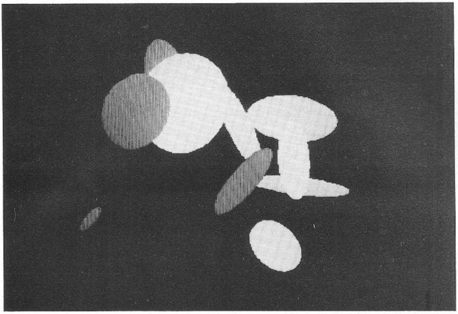
The trigonometry that we learnt at school, and have now forgotten, gives us formulae to convert $COS(\theta + d\theta)$ and $SIN(\theta + d\theta)$ into expressions involving only $COS\theta$, $SIN\theta$, $COS\ d\theta$ and $SINd\theta$:
$X_{n+1} = RCOS(\theta + d\theta)$
$= RCOS\theta COSd\theta - RSIN\theta SINd\theta$
$= X_n COSd\theta - Y_n SINd\theta$
$Y_{n+1} = RSIN(\theta + d\theta)$
$= RSIN\theta COSd\theta + RCOSOSINd\theta$
$= Y_n COSd\theta + X_n SINd\theta$

So, we only need to calculate $COSd\theta$ and $SINd\theta$ once and set initial values for X_1 and Y_1, and we have a method which only uses repetitive multiplications rather than trigonometric calculations. Did you follow all that? Good, you can now extend the principle to an ellipse with an inclination to the x-axis! Well, you cannot expect us to do everything for you, can you?

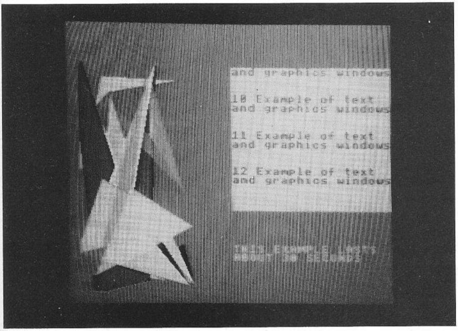
Anyway the procedure will draw random ellipses in three colours or, if you have pressed key 6, will fill them in as well.

PROCWINDOWS (Press key 7) One of the impressive, and useful, features of BBC BASIC is the ability to limit writing and drawing on the screen to restricted sections, known as text and graphics windows. The drawing program described elsewhere in this issue uses these features. This procedure provides another illustration having two separate windows. In the left-hand one, random triangles are drawn in four colours, and every so often the graphics window is cleared to a new background colour. On the right-hand side of the screen is a scrolling text window. Admittedly, the scrolling does hesitate slightly when the graphics window is cleared, but the procedure effectively demonstrates how text and graphics windows can both be active at (nearly) the same time.

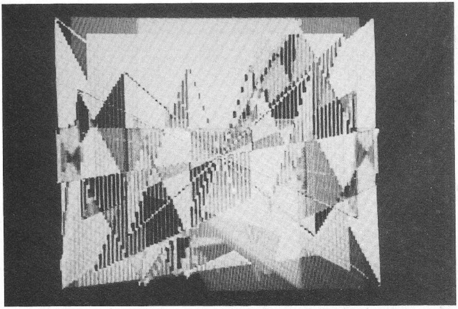
PROCTRIANGLES (Press key 8) If you have friends with epileptic tendencies then this demonstration is probably not for them. It uses Mode 2 and all 16 colour effects to show the speed with which triangles can be used to fill large areas of the screen with everchanging patterns. In addition, it uses values for the first parameter of the GCOL statement which are outside the defined limit (ie values of 5, 6 and 7). The result of this is to draw striped triangles: there is a whole new area for experimentation here. There are rumours that this feature is actually a fault on the video ULA chip. Certainly, the Torch microcomputer (the business version of the BBC micro) does ***not*** draw striped triangles when running this routine.



Demo 6 follows the trend set by Demo 5 but this time fills them in.



The powerful window function is shown in Demo 7.



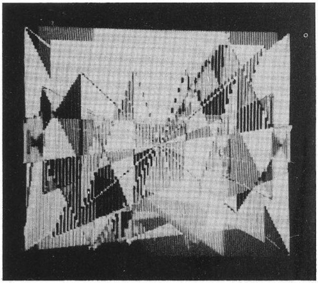
The apparently striped triangles of Demo 8 are caused by using an out-of-range GCOL statement.

```
100 REM  Graphics Demonstration
110 REM           by
120 REM  1 G Nicholis Dec 1982
130 REM
140 REM  See text for details of
150 REM  the different demonstrations
160 REM
170 REM  Print title page "menu"
180 REM
190 ON ERROR GOTO360
200 MODE7
210 PROCCURSE
220 PROCINSTR
230 GOTO260
240 F=INKEY(10)
250 F=F-48
260 IF F<1 OR F>9 GOTO240
270 ON F GOTO 280,290,300,310,320,330,340,350,360
280 MODE 1:PROCCURSE:PROCDRAW(5):GOTO260
290 MODE 1:PROCCURSE:PROCDRAW(21):GOTO260
300 MODE 1:PROCCURSE:PROCFILL:GOTO260
310 MODE2:PROCCURSE:PROCRECTANGLES:GOTO260
320 MODE1:PROCCURSE:PROCELLIPSE(0):GOTO260
330 MODE1:PROCCURSE:PROCELLIPSE(1):GOTO260
340 MODE1:PROCCURSE:PROCWINDOWS:GOTO260
350 MODE2:PROCCURSE:PROCTRIANGLES:GOTO260
360 MODE7:PROCCURSE:PROCINSTR:GOTO260
370 REM
380 REM  Lines and dotted lines
390 REM
400 DEF PROCDRAW(Z%)
410 REPEAT
420   CLG
430   X%=RND(1279):Y%=RND(1023)
440   A%=RND(7)
450   B%=RND(7)
460   IFA%=B% GOTO0450
470   VDU19,0,A%,0,0,0
480   VDU19,1,B%,0,0,0:GCOL 0,1
490   FOR U%= 0TO 1279 STEP20
500     MOVE U%,0
510     PLOT Z%,X%,Y%
520     PLOT Z%,(1279-U%),1023
530     NEXT
540   FOR W%= 0 TO 1023 STEP20
550     MOVE 1279,W%
560     PLOT Z%,X%,Y%
570     PLOT Z%,0,(1023-W%)
580     NEXT
590   F=INKEY(0)
600   UNTIL F>1
610 F=F-48
620 ENDPROC
630 REM
640 REM  Triangles
650 REM
660 DEF PROCFILL
670 CLG
680 REPEAT
690   A%=RND(1279):B%=RND(1023)
700   C%=RND(1279):D%=RND(1023)
710   E%=RND(1279):F%=RND(1023)
720   MOVE A%,B%
730   MOVE C%,D%
740   PLOT 86,E%,F%
750   FOR Q%= 0 TO 1000:NEXT
760   F=INKEY(0)
770   UNTIL F>1
780 F=F-48
790 ENDPROC
800 REM
810 REM  Rectangles in 16 colour effects
820 REM
830 DEF PROCRECTANGLES
840 CLG
850 REPEAT
860   Z%=RND(2)
870   A%=RND(15)
880   R1%=RND(400)+20
890   R2%=RND(400)+20
900   X%=RND(1279-R1%)
910   Y%=RND(1023-R2%)
920   GCOLZ%,A%
930   VDU29,X%;Y%;
940   MOVE R1%,0
950   MOVE 0,0
960   PLOT85,R1%,R2%
970   PLOT85,0,R2%
980   FOR Q%= 0 TO 1000:NEXT
990   F=INKEY(0)
1000  UNTILF>1
1010 F=F-48
1020 ENDPROC
1030 REM
1040 REM  Ellipses - outline and solid
1050 REM
1060 DEF PROCELLIPSE(FILL%)
1070 IF FILL%=0 THEN Z%=5 ELSE Z%=85
1080 CLG
1090 REPEAT
1100   W%=RND(3)
1110   GCOL0,W%
1120   A%=RND(200):B%=RND(200)
1130   X%=RND(880)+200:Y%=RND(624)+200
1140   INC=RND(180)
1150   N%=100
1160   P=2*PI/(N%-1)
1170   C1=COS(INC):S1=SIN(INC)
1180   C2=COS(P):S2=SIN(P)
1190   C3=1:S3=0
1200   VDU29,X%;Y%;
1210   MOVE(A%*C1),(A%*S1)
1220   FOR M%=1TON%
1230     IF FILL%<>0 THEN MOVE 0,0
1240     X1=A%*C3:Y1=B%*S3
1250     PLOTZ%,(X1*C1-Y1*S1),(X1*S1+Y1*C1)
1260     T1=C3*C2-S3*S2
1270     S3=S3*C2+C3*S2
1280     C3=T1
1290     NEXT
1300   FOR Q=0TO 1000:NEXT
1310   F=INKEY(0)
1320   UNTILF>1
1330 F=F-48
1340 ENDPROC
1350 REM
1360 REM  Rapid pattern changes
1370 REM
1380 DEF PROCTRIANGLES
1390 CLG
1400 REPEAT
1410   GCOLRND(7),RND(7)
1420   X=RND(640):Y=RND(512)
1430   PLOT85,640+X,512-Y
1440   PLOT85,640-X,512+Y
1450   PLOT85,640+X,512+Y
1460   PLOT85,640-X,512-Y
1470   F=INKEY(0)
1480   UNTILF>1
1490 F=F-48
1500 ENDPROC
1510 REM
1520 REM  Instruction menu
1530 REM
1540 DEF PROCINSTR
1550 CLS
1560 PRINT''"This is a demonstration program which"
1570 PRINT"illustrates some of the graphics"
1580 PRINT"facilities of the BBC Microcomputer."
1590 PRINT''"Press one of the keys 0-9 to obtain"
1600 PRINT"a demonstration. To obtain another"
1610 PRINT"demonstration, press a different key."
1620 PRINT'''"1 - draws continuous lines"
1630 PRINT"2 - draws dotted lines"
1640 PRINT"3 - plots triangles in inverse"
1650 PRINT"4 - plots rectangles in 8 colours"
1660 PRINT"5 - draws ellipses"
1670 PRINT"6 - draws ellipses and fills them"
```

```
1680 PRINT"7 - text and graphics windows"
1690 PRINT"8 - random patterns at 160*256 pixels"
1700 PRINT"9 - displays INSTRUCTIONS again"
1710 F=GET-48
1720 ENDPROC
1730 REM
1740 REM  Remove cursor
1750 REM
1760 DEF PROCCURSE
1770 ?&FE00=10:?&FE01=32
1780 ENDPROC
1790 REM
1800 REM  Text and graphics windows
1810 REM
1820 DEF PROCWINDOWS
1830 VDU28,20,31,39,25
1840 COLOUR1:PRINT"THIS EXAMPLE LASTS"'"ABOUT 30 SECONDS

1850 VDU24,50;50;500;900;
1860 VDU28,20,20,39,5
1870 VDU19,0,2,0,0,0,19,2,4,0,0,0,19,3,3,0,0,0,19,1,6,0,
0,0
1880 COLOUR129
1890 COLOUR2
1900 GCOL0,130
1910 CLS:CLG
1920 M%=0:L%=0
1930 FOR N%=1TO300
1940   M%=M%+1:IF M%<10 THEN GOTO1990
1950   M%=0:PRINT;INT(N%/10);" Example of text"'"and gra
phics windows"'
1960   L%=L%+1:IF L%<5:GOTO1990
1970   GCOL0,128+RND(4):CLG
1980   L%=0
1990   GCOL0,RND(7)
2000   PLOT85,RND(500),RND(900)
2010   F=INKEY(0):IF F>1 F=F-48:N%=1000
2020   NEXT
2030 IF F<1 OR F>9 F=9
2040 ENDPROC
```

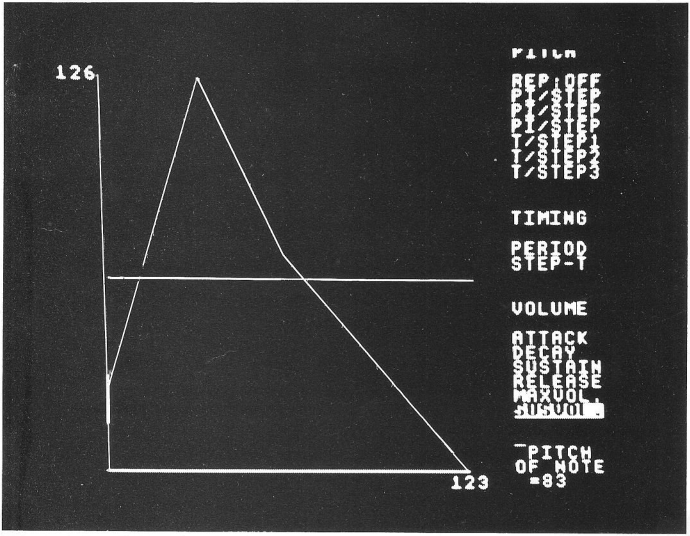
Listing 1. The graphics demonstration program.



S Draper

ENVELOPE DESIGN

Utilising the BBC Micro's SOUND capability is facilitated by this program for designing ENVELOPEs for complex sound effects.



The program given here is intended as an aid to designing ENVELOPEs for complex sound effects, or for the simulation of musical instruments. In use it presents a time graph of the envelope's amplitude and pitch, all parameters being alterable in real time, and the present envelope being played on demand. When the user is satisfied with one envelope he may either go on to design another (up to twenty at one time may be stored), or get a printout of the ENVELOPE and SOUND commands needed to use the envelope in a program.

USING THE PROGRAM

When the program is run an initial graph will appear on the screen. To the right of it is a menu, and the parameter to be altered may be chosen by moving a pointer on this menu. This is done by means of the cursor up and down keys, and the present choice will be shown in the menu in reverse graphics.

When the desired parameter has been chosen it can be altered by means of the 'I' and 'D' keys:

'I' increments the currently selected parameter, and 'D' decrements it. The other controls are as follows:

∧ . . . Raise pitch of sound produced.
Shift (~) . . . Lower pitch of sound produced.
T . . . Change type of sound produced (sound, pulse, noise).
R . . . Change pitch envelope from auto-repeat to non-repeating or *vice versa*.
N . . . go on to next envelope.

PROGRAM STRUCTURE

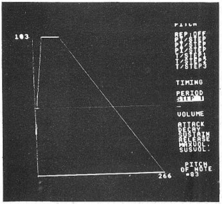
Statement	Function	Action
Lines 10-60	Set up	Sets up arrays for the various envelope parameters.
Lines 70-125	Constants	Reads constant values into their respective arrays.
Line 140	First	Sets the envelope no. to the first one.
Lines 150-200	Env set	Sets up constants for a new envelope.
Lines 210, 220	Screen	Clears to Mode 4, and defines text and graphics windows.
Lines 230, 240	Initialize	Puts initial values into the present envelope parameters array.
Lines 250, 260	Init display	Displays the initial envelope.
Line 270	Start loop	Beginning of the envelope defining loop.
Lines 280-365	Controls	Get input from keyboard.
Line 380	Display	Draw a new graph if some parameter has been changed.
Line 400	Loopend	Terminate inner loop if the control used was either N or Q.
Lines 410-465	Array place	Place the values finally decided on in the parameter arrays.
Line 470	Loopend	Terminate outer loop if control used was Q.
Lines 480-590	Printout	Output the ENVELOPE and SOUND commands needed for the envelopes which have been defined.
Lines 600-860	Display	Defines a procedure to draw auto scaled graphs of volume and pitch against time.
Lines 870-950	Rangecheck	Checks that no parameters have been altered to a value outside their legal range.
Lines 960-1050	Placemenu	Prints the menu out onto the text window.
Lines 1060-1160	Pointer move	Moves the pointer on the menu.
Lines 1170-1200	Playnote	Plays the presently defined sound.
Lines 1210, 1220	FNINC	Defines a function to increment a parameter.
Lines 1230, 1240	FNDEC	Defines a function to decrement a parameter.
Lines 1250-1320	Frequency	Prints out the frequency of the present sound.
Lines 1330-1360	Repeat	Alters the display after pitch envelope has been changed to or from auto repeat.
Lines 2000-2020	Type	Prints out type of sound.
Lines 5000-5030	Noise	Defines envelopes for noise pulse type sounds.

Q . . . Quit. The program will then print out the ENVELOPE and SOUND commands needed for all the envelopes previously defined.
P . . . Plays present sound once.

Since the use of long variable and PROCedure names in the program makes it fairly readable there is really not much to be said here. For those who have not used the BBC for long though it might be worthwhile pointing out what a few of the more unusual statements do.

TECHNICAL DETAILS

The only things that fall under this heading are, I think, the *FX4,1 statement on line 130 and the *FX15,1 on line 390. *FX4,1 simply allows the cursor keys to be accessed by the GET statement (cursor up returning the value 139, and cursor down giving 138). *FX15,1 clears the

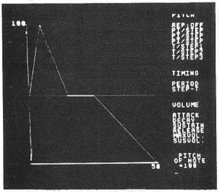


Adjusting the scaling of the time period.

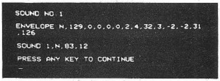
keyboard type ahead buffer — this is necessary to ensure that the controls only operate when actually being pressed and do not continue to do so for a while afterwards, as would otherwise be the case.

The only other thing is the use of the variable DIS as a logical variable on line 380. Here the statement IF DIS is short for IF DIS = TRUE — this has been used in several places.

Unfortunately the program is too long to fit into a model A although a stripped down version handling only one envelope at a time might just do so.



The final envelope design.



The parameter printout.

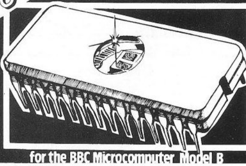
```
   10 @%=0
   20 DIM PITCHSTEP1(20),PITCHSTEP2(20),PITCHSTEP3(20)
   30 DIM PITCHTIME1(20),PITCHTIME2(20),PITCHTIME3(20)
   40 DIM PERIOD(20),STPTIME(20),RPT(20)
   50 DIM ATTACK(20),DECAY(20),SUSTAIN(20),RELEASE(20),FR
EQ(20),TYPE(20)
   60 DIM MAXVOL(20),TARGETVOL(20),A(13),MIN(13),MAX(13),
MESSAGE$(13),TYPE$(2)
   70 FORLOOP=0TO13
   80   READ MIN(LOOP),MAX(LOOP)
   90   NEXTLOOP
  100 FORMESSAGE=0TO13
  110   READ MESSAGE$(MESSAGE)
  120   NEXTMESSAGE
  125 TYPE$(0)="SOUND":TYPE$(1)="PULSE":TYPE$(2)="NOISE"
  130 *FX4,1
  140 N=-1
  150 REPEAT
  160   PITCHREPEAT=FALSE
  170   PARAMETER=0
  175   TYPE=0
  180   DIS=FALSE
  190   F=100
  200   N=N+1
  210   MODE4
  220   VDU24,0;0;1056;1011;28,33,31,39,0
  230   A(0)=0:A(1)=0:A(2)=0:A(3)=0:A(4)=0:A(5)=0:A(6)=5:
A(7)=1:A(8)=20:A(9)=-5
  240   A(10)=0:A(11)=-2:A(12)=100:A(13)=50
  250   PROCPLACEMENU
  260   PROCDISPLAY
  270   REPEAT
  280     CON=GET
  290     IFCON=139THENPARAMETER=(PARAMETER+13)MOD14:PROC
POINTER(1)
  300     IFCON=138THENPARAMETER=(PARAMETER+1)MOD14:PROCP
OINTER(13)
  310     IFCON=ASC("P")THENPROCPLAYNOTE
  320     IFCON=ASC("I")THENA(PARAMETER)=FNI(A(PARAMETER)
):DIS=TRUE
  330     IFCON=ASC("D")THENA(PARAMETER)=FND(A(PARAMETER)
):DIS=TRUE
  340     IFCON=ASC("^")THENF=FNI(F):PROCF
  350     IFCON=ASC("~")THENF=FND(F):PROCF
  360     IFCON=ASC("R") THEN PITCHREPEAT=PITCHREPEAT EOR
128:PROCREP
  365     IFCON=ASC("T")THENTYPE=(TYPE+1)MOD3:PROCTYPE
  370     PROCRANGECHECK
  380     IF DIS THEN PROCDISPLAY:DIS=FALSE
  390     *FX15,1
  400     UNTILCON=ASC("N") OR CON=ASC("Q")
  410   PITCHSTEP1(N)=A(0):PITCHSTEP2(N)=A(1):PITCHSTEP3(
N)=A(2)
>LIST420,
  420 PITCHTIME1(N)=A(3):PITCHTIME2(N)=A(4):PITCHTIME3(N)
=A(5)
  430 PERIOD(N)=A(6):STPTIME(N)=A(7)
  440 ATTACK(N)=A(8):DECAY(N)=A(9):SUSTAIN(N)=A(10):RELEA
SE(N)=A(11)
  450 MAXVOL(N)=A(12):TARGETVOL(N)=A(13):FREQ(N)=F
  460 RPT(N)=PITCHREPEAT EOR128
  465 TYPE(N)=TYPE
  470 UNTILCON=ASC("Q")
  480 FORENV=0 TO N
  490   MODE7
  500   PRINT'"SOUND NO.";ENV+1
  510   PRINT'"ENVELOPE N,";STPTIME(ENV)+RPT(ENV);",";PIT
CHSTEP1(ENV);",";
  520   PRINTPITCHSTEP2(ENV);",";PITCHSTEP3(ENV);",";PITC
HTIME1(ENV);",";
  530   PRINTPITCHTIME2(ENV);",";PITCHTIME3(ENV);
  540   IFTYPE(ENV)>0THENPRINT",0,0,0,0,0,0"ELSEPRINT",";
ATTACK(ENV);",";DECAY(ENV);",";SUSTAIN(ENV);",";RELEASE(E
NV);",";MAXVOL(ENV);",";TARGETVOL(ENV)
  545   IF TYPE>0 THENPRINT"ENVELOPE M,0,0,0,0,0,0,";ATTA
CK(ENV);",";DECAY(ENV);",";SUSTAIN(ENV);",";RELEASE(ENV);
",";MAXVOL(ENV);",";TARGETVOL(ENV)
  550   IF TYPE>0 THENPRINT'"SOUND&101,N,";FREQ(ENV);",";
PERIOD(ENV):PRINT"SOUND&100,M,";4*TYPE(ENV)-1;",";PERIOD(
ENV) ELSE PRINT'"SOUND 1,N,";FREQ(ENV);",";PERIOD(ENV)
  560   PRINT'"PRESS ANY KEY TO CONTINUE."
  570   DUMMY=GET
  580   NEXTENV
  590 END
  600 DEFPROCDISPLAY
  610 TTALTIME=A(6)*5-A(7)*A(13)/A(11)
  620 HUNDREDTHS=850/TTALTIME
  630 IFA(12)>A(13) THEN VOLSCALE=850/A(12) ELSE VOLSCALE
=850/A(13)
  640 CLG
  650 MOVE100,950:DRAW100,100:DRAW950,100
  660 VDU5:MOVE900,90:PRINTINT(TTALTIME)
  670 MOVE5,965:IFA(13)>A(12) THEN PRINTA(13) ELSE PRINTA
(12)
  680 VDU4
  690 MOVE100,100
  700 DRAW HUNDREDTHS*A(12)/A(8)+100,100+VOLSCALE*A(12)
  710 PLOT1,(A(13)-A(12))*HUNDREDTHS/A(9),(A(13)-A(12))*V
OLSCALE
  720 DIFFTIME=5*A(6)-(A(13)-A(12))/A(9)-A(12)/A(8)
  730 PLOT1,HUNDREDTHS*DIFFTIME,A(10)*VOLSCALE*DIFFTIME
  740 DRAW 950,100
  750 MOVE100,525
  760 X=0
  770 REPEAT
  780   PLOT17,HUNDREDTHS*A(3),A(0)*A(3)*4
  790   PLOT17,HUNDREDTHS*A(4),A(1)*A(4)*4
  800   PLOT17,HUNDREDTHS*A(5),A(2)*A(5)*4
  810   X=X+A(3)+A(4)+A(5)
  820   UNTILNOT PITCHREPEAT OR X>=TTALTIME
  830 IF PITCHREPEAT OR X>=TTALTIME THEN ENDPROC
  840 PLOT17,HUNDREDTHS*(TTALTIME-X),0
  850 ENDPROC
  860 ENDPROC
  870 DEFPROCRANGECHECK
  880 FORCHECKLOOP=0 TO 13
  890   IF A(CHECKLOOP)<MIN(CHECKLOOP) THEN A(CHECKLOOP)=
MIN(CHECKLOOP)
  900   IF A(CHECKLOOP)>MAX(CHECKLOOP) THEN A(CHECKLOOP)=
MAX(CHECKLOOP)
  910   NEXTCHECKLOOP
  920 IFA(9)<>0THEN940
  930 IFCON=ASC("I")THEN A(9)=1 ELSE A(9)=-1
  940 IF(A(13)>A(12) AND A(9)<0)OR(A(13)<A(12) AND A(9)>0
) THEN A(9)=-A(9)
  950 ENDPROC
  960 DEFPROCPLACEMENU
  970 CLS
  980 PRINT"PITCH"'
  990 PRINT"REP:OFFPI/STEPPI/STEPPI/STEPT/STEP1T/STEP2T/S
TEP3"
 1000 PRINT'"TIMING"'
 1010 PRINT"PERIOD STEP-T"'
 1020 PRINT'"VOLUME"'
 1030 PRINT"ATTACK DECAY  SUSTAINRELEASEMAXVOL.SUSVOL."
 1040 PRINT'" PITCH OF NOTE =";F
 1050 ENDPROC
 1060 DEFPROCPOINTER(CHANGE)
 1070 PREVIOUS=(PARAMETER+CHANGE)MOD14
 1080 DOWN=PREVIOUS+3
 1090 IFPREVIOUS>7 THEN DOWN=DOWN+8 ELSE IF PREVIOUS>5 DO
WN=DOWN+4
 1100 PRINTTAB(0,DOWN);MESSAGE$(PREVIOUS)
 1110 DOWN=PARAMETER+3
 1120 IFPARAMETER>7 THEN DOWN=DOWN+8 ELSE IF PARAMETER>5
DOWN=DOWN+4
 1130 COLOUR129:COLOUR0
 1140 PRINTTAB(0,DOWN);MESSAGE$(PARAMETER)
 1150 COLOUR1:COLOUR128
 1160 ENDPROC
 1170 DEFPROCPLAYNOTE
 1175 IFTYPE>0THEN5000
 1180 ENVELOPE1,A(7)+PITCHREPEAT EOR128,A(0),A(1),A(2),A(
3),A(4),A(5),A(8),A(9),A(10),A(11),A(12),A(13)
 1190 SOUND1,1,F,A(6)
 1200 ENDPROC
 1210 DEFFNI(X)
 1220 =X+INT(ABS(X/10)+1)
 1230 DEFFND(X)
 1240 =X-INT(ABS(X/10)+1)
 1250 DATA -127,127,-127,127,-127,127,0,255,0,255,0,255,1
,255,1,127,1,127
 1260 DATA -127,127,-127,0,-127,-1,0,126,0,126
 1270 DATA PI/STEP,PI/STEP,PI/STEP,T/STEP1,T/STEP2,T/STEP
3,PERIOD,STEP-T
 1280 DATA ATTACK,DECAY,SUSTAIN,RELEASE,MAXVOL.,SUSVOL.
 1290 DEFPROCF
 1300 F=(F+256)MOD256
 1310 PRINTTAB(0,29);" =";F;"  ";
 1320 ENDPROC
 1330 DEFPROCREP
 1340 PRINTTAB(0,2);"REP:";:IFPITCHREPEAT PRINT"ON " ELSE
PRINT"OFF"
 1350 PROCDISPLAY
 1360 ENDPROC
 2000 DEFPROCTYPE
 2010 PRINTTAB(0,31);TYPE$(TYPE);
 2020 ENDPROC
 5000 ENVELOPE1,A(7)+PITCHREPEAT EOR128,A(0),A(1),A(2),A(
3),A(4),A(5),0,0,0,0,0,0
 5010 ENVELOPE2,A(7)+PITCHREPEAT EOR128,0,0,0,0,0,0,A(8),
A(9),A(10),A(11),A(12),A(13)
 5020 SOUND&101,1,F,A(6):SOUND&100,2,TYPE*4-1,A(6)
 5030 ENDPROC
```

Listing 1. A program to design ENVELOPEs for complex sound effects.

Software for the BBC Micro

from COMPUTER CONCEPTS

Wordwise



for the BBC Microcomputer Model B

The word processor for the BBC machine. This ROM based word processor has received superb reviews.
Supplied with full spiral bound manual and cassette containing an example document and free typing tutor program.
Now available from stock. Quantity Discounts.
£39.00 + £1.50 p&p + VAT.

from COMPUTER CONCEPTS

Forthcoming ROMs

Beeb-calc

A ROM based spread sheet program.

Debugging Program

2 machine code debugging programs — one in ROM, one on tape. Essential for the machine code programmer. An ideal complement to the assembler built into the BBC machine.

Disk Doctor

A ROM containing useful disk utility programs. Enables the recovery of any data off the disk including deleted files etc.

Available Soon! SEND FOR DETAILS

from COMPUTER CONCEPTS

Games FOR THE MODEL B.

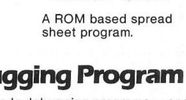
NEW!
SWARM
Alien against you, mode 2 graphics. Very fast. £7.80 + VAT.

Spacehawks

Mode 2 graphics, a machine code game much like the 'Galaxians' found in the arcades — very fast and also works with joysticks.

£7.80 +VAT

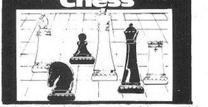
Asteroid belt



High speed and high resolution graphics are combined in this game to produce an exciting game — almost identical to the original arcade version.

£7.80 +VAT

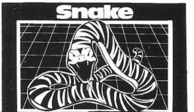
Chess



High resolution graphics with thousands of skill levels — more features than any other chess game.

£10.00 + VAT.

Snake



Another game that has received very favourable reviews - Fast and addictive.

£7.80 +VAT

We give very generous trade discounts

Cash or royalties — we pay the best rates around for any good BBC micro software.

Our tapes are guaranteed to work on all Operating Systems.

COMPUTER CONCEPTS

Dept PS1
16 Wayside, Chipperfield,
Herts, WD4 9JJ. tel (09277) 69727

Access — Buy it with Access

BARCLAYCARD VISA

Ian Nicholls

JOYSTICKS ON THE BBC

Software to help you use your joysticks on the BBC Micro

Many people have bought the joystick offered by Acorn as part of the BBC Microcomputer system, or an equivalent product, plugged them in, and then found that their favourite programs refuse to take any notice of them. To be able to use the joysticks you must include in your program the instructions which will read values from them. This instruction is the BASIC word ADVAL. In this article we will explore the use of ADVAL and see how to produce a drawing program which uses the joysticks, but not the keyboard at all.

HOW JOYSTICKS WORK

Let's begin by looking at how the joysticks actually work. Inside they are very simple consisting of just two potentiometers (variable resistors) and an on/off push button. If you move the joystick vertically, then only one potentiometer moves, and if you move it horizontally, then only the other moves. Moving in any other direction, both potentiometers move. Figure 1 shows this diagrammatically.

As the joystick is moved the sliders move inside the potentiometers, and the voltages on the sliders range from 0 to 1.8 volts. The BBC joysticks are arranged as in Fig. 1. Holding the joystick with the Fire button pointing away from you, the following 'horizontal' and 'vertical' voltages are obtained:

The voltages are fed to the analogue to digital converter (ADC) chip, the uPD7002, where they are converted into numbers between 0 and 4095 which the machine operating system then multiplies by 16 to give numbers between 0 and 65520, as shown in the column labelled 'ADC reading'. These readings thus change in units of 16. The chip used has a resolution of 12 bits giving numbers between 0 and 4095. The software can cope with future chips which have resolutions up to 16 bits; in this case the readings would change in steps of 1 between 0 and 65535.

Joystick Position	ADC reading
Extreme left — horizontal voltage, 1.8 volts	65520
Extreme right — horizontal voltage, 0 volts	0
Top — vertical voltage, 1.8 volts	65520
Bottom — vertical voltage, 0 volts	0

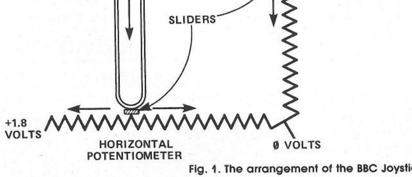


Fig. 1. The arrangement of the BBC Joysticks.

DRAWING FIRE!

We have progressed far enough to write a short program to draw on the screen. If we were to use the values produced by the ADC chip directly then any points we plotted would rapidly disappear off the screen, since we would be trying to plot values from 0 to 65520, but we can only plot values up to 1279 horizontally and 1023 vertically, so we must scale the values accordingly. We must multiply the vertical, Y, values by 1023/65520 and the horizontal, X, values by 1279/65520. Also, we want the value X=0, Y=0 to be in the left hand corner of the screen so we will have to modify the X value by subtracting it from 65520.

The ADVAL function usually takes the five positive values ADVAL(0) to ADVAL(4); it can also take the values ADVAL(− 1) to ADVAL(− 9), their use will be explained later. The four positive values 1 to 4 are the channel numbers.

ADVAL(1)	reads the left hand joystick's horizontal value
ADVAL(2)	reads the left hand joystick's vertical value
ADVAL(3)	reads the right hand joystick's horizontal value
ADVAL(4)	reads the right hand joystick's vertical value

ADVAL(0) performs the special function of reading which of the Fire buttons has been pressed. If we use the statement:

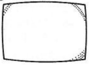
```
fire%=ADVAL (Ø) AND 3
```

in a program then:
fire% = 0 means no button is being pressed
fire% = 1 means the left-hand button is being pressed
fire% = 2 means the right-hand button is being pressed
fire% = 3 means both buttons are being pressed

We can now try Listing 1 which will run on a Model A or B. If you run this program, nothing will happen until you press one of the Fire buttons on the joystick (***not*** both at the same time — see line 180). A box round the edge of the display will appear and the message 'Joystick No. Z', where Z will be 1 or 2. If Z is 1 then the joystick you are holding is the left-hand one, if Z is 2 then it is the right-hand one. It is worth putting sticky labels on the joysticks to remind you of this if yours, like mine, are not marked in any way.

SCALING UP

When you move the joystick a line of dots will follow its path round the screen. If you now move the joystick keeping it as far from the centre position as possible, you would probably expect it to follow the edges of the box. However, at the corners of the box you will find something like this:



What is happening is that the joystick cannot simultaneously reach the limits of travel of both potentiometers at the same time, which would be necessary to plot right into all four corners. You will find that it possibly can reach one or two corners, but not all four. This will be important to us in applications where we *do* want to reach every part of the screen. To overcome this problem we will have to use scaling factors different from those given above. Listing 2 gives us a means of finding out what these new scaling factors should be. They will differ for each joystick.

When you run Listing 2, you will see the following type of display:

CHANNEL	VALUE	MINIMUM	MAXIMUM
ADVAL(1)	....	32	65392
ADVAL(2)	....	32	65504
ADVAL(3)	....	48	65520
ADVAL(4)	....	48	65520

THIS IS FIRE BUTTON No. 1
ADVAL(1) = ADVAL(2) AT — min 352 — max 59568
ADVAL(3) = ADVAL(4) AT — min 1056 — max 62640

As you move each of the joysticks the numbers for ADVAL(1) and ADVAL(2) or ADVAL(3) and ADVAL(4), under the column headed VALUE will change. The numbers under the columns headed MINIMUM and MAXIMUM will keep track of the minimum and maximum values which can be obtained on each channel. Ideally these should be 0 and 65520. The above numbers, obtained with an actual BBC joystick show that the ideal is not usually possible.

If you want to make sure that the computer actually reads the value 0, say, when the left-hand joystick is at the bottom of its travel and 1023 when it is at the top, then ADVAL(2) needs to be scaled as follows:

```
Y=(ADVAL(2)-MINIMUM)*1Ø23/(MAXIMUM-
  MINIMUM)
```

In the example above, Y = (ADVAL(2) — 32) * 1023/65472.

If you press the Fire button on each of the joysticks, the message in the middle of the screen will tell you which number the computer assigns to the buttons. You will find, hopefully, that Fire button 1 is on the joystick which controls channels 1 and 2: this is the left-hand joystick.

The bottom two lines of the display produced by Listing 2 show the lowest and highest values where ADVAL(1) and ADVAL(2) are the same, and where ADVAL(3) and ADVAL(4) are the same. In other words, these are the lowest and highest values which can be obtained simultaneously in the vertical and horizontal directions for each joystick. If we use these values as the scaling factors, then we will be able to reach any point in the desired plotting area.

A bit of dexterity is needed to find these values: move the joystick slowly around in a circle keeping it as far away from the centre point as possible. Watch the values on channels 1 and 2 (or 3 and 4) until they become nearly equal, either at a low value or at a high value. Now move the joystick even more slowly, in very small steps, trying to bring the two numbers as close together as possible. Suddenly (we hope!) the number against 'min' or 'max' will change from 0 to a larger number. Repeat for the other position on the joystick, and then for the other joystick.

THE MAIN THEME

We now have both joysticks calibrated and can turn to Listing 3, which is quite a sophisticated drawing program, offering various 'menu' choices to the user which are selected using the right-hand joystick and its Fire button. The left-hand joystick and Fire button are used to draw on the screen. The program requires a Model B or 32K Model A. The various menu choices are:

Procedure PROCcolour

1) colour to be used when drawing (one of four possible colours)
2) draw using triangles
3) draw using lines
4) draw using dotted lines
5) draw using points
6) move to menu which allows changes of colour and of drawing action (Procedure PROCchng-col)
7) Clear the screen

Procedure PROCchng-col

1) Change the mode of drawing action from direct over-plotting to the use of Boolean options (Procedure PROCchng-act)
2) Change the actual set of four colours available (Procedure PROCalter-col)

Procedure PROCchng-act

1) Direct over-plotting (normal drawing mode)
2) Draw using AND
3) Draw using OR
4) Draw using EXOR
5) Draw by inverting what is already on the screen.

Procedure PROCalter-col

1) Change logical colour A(A = 0 to 3) — YES/NO?
2) If answer to 1 is YES, choose one of the 16 possible colours to be the actual colour displayed when the corresponding logical colour (0 to 3) is used.

The procedure PROCcalibrate is used to read in values obtained from the calibration program, Listing 2, and then to calculate the appropriate scaling factors to be used.

Readers will need to substitute their own values in the DATA statement. Line 870, the READ statement, reads in four variables onemin, onemax, two and three. Readers should substitute the following values from using their joysticks with Listing 2 in the DATA statement line.

onemin	—	minimum value where ADVAL(1) = ADVAL(2) for left joystick
onemax	—	maximum value where ADVAL(1) = ADVAL(2) for left joystick
two	—	minimum value obtained on channel, ADVAL(3)
three	—	maximum value obtained on channel, ADVAL(3)

HOW IT RUNS

The main part of the program is contained in line 570 to 620

```
570  REPEAT
580  PROCread(39,0)
590  IF fire%=1 OR fire%=3 draw=1
     ELSE draw=0:PROCflash
600  PROCcolour
610  PLOT plot%+draw,X,Y
620  UNTIL FALSE
```

The important elements of these six lines are as follows:

PROCread (scale, T%) This procedure reads the left-hand joystick position and calculates the x and y co-ordinates, appropriately scaled (x,y). It then checks the two Fire buttons. It also reads the horizontal position of the right-hand joystick and uses the parameters 'scale' and T% as follows. Firstly, it scales the horizontal value such that at the extreme left of the joystick's travel, it returns a value of T%, and at the opposite end of its travel, it returns a value equal to T% + scale. It then plots the '∧' symbol at the horizontal position, T% + scale, within a one line text window at the bottom of the screen. This text window is produced by line 720:

```
720  VDU 28,0,31,39,30
```

PROCflash In order to be able to draw successfully, most artists need to be able to see the pencil and paper. PROCflash draws a flashing cursor, which shows the point of our coloured pencil. We need to be able to move the cursor without drawing on the screen. To do this the program only draws when Fire button 1 is pressed (in which case the variable draw takes the value 1, otherwise it is 0 — see beginning of line 590). To prevent the cursor itself leaving a mark as it is moved, it is plotted twice using the exclusive-or function, EXOR, in the GCOL parameter. The cursor itself is a small square, made up from two triangles centred at the current x and y co-ordinates.

PROCcolour One of the facilities in this procedure is to choose whether to plot with points, lines, etc: the appropriate value is assigned to plot% as follows:

triangles	— plot%	= 84
lines	— plot%	= 4
dotted lines	— plot%	= 28
points	— plot%	= 68

PLOT plot% + draw, x,y This is the line which actually plots on the screen if draw = 1. The value of plot% will have been determined by PROCcolour in the previous statement.

To use the program, then, use the right-hand joystick by moving it left or right to select the colour, choice of lines, triangles, etc, which you wish to use, and press its Fire button to make your choice. Then use the left-hand joystick to move the flashing cursor about the screen; when you are ready to draw, press the left-hand Fire button. Happy drawing!

A FINAL NOTE

Putting a negative number in the ADVAL statement, such as Y=ADVAL(−4) enables you to see how full any of the internal buffers are. Putting in negative numbers from −1 to −9 returns the number of free spaces in the following buffers:

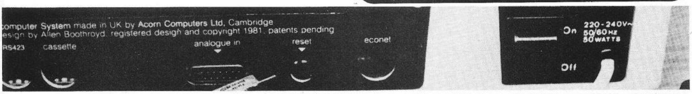
Y=ADVAL(−)	— keyboard buffer
Y=ADVAL(−2)	— RS423 input buffer
Y=ADVAL(−3)	— RS423 output buffer
Y=ADVAL(−4)	— printer output buffer
Y=ADVAL(−5)	— SOUND 0 buffer
Y=ADVAL(−6)	— SOUND 1 buffer
Y=ADVAL(−7)	— SOUND 2 buffer
Y=ADVAL(−8)	— SOUND 3 buffer
Y=ADVAL(−9)	— SPEECH buffer

```
>L.
  100 REM  Joystick Experimentation
  110 REM             by
  120 REM  I G Nicholls Dec 1982
  130 REM
  140 REM  Wait until a fire button is pressed
  150 REPEAT
  160   fire%=ADVAL(0) AND 3
  170   button%=(fire%-1)*2
  180   UNTIL(fire%=1 OR fire%=2)
  190 MODE 4
  200 REM
  210 REM  Repeat forever
  220 REM
  230 REPEAT
  240   REM
  250   REM  draw box round edge of screen
  260   REM
  270   MOVE 0,0
  280   PLOT5,1279,0
  290   PLOT5,1279,1023
  300   PLOT5,0,1023
  310   PLOT5,0,0
  320   REM
  330   REM  print Joystick no.
  340   REM  1-left, 2-right
  350   REM
  360   PRINTTAB(13,13);"Joystick No.";fire%
  370   REM
  380   button%=(fire%-1)*2
  390   REM
  400   REM  repeat until fire button pressed
  410   REM
  420   REPEAT
  430     REM
  440     REM  calculate x and y values
  450     REM
  460     X=1279-ADVAL(1+button%)*1279/65520
  470     Y=ADVAL(2+button%)*1023/65520
  480     REM
  490     REM  plot a point at X,Y
  500     REM
  510     PLOT69,X,Y
  520     REM
  530     REM  see whether button pressed
  540     REM
  550     fire%=ADVAL(0) AND 3
  560     UNTIL(fire%=1 OR fire%=2)
  570   REM
  580   REM  if button pressed, clear screen
  590   REM  and return to beginning
  600   REM
  610   CLG
  620   UNTIL FALSE
```

Listing 1. Joystick experimentation.

```
>L.
  100 REM  Joystick calibration
  110 REM           by
  120 REM  I G Nicholls Dec 1982
  130 REM
  140 ON ERROR GOTO610
  150 MODE7
  160 DIM A(4),MIN(4),MAX(4)
  170 FOR I%=1TO4
  180    MIN(I%)=32000
  190    MAX(I%)=32000
  200    NEXT
  210 onemin=0:onemax=0:twomin=0:twomax=0
  220 REM
  230 REM  Remove cursor
  240 REM
  250 VDU23,0,11,0,0,0,0,0,0,0
  260 PRINTTAB(0,1);"CHANNEL";TAB(15);"VALUE";TAB(23);"MI
NIMUM";TAB(33);"MAXIMUM"
  270 Q$=STRING$(39," ")
  280 REM
  290 REM  Start of main program loop
  300 REM
  310 REPEAT
  320    PROCread
  330    FOR I%=1TO4
  340       PRINTTAB(0,1+2*I%);"ADVAL(";I%;")";TAB(15);A(I%
);TAB(25);MIN(I%);TAB(35);MAX(I%)
  350       NEXT
  360    IF fire%=0 OR fire%=3 PRINTTAB(0,12);Q$;:GOTO380
  370    PRINTTAB(0,12);"THIS IS FIRE BUTTON NO.";fire%
  380    PRINTTAB(0,14);"ADVAL(1) = ADVAL(2) AT - min ";on
emin;"   "
  390    PRINTTAB(23,15);"- max ";onemax;"    "
  400    PRINTTAB(0,16);"ADVAL(3) = ADVAL(4) AT - min ";tw
omin;"   "
  410    PRINTTAB(23,17);"- max ";twomax;"    "
  420    UNTIL FALSE
  430 REM
  440 REM  End of main program loop
  450 REM
  460 DEF PROCread
  470 fire%=ADVAL(0) AND 3
  480 FOR I%=1TO4
  490    A(I%)=ADVAL(I%)
  500    IF A(I%)<MIN(I%) MIN(I%)=A(I%)
  510    IF A(I%)>MAX(I%) MAX(I%)=A(I%)
  520    NEXT
  530 IF A(1)=A(2) AND A(1)<8000 onemin=A(1)
  540 IF A(1)=A(2) AND A(1)>54000 onemax=A(1)
  550 IF A(3)=A(4) AND A(3)<8000 twomin=A(3)
  560 IF A(3)=A(4) AND A(3)>54000 twomax=A(3)
  570 ENDPROC
  580 REM
  590 REM  Restore cursor
  600 REM
  610 VDU23,0,11,255,0,0,0,0,0,0
  620 CLS
  630 END
```

Listing 2. Joystick calibration.




```
>>
>L.
  100 REM  Drawing Program Using Joysticks
  110 REM             by
  120 REM  I G Nicholls December 1982
  130 REM
  140 REM  Press "ESCAPE" to exit
  150 REM
  160 REM  Remove interlace to give steady picture
  170 REM
  180 *TV0,1
  190 ON ERROR GOTO2340
  200 MODE1
  210 REM
  220 REM  Initialisation
  230 REM
  240 DIM COL$(15),ACT_COL%(3)
  250 ACT_COL%(0)=0:ACT_COL%(1)=1
  260 ACT_COL%(2)=3:ACT_COL%(3)=7
  270 RESTORE 310
  280 FOR I%=0TO15
  290    READ COL$(I%)
  300    NEXT
  310 DATAblack,red,green,yellow,blue,magenta,cyan,white,
bla/white,red/cyan,green/mag,yell/blue,blue/yell,mag/gree
n,cyan/red,white/bla
  320 REM
  330 REM  Read in Joystick calibration parameters
  340 REM  from calibration program
  350 REM
  360 PROCcalibrate
  370 xx%=3:Z%=0:plot%=68:col%=3
  380 REM
  390 REM  Remove cursor
  400 REM
  410 VDU23,0,11,0,0,0,0,0,0,0
  420 VDU23,224,255,255,255,255,255,255,255,255
  430 BLOCK$=STRING$(4,CHR$224)
  440 Q$=STRING$(40," ")
  450 REM
  460 REM  Set up text window at bottom of screen
  470 REM
  480 PROCwindow
  490 REM
  500 REM  Create graphics window
  510 REM
  520 VDU24,0;132;1279;1023;
  530 Q=INKEY(50)
```

```
 540 REM
 550 REM  Main program loop
 560 REM
 570 REPEAT
 580   PROCread(39,0)
 590   IF fire%=1 OR fire%=3 draw=1 ELSE draw=0:PROCflas

 600   PROCcolour
 610   PLOT plot%+draw,x,y
 620   UNTIL FALSE
 630 REM
 640 REM  End of main program loop
 650 REM
 660 REM  Routine to put pointer into text
 670 REM  window (pointer -↑- moved with
 680 REM  right hand joystick); also reads
 690 REM  left hand joystick values
 700 REM
 710 DEF PROCread(scale,T%)
 720 VDU28,0,31,39,30
 730 x=a-ADVAL(1)*b
 740 y=ADVAL(2)*c-d
 750 fire%=ADVAL(0) AND 3
 760 xx=(e-ADVAL(3)*f)*scale/1023
 770 tab%=INT(xx):SCALE%=INT(scale+0.1)
 780 IF tab%<0 THEN tab%=0 ELSE IF tab%>SCALE% THEN tab%
=SCALE%
 790 PRINTTAB(0,0);SPC(T%+tab%);"↑";SPC(39-T%-tab%);
 800 ENDPROC
 810 REM
 820 REM  Calculate scaling parameters
 830 REM  for joystick values
 840 REM
 850 DEF PROCcalibrate
 860 RESTORE 980
 870 READ onemin,onemax,two,three
 880 b=onemax-onemin
 890 a=1279*onemax/b
 900 d=1023*onemin/b
 910 c=1023/b
 920 b=1279/b
 930 f=three-two
 940 e=1279*three/f
 950 h=1023*two/f
 960 g=1023/f
 970 f=1279/f
 980 DATA 352,59568,48,65520
 990 ENDPROC
1000 REM
1010 REM  Draw flashing cursor
1020 REM
1030 DEF PROCflash
1040 FOR J%=0TO1
1050   GCOL3,3
1060   MOVE x-8,y-8
1070   MOVE x+8,y-8
1080   PLOT 85,x-8,y+8
1090   PLOT 85,x+8,y+8
1100   GCOLZ%,col%
1110   NEXT
1120 ENDPROC
1130 REM
1140 REM  First menu options
1150 REM
1160 DEF PROCcolour
1170 xx%=tab% DIV 4
1180 IF fire%=0 OR fire%=1 ENDPROC
1190 IF xx%<4 col%=xx%:GCOL Z%,col%:ENDPROC
1200 IF xx%=4 plot%=84:ENDPROC
1210 IF xx%=5 plot%=4:ENDPROC
1220 IF xx%=6 plot%=28:ENDPROC
1230 IF xx%=7 plot%=68:ENDPROC
1240 IF xx%=8 PROCchng_col:ENDPROC
1250 IF xx%=9 CLG:ENDPROC
1260 ENDPROC
1270 REM
1280 REM  Two line text window at bottom
1290 REM  of screen
1300 REM
1310 DEF PROCwindow
1320 VDU28,0,31,39,29:COLOUR128:CLS
1330 COLOUR0
1340 PRINTTAB(0,0);BLOCK$
1350 COLOUR1
1360 PRINTTAB(4,0);BLOCK$
1370 COLOUR2
1380 PRINTTAB(8,0);BLOCK$
1390 COLOUR3
1400 PRINTTAB(12,0);BLOCK$;
1410 COLOUR0:COLOUR130
1420 PRINTTAB(16,0);"TRI ";
1430 COLOUR1:COLOUR131
1440 PRINTTAB(20,0);"LIN ";
1450 COLOUR2:COLOUR132
1460 PRINTTAB(24,0);"DLN ";
1470 COLOUR3:COLOUR129
1480 PRINTTAB(28,0);"PNT ";
1490 COLOUR0:COLOUR130
1500 PRINTTAB(32,0);"COL ";
1510 COLOUR1:COLOUR131
1520 PRINTTAB(36,0);"CLS ";
1530 COLOUR128:COLOUR3
1540 ENDPROC
1550 REM
1560 REM  Second menu options
1570 REM
1580 DEF PROCchng_col
1590 VDU28,0,31,39,29
1600 COLOUR 3:COLOUR 128:CLS
1610 PRINTTAB(0,0);"CHANGE COLOUR ACTION?";
1620 COLOUR 0:COLOUR 131
1630 PRINTTAB(22,0);"   YES  ";
1640 COLOUR 130
1650 PRINTTAB(30,0);"   NO   ";
1660 COLOUR 3:COLOUR 128
1670 Q=INKEY(100)
1680 REPEAT
1690   PROCread(15,22)
1700   UNTIL (fire%=2)
1710 IF tab%<8 PROCchng_act
1720 PROCalter_col
1730 ENDPROC
1740 REM
1750 REM  Third menu options
1760 REM
1770 DEF PROCchng_act
1780 VDU28,0,31,39,29
1790 COLOUR 3:COLOUR 128:CLS
1800 PRINTTAB(0,0);"COLOUR ACTION ";
1810 COLOUR 0:COLOUR 129
1820 PRINTTAB(14,0);"NORM ";
1830 COLOUR 130
1840 PRINTTAB(19,0);" AND ";
1850 COLOUR 131
1860 PRINTTAB(24,0);" OR  ";
1870 COLOUR 129
1880 PRINTTAB(29,0);"EXOR ";
1890 COLOUR 130
1900 PRINTTAB(34,0);" INV ";
1910 COLOUR 3:COLOUR 128
1920 Q=INKEY(100)
1930 REPEAT
1940   PROCread(24,14)
1950   UNTIL (fire%=2)
1960 Z%=tab% DIV 5
1970 GCOL Z%,col%:PROCwindow
1980 ENDPROC
1990 REM
2000 REM  Fourth menu options
2010 REM
2020 DEF PROCalter_col
2030 VDU28,0,31,39,29
2040 COLOUR 3:COLOUR 128:CLS
2050 FOR I%=0TO3
2060   L%=LEN(COL$(ACT_COL%(I%)))
2070   PRINTTAB(0,0)"CHANGE COLOUR ";I%;" - ";COL$(ACT.C
OL%(I%));SPC(20-L%);
2080   COLOUR 0:COLOUR 129
2090   PRINTTAB(30,0);"YES ";
2100   COLOUR 130
2110   PRINTTAB(34,0);" NO ";
2120   COLOUR 3:COLOUR 128
2130   Q=INKEY(100)
2140   REPEAT
2150     PROCread(7,30)
2160     UNTIL (fire%=2)
2170   VDU28,0,31,39,29
2180   IF tab%>3 GOTO2300
2190   PRINTTAB(0,1);Q$;
2200   Q=INKEY(50)
2210   REPEAT
2220     xx=(e-ADVAL(3)*f)*15/1023
2230     tab%=INT(xx)
2240     IF tab%<0 THEN tab%=0 ELSE IF tab%>15 THEN tab%
=15
2250     L%=LEN(COL$(tab%))
2260     fire%=ADVAL(0) AND 3
2270     PRINTTAB(18,0);COL$(tab%);SPC(20-L%);
2280     UNTIL (fire%=2)
2290   ACT_COL%(I%)=tab%:VDU19,I%,tab%,0,0,0
2300   NEXT
2310 PROCwindow
2320 ENDPROC
2330 REM  Restore normal cursor
2340 VDU23,0,11,255,0,0,0,0,0,0
2350 VDU26:CLS
2360 REM  Print "error" message
2370 REPORT
2380 PRINT" at line ";ERL
2390 END
```

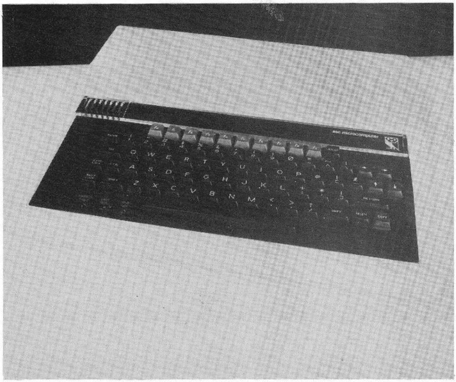
Listing 3. Drawing program using joysticks.

S Draper.

DISASSEMBLER #2

To take a machine code program from memory and print it out in assembly language on the screen, just read this article.

The program given here will take a machine code program from anywhere in memory and print it out on the screen in assembly language. I found it particularly useful for understanding how some of my old machine code programs worked when I wanted to modify them (I know — we should all keep assembly language copies, but who does). It is also very useful for figuring out how other peoples' programs do some of the clever things they do!



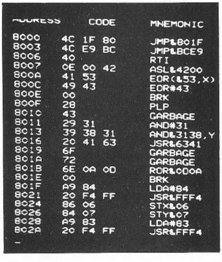
TECHNICAL DETAILS

There is nothing particularly notable about the program except the way in which the start address is input (line 50). The use of EVAL here allows the address to be typed in in hexadecimal notation which is much easier for the user to relate to than a long decimal number.

The table of variables used (below) should be of considerable assistance to those who wish to convert the program to run on other machines. This should not prove too difficult (on a 6502 machine of course), but note that the ? operator (eg line 150) is BBC for PEEK or POKE depending on context. Thus:

```
ADD? I=VALUE  means POKE (ADD+I, VALUE
?ADD=VALUE    means POKE ADD, VALUE; and
VALUE=ADD?I   means VALUE=PEEK (ADD+1)
```

Note also when converting to other systems the use of instruction *FX15, 1 in line 260. This simply clears the keyboard type ahead buffer. This could be replaced on the PET for instance by the lines:



ADDRESS	CODE	MNEMONIC
8000	4C 1F 80	JMP&801F
8003	4C E9 BC	JMP&BCE9
8006	40	RTI
8007	0E 00 42	ASL&4200
800A	41 53	EOR(&53,X)
800C	49 43	EOR#43
800E	00	BRK
800F	28	PLP
8010	43	GARBAGE
8011	29 31	AND#31
8013	39 38 31	AND&3138,Y
8016	20 41 63	JSR&6341
8019	6F	GARBAGE
801A	72	GARBAGE
801B	6E 0A 0D	ROR&0D0A
801E	00	BRK
801F	A9 84	LDA#84
8021	20 F4 FF	JSR&FFF4
8024	86 06	STX&06
8026	84 07	STY&07
8028	A9 83	LDA#83
802A	20 F4 FF	JSR&FFF4

TABLE OF VARIABLES

NAME	USE
MNEM$(X)	Mnemonics for the 56 different 6502 instructions.
PRE$(X)	Prefixes for the various addressing modes.
POST$(X)	Postfixes for the various addressing modes.
OPS(X)	No. of the instruction with code X in the above arrays.
OPM(X)	No. of the addressing mode of the instruction with code X.
N(X)	No. of bytes taken up by instructions with add. mode X.
ADD	Address presently being disassembled.
MO	No. of instruction presently being disassembled.

Table 1. A table of the variables used.

260 GET A$
265 IF A$<>""THEN 260
The purpose of lines 250, 260 is simply to wait for a keypress at the end of printing a full screen before going on to the next.

USING THE PROGRAM

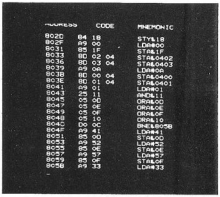
This should present no problems to anyone with any previous experience of machine code as it is fairly self-explanatory in use — simply enter the desired start address for disassembly in Hex and sit back. The machine code will be printed out in the following format:

ADDRESS MACHINE CODE ASSEMBLY LANGUAGE

Note that with jumps and branches it is the target address that is printed out in the assembly language section, even with relative branches.

If the disassembler encounters a byte with no corresponding instruction 'GARBAGE' will be printed out as its assembly language equivalent.

Disassembler should run equally well on a Model A or B.



PROGRAM STRUCTURE

Statement	Function	Action
Lines 10-70	Set up	Reads data into the arrays.
Line 80	Start of main loop	Each loop prints out one complete screen of assembly language.
Line 90	Headings	Clears the screen and prints column headings out.
Line 100	Start of secondary loop	Each loop prints out one instruction.
Lines 110 -240	Secondary loop	Prints current instruction and increments address pointer (ADD)
Lines 250 -270	Key wait	Waits for a keypress before doing next page of program.
Line 280	Target add.	Prints out target address for relative branches in Hex.
Lines 300 -520	Data	All the needed data.

Table 2. The program structure.

```
>LIST
  232  DIM MNEM$(56),PRE$(13),POST$(13),OPM(256),OPS(256)
,N(13):FORI=1TO13:READN(I):NEXT
 1005 REPEATREADA,B,C:OPS(A)=B:OPM(A)=C:UNTILA=255
 1010 FORI=1TO13:READPRE$(I),POST$(I):NEXT
 1020 FORI=0TO56:READMNEM$(I):NEXT
 1030 INPUT"START ADDRESS",ADD$:ADD=EVALADD$
 1035 N(0)=1
 1037 @%=&A00
 1040 REPEAT
 1045   CLS:PRINT"ADDRESS    CODE     MNEMONIC"'
 1050   FORI=2TO23
 1060     OPERAND=?ADD
 1062     MO=OPM(OPERAND)
 1065     PRINT~ADD;"   ";
 1070     FORJ=1TON(MO):IFADD?(J-1)<16THENPRINT"0";
 1075       PRINT~ADD?(J-1);" ";:NEXT
 1080     PRINTTAB(20,I);MNEM$(OPS(OPERAND));PRE$(MO);
 1085     IFMO=11THEN2000
 1090     IFN(MO)=1THEN1110
 1095     IFN(MO)<2THEN1110
 1100     FORJ=ADD+N(MO)-1TO ADD+1 STEP-1:IF?J<16THENPRIN
T"0";
 1105       PRINT~?J;:NEXT
 1110     PRINTPOST$(MO)
 1120     ADD=ADD+N(MO)
 1130     NEXT
 1135   A=GET
 1136   *FX15,1
 1140   UNTIL0
 2000 IFADD?1<128THENPRINT~(ADD+2+ADD?1) ELSEPRINT~(ADD-2
56+2+ADD?1)
 2010 GOTO1120
 4000 DATA1,1,3,2,2,3,3,2,2,2,2,3,2
 5000 DATA &6D,1,3,&65,1,4,&69,1,5,&7D,1,6,&79,1,7,&61,1,
8,&71,1,9,&75,1,10
 5010  DATA &2D,2,3,&25,2,4,&29,2,5,&3D,2,6,&39,2,7,&21,2
,8,&31,2,9,&35,2,10
 5020 DATA 10,3,2,14,3,3,6,3,4,&1E,3,6,22,3,10,&90,4,11,&
B0,5,11,&F0,6,11,&2C,7,3,&24,7,4
 5030 DATA 48,8,11,&D0,9,11,16,10,11,0,11,1,&50,12,11,&70
,13,11,&18,14,1,&D8,15,1
 5040 DATA &58,16,1,&B8,17,1,&CD,18,3,&C5,18,4,&C9,18,5,&
DD,18,6,&D9,18,7,&C1,18,8,&D1,18,9,&D5,18,10
 5050 DATA &EC,19,3,&E4,19,4,&E0,19,5,&CC,20,3,&C4,20,4,&
C0,20,5,&CE,21,3,&C6,21,4,&DE,21,6,&D6,21,10,&CA,22,1,&88
,23,1
 5060 DATA &4D,24,3,&45,24,4,&49,24,5,&5D,24,6,&59,24,7,6
5,24,8,&51,24,9,&55,24,10
 5070 DATA &EE,25,3,&E6,25,4,&FE,25,6,&F6,25,10
 5080 DATA &E8,26,1,&C8,27,1,&4C,28,3,&6C,28,12,32,29,3,&
AD,30,3,&A5,30,4,&A9,30,5,&BD,30,6,&B9,30,7,&A1,30,8,&B1,
30,9,&B5,30,10
 5090 DATA &AE,31,3,&A6,31,4,&A2,31,5,&BE,31,7,&B6,31,13,
&AC,32,3,&A4,32,4,160,32,5,&BC,32,6,&B4,32,10
 5100 DATA &4A,33,2,&4E,33,3,&46,33,4,&5E,33,6,&56,33,10
 5110 DATA &EA,34,1,13,35,3,5,35,4,9,35,5,&1D,35,6,&19,35
,7,1,35,8,17,35,9,21,35,10
 5120 DATA &48,36,1,8,37,1,&68,38,1,&28,39,1,&2A,40,2,&2E
,40,3,&26,40,4,&3E,40,6,&36,40,10
 5130 DATA &6A,41,2,&6E,41,3,&66,41,4,&7E,41,6,&76,41,10
 5140 DATA 64,42,1,96,43,1,&ED,44,3,&E5,44,4,&E9,44,5,&FD
,44,6,&F9,44,7,&E1,44,8,&F1,44,9,&F5,44,10
 5150 DATA &38,45,1,&F8,46,1,&78,47,1,&8D,48,3,&85,48,4,&
9D,48,6,&99,48,7,129,48,8,&91,48,9,&95,48,10
 5160 DATA &8E,49,3,&86,49,4,&96,49,13,&8C,50,3,&84,50,4,
&94,50,10
 5170 DATA &AA,51,1,&A8,52,1,&BA,53,1,&8A,54,1,&9A,55,1,&
98,56,1,255,0,0
10000 DATA"","",A,"",&,"",&,"",#,"",&,",X",&,",Y",(&,",X)
",(&,"),Y",&,",X",&,"",(&,),&,""
10010  DATA GARBAGE,ADC,AND,ASL,BCC,BCS,BEQ,BIT,BMI,BNE,B
PL,BRK,BVC,BVS,CLC,CLD,CLI,CLV,CMP
10020 DATA CPX,CPY,DEC,DEX,DEY,EOR,INC,INX,INY,JMP,JSR,LD
A,LDX,LDY,LSR,NOP,ORA
10030 DATA PHA,PHP,PLA,PLP,ROL,ROR,RTI,RTS,SBC,SEC,SED,SE
I,STA,STX,STY,TAX,TAY,TSX,TXA,TXS,TYA
```

Listing 1. Disassembler program.

BEEBUG FOR THE BBC MICRO

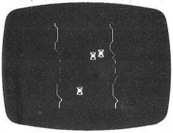
REGISTERED REFERRAL CENTRE FOR THE BBC PROJECT

BRITAIN'S LARGEST SINGLE —MICRO USER GROUP

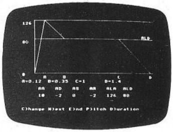
MEMBERSHIP NOW EXCEEDS 13,000

13,000 members can't be wrong — BEEBUG provides the best support for the BBC Micro.
BEEBUG Magazine — now 54 pages — devoted exclusively to the BBC Micro.
Programs—Hints & Tips—Major Articles—News—Reviews—Commentary.
PLUS members discount scheme with National Retailers. PLUS members Software Library.
10 Magazines a year. First issue April 1982. Reprints of all issues available to members.

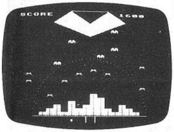
SCREEN SHOTS FROM PROGRAMS IN BEEBUG MAGAZINE



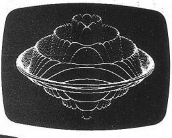
RACER
NOVEMBER
1982



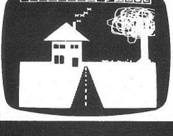
ENVELOPE
EDITOR
NOVEMBER 1982



SPACE
CITY
DECEMBER
1982



3D SHAPE
DECEMBER
1982



ARTIST
Painting by
Joystick
DECEMBER 1982

April Issue: 3D Noughts and Crosses, Moon Lander, Ellipse and 3D Surface. Plus articles on Upgrading to Model B, Making Sounds, and Operating System Calls.

May Issue: Careers, Bomber, Chords, Spiral and more. Plus articles on Graphics, Writing Games Programs, and Using the Assembler.

June Issue: Mazetrap, Mini Text Editor, Polygon; plus articles on upgrading, The User Port, TV set and Monitor review, Graphics part II, More Assembler Hints, Structuring in BBC Basic, plus BBC Bugs.

July Issue BEEB INVADERS and other programs—plus articles on using the Teletext mode, BBC cassette bugs fix, Software Review, using user defined keys. More on structuring in Basic. Using the User Port, and many hints and tips.

September Issue: High/Low Card Game, and Hangman Programs. Articles on Logic on the Beeb, Debugging, Moving multicoloured characters, creating new colours, Operating system 1.1. Plus Postbag, Hints and Tips, and Procedure Library.

October Issue: Program Features: Alien Attack; Calendar Generator; Union Jack; Memory Display utility. Plus articles on Beebugging; Improving Key Detection; Acorn Press Release on O.S.H.2; and Issue II Basic; The Tube and Second Processor Options; or New Series for less experienced users; and Software Reviews.

November Issue: Program Features: Racer (excellent 16K racing car game), Mini Text Editor (Mk2), Transparent Loader, Music with Memory, Harmonograph Emulator, New Character set for Modes 2 & 5; and cassette block-zero—bug retrieve. Plus articles on sound and envelope design—includes indispensable envelope editor program; Debugging Part 3, BBC Basics—Memory Maps and addressing explained; Serial Printer Port (RS423) and RGB upgrade. Plus a large number of Hints & Tips, and a guide to our past issues and their contents.

Dec/Jan Issue: Program Features: Space City (invader-type game), Breakout, Artist (Joystick painting program); Rescue (miraculously retrieves programs after bad loading or 'Bad Program' message); and Pack—a program to compact Basic programs. PLUS Disc System Review, Software reviews—including Wordwise, Book reviews, Adding Joystick interface to model A; How to access the video controller chip; and ideas for the newcomer; plus a new crop of Hints and Tips.

BEEBUGSOFT: BEEBUG SOFTWARE LIBRARY

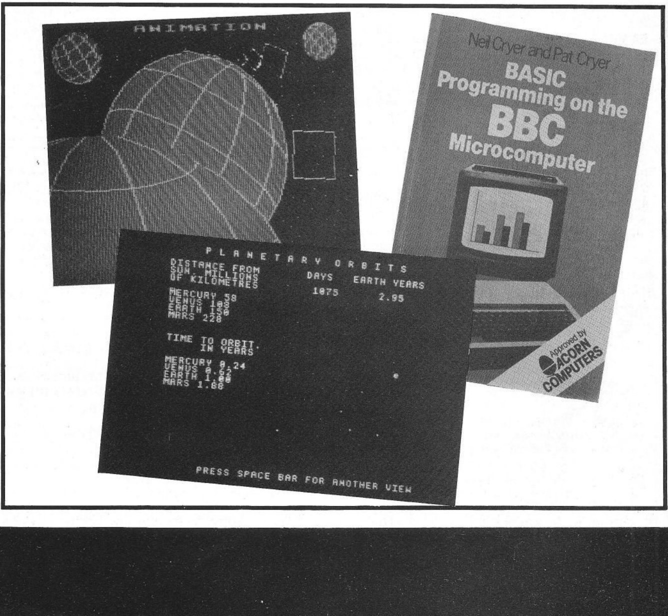
offers members a growing range of software from £3.50 per cassette.

1. Starfire (32K). 2. Moonlander (16K). 3D Noughts and Crosses (32K). 3. Shape Match (16K). Mindbender (16K). 4. Magic Eel (32K). 5. Cylon Attack (32K). 6. Astro-Tracker (32K).
Utilities: 1. Disassembler (16K). Redefine (16K). Mini Text Ed (32K).
Applications: 1. Superplot (32K). 2. Masterfile (32K).

13% DISCOUNT TO MEMBERS ON THE EXCELLENT WORDWISE WORD PROCESSING PACKAGE— THIS REPRESENTS A SAVING OF OVER £5.00.

Send £1.00 & SAE for Sample
Membership: UK 5.40 for six months, 9.90 for one year.
Overseas one year only: Europe £16.00, Middle East £19.00, Americas & Africa £21.00, Other Counrties £23.00
Make cheque to BEEBUG and send to: BEEBUG Dept 7, 374 Wandsworth Rd., London SW8 4TE
Send editorial material to: The Editor, BEEBUG, PO Box 50, St. Albans, Herts AL1 2AR

EDUCATION/INFORMATION



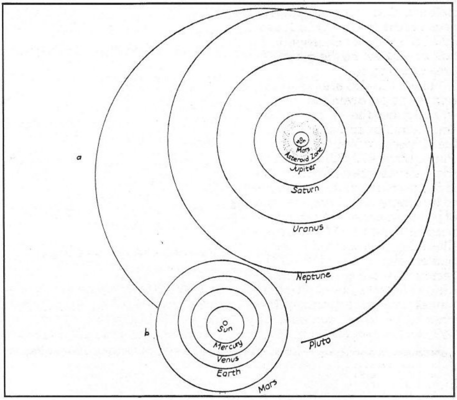
C R Dickens

KEPLER'S LAWS

The relative motions of the planets are illustrated in this program.

Since Kepler discovered the three laws which bear his name relating to the physical motions of the planets, and Newton interpreted them with his theory of gravitation, (helped by the apple!), the movement of the planets in their orbits has continuously fascinated astronomers. Kepler said that the planets move in ellipses with the Sun at one focus; the line connecting the Sun to a planet sweeps out equal areas in equal times; and the square of the revolution period is proportional to the cube of the distance from the Sun.

Scientists have often tried to find a formula which will give the relative distances of the planets from the Sun. There is one well known formula known as Bode's Law which worked well with the known planets. (Planets as far out as Saturn have been known as stars in the sky since early civilizations realized that these 'stars' were wanderers — planets — among the fixed stars). For this law, take the numbers 3, 6, 12, etc, doubling each time; add 4 and start with 4 to get the sequence 4, 7, 10, 16, etc. When Uranus was discovered its relative position came very close to 196, the eighth number in the series. However the law fell down badly following the discoveries of Neptune and Pluto at 301 and 396 when the series predicted 388 and 772.



LOOKING AT IT

The planets, themselves, revolve around the Sun more or less in the same plane, known as the ecliptic. Interestingly, too, they all revolve in the same direction. Whilst their orbits are ellipses, as stated by Kepler, the eccentricities, (a number between 0 and 1 representing the oblateness of the ellipse, between a circle, eccentricity of 0, and a straight line, eccentricity of 1) are so small that for all but two of the planets, on a scale the size of a computer display screen, the orbit is indistinguishable from a circle. Thus a viewer approaching the solar system from say the stars Vega or Deneb, North of the ecliptic, would see the planets' orbits through his image intensifier much as they are seen in this simulation. If that same extra terrestrial being has some means of variably expanding his own time frame then he will see their relative motions as they are portrayed by this program. Of course, he would need to suppress, as has been done here, the overwhelming brightness of the sun with his photon inhibitor.

The two exceptions to circles, Mercury and Pluto, clearly show up as ellipses when viewed at a suitable magnification. Pluto, in fact, appears to come within the path of Neptune such is its eccentricity, although since Pluto's path is inclined at 17 degrees to the ecliptic it does not come within millions of kilometres of its neighbour. There will seem to be a large gap between the orbits of Mars and Jupiter. This is represented by the fifth number of Bode's Law and approximately establishes the orbits of the minor planets, or asteroids, which are not shown in this simulation.

THE PROGRAM

The program will run on a minimum BBC Computer. But in order to fit it into the available free memory, the BASIC has to be typed in with very few spaces between the keywords and variables. (Typed as shown it will run on a Model A)

Take care to distinguish between a single quote (line feed) and a double quote (surrounding an alphabetic string). Integer variables are used whenever possible to save memory space.

Lines 20-100 set the orbital elements. Line 270 is long, and has to be typed as shown since all statements in it depend upon the result of the IF. . .THEN. Lines 290-310 set the background colour and define the graphics area and origin.

Lines 430-650 are the working part of the program. Lines 450-460 establish the eccentricities, and the major and minor axes of the ellipses for Mercury and Pluto. Lines 470-520 determine the distance apart of each point of the planets' orbits to be plotted so that there is an integer number of points to each orbit; but if the number of points is three or less (line 480) then the points are plotted 'freely'. Lines 560-600 remove the four previous points which plotted a planet's position, but retain a single point to show where it has been, and lines 610-640 plot the new position.

THINGS TO TRY

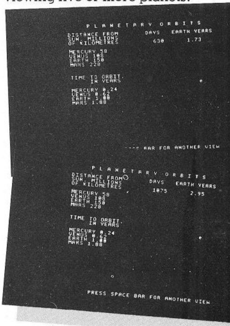
1) Look at the first two planets at seven day intervals and notice the eccentricity of Mercury.
2) View the orbit of Mars, the fourth planet, and see that Mars and Earth are in opposition approximately every two Earth years.
3) Notice the large gap, where the majority of the asteroids are, between Jupiter, the fifth planet, and Mars.
4) Look out as far as six planets, at 130 day intervals, and see that the inner orbits are just distinguishable.
5) View all the planets and see that Pluto actually passes inside Neptune's orbit.
6) The Earth rotates from West to East. The view is from the North, so when Venus is the Evening Star, as distinct from the Morning Star, is it approaching the Earth or receding?

HALLEY'S RETURN

In 1986 Halley's comet will return to the environs of the Earth. You can see this object's orbit by substituting it in the program in place of Pluto.

```
100  D%(8)=2700:P%(8)=27740
460  IF J%=8 THEN E=0.967:B=A*0.255
```

But note, for highly elliptical orbits this program simulates an object's ***average*** speed and not its true speed according to Kepler's Second Law. This comet's orbit is best seen when viewing five or more planets.



Typical screen displays produced by the program.

```
10   DIM D%(8),P%(8),I(8),A%(8),B%(8),C$(8)
20   D%(0)=58:P%(0)=88
30   D%(1)=108:P%(1)=225
40   D%(2)=150:P%(2)=365
50   D%(3)=228:P%(3)=687
60   D%(4)=778:P%(4)=4333
70   D%(5)=1427:P%(5)=10759
80   D%(6)=2870:P%(6)=30685
90   D%(7)=4497:P%(7)=60190
100  D%(8)=5969:P%(8)=90741
110  C$(0)="MERCURY "
120  C$(1)="VENUS "
130  C$(2)="EARTH "
140  C$(3)="MARS "
150  C$(4)="JUPITER "
160  C$(5)="SATURN "
170  C$(6)="URANUS "
180  C$(7)="NEPTUNE "
190  C$(8)="PLUTO "
200  MODE 7
210  H$="P L A N E T A R Y  O R B I T S"
220  PRINT'SPC(5);H$
230  INPUT''"HOW MANY OF THE 9 PLANETS DO YOU WISH TO
     VIEW ",S%
240  IF S%>9 OR S%<1 THEN PRINT''"TYPE 1 TO 9";GOTO 230
250  S%=S%-1:SIZE=D%(S%)/375
260  INPUT''''"TIME INTERVAL (DAYS)"''"TYPE A NUMBER TO
     SHOW THE PLANETS'"'"POSITIONS AT INTERVALS OF ",T
270  IF T<P%(S%)/90 OR T>P%(S%)/10 THEN PRINT';T" DAYS
     ARE UNSUITABLE FOR"'"VIEWING ";S%+1;" PLANETS"''
     INT(P%(S%)/50);" DAYS ARE SUBSTITUTED"''"PRESS
     SPACE BAR TO CONTINUE":T=P%(S%)/50:REPEAT UNTIL
     GET=32
280  FOR J%=0 TO 8:I(J%)=0:NEXT
290  MODE 4:VDU 19,0,4,0,0,0
300  VDU 24,420;32;1279;832;
310  VDU 29,870;432;
320  PRINT'SPC(5);H$
330  PRINT'"DISTANCE FROM[8 SPC]DAYS[3 SPC]EARTH YEARS"
     '"SUN. MILLIONS"'"OF KILOMETRES"'
340  FOR J%=0 TO S%
350  PRINT C$(J%);D%(J%):NEXT
360  PRINT''"TIME TO ORBIT."'"[5 SPC]IN YEARS"'
370  @%=131594
380  FOR J%=0 TO S%
390  PRINT C$(J%);P%(J%)/365
400  NEXT:@%=2570
410  PRINTTAB(6,31)"PRESS SPACE BAR FOR ANOTHER VIEW";
420  N%=0:M=0:G=PI/180
430  FOR J%=0 TO 8:R=D%(J%)/SIZE
440  A=R:B=R:E=0
450  IF J%=0 THEN E=0.2:B=A*0.98
460  IF J%=8 THEN E=0.26:B=A*0.96
470  P=P%(J%)/T
480  IF P>3 THEN P=INT(P+0.5)
490  I(J%)=I(J%)+360/P
500  Y=G*I(J%)
510  X=A*(COS(Y)-E)
520  Y=B*SIN(Y)
530  IF N%=1 THEN 550
540  IF J%=2 THEN VDU 5:MOVE X,Y:PRINT"e":N%=1
550  VDU 4
560  PLOT 71,A%(J%)+2,B%(J%)+2
570  PLOT 71,A%(J%)-2,B%(J%)+2
580  PLOT 71,A%(J%)+2,B%(J%)-2
590  PLOT 71,A%(J%)-2,B%(J%)-2
600  PLOT 69,A%(J%),B%(J%)
610  PLOT 69,X+2,Y+2
620  PLOT 69,X-2,Y+2
630  PLOT 69,X+2,Y-2
640  PLOT 69,X-2,Y-2
650  A%(J%)=X:B%(J%)=Y:NEXT
660  M=M+T
670  PRINTTAB(15,5),INT(M)
680  @%=131594
690  PRINTTAB(25,5)M/P%(2)
700  @%=2570
710  C=INKEY(0):IF C=32 THEN RUN
720  GOTO 430
730  END
```

H Roberts

LIFE

A game to play that really is true to life!

The game of Life was invented by John Conway of the University of Cambridge and was first described in ***Scientific American*** in October 1970. It is a game with simple rules, but it can produce complex and beautiful patterns. The game is played on a large grid of squares, each square has two states, Alive and Dead, and alternates between these states according to the following rules:

1) Every live cell with two or three live neighbours survives to the next generation.
2) Every live cell with more than three or less than two live neighbours dies.
3) Every dead cell with exactly three live neighbours becomes alive in the next generation.

All these births and deaths occur simultaneously and together they constitute a single generation. Many weird and unusual shapes have been discovered, and some of these are shown in the photographs. The glider moves up and to the right every two generations, the R-Pentomino grows incredibly, lasting 1137 generations before becoming stable and the spaceship moves to the right one, square every two generations.

The program is in two halves: a section in BASIC to allow the user to input his patterns, and a section in machine code which updates the board. The screen layout on the BBC Micro is very complicated, and this is reflected in the complexity of the machine code program. The program is geared for speed and so takes up a lot of memory space, it won't fit on the Model A.

INSTRUCTIONS

First type in the size (in characters — eight pixels per

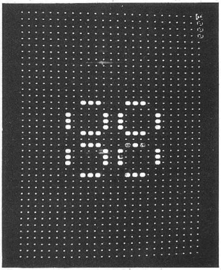
PROGRAM STRUCTURE

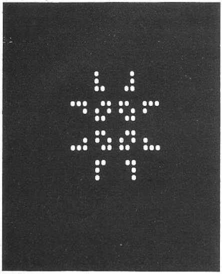
Line	Action
10-30	Print title
40-70	Input width and height of screen
80	Assemble machine code routine
90-120	Set up variables for plotting
130-240	Plot design on screen
250	Call machine code subroutine repeatedly
280-290	Start of loop for two pass assembler
310	Set up initial pointers into TABLE
320	Set up initial pointers onto screen
330	Set up previous line pointer to screen (dummy at start)
340	Set up vertical character count
350	Set up vertical pixel count
370	Set up mask for plotting
380	Set up horizontal character count
390	Set up horizontal pixel count
400	Get a byte from the screen containing eight alive or dead cells
410	Save it
420	Get the current pixel out of this byte and set the live cell count to zero
430	If it is alive add one to the live cell count of the surrounding cells
440	Add &7F to its own live cell count
460-490	The cell is alive in the next generation if the cell count is 3(birth), &82 or &83(remains alive)
500	The cell is dead or dying so rub out pixel on screen using plotting mask
510	The cell is alive so plot in pixel on screen using mask
520	Start updating pointers
530-550	Rotate the plotting mask. If it is rotated through the end of a byte, increment the plotting pointer to screen
560	Increment the pointer into TABLE
570	Decrement the number of the current pixel, if it is not the last pixel in the byte, loop back. Otherwise:
580	Read the next byte to be scanned,
590	Decrement the horizontal character count and if it is not the last character, loop back. Otherwise:
600	Move to the next line on the screen and the next line in TABLE
610	Decrement the vertical pixel count and loop back if it is not the last in the character
620	This is the last pixel in the vertical character so move to the next vertical line down
630	Decrement the vertical character count and loop if it is not the last
650	Return

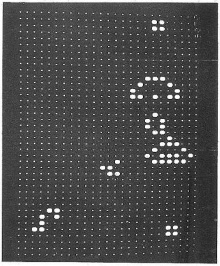
Table 3. A description of the program.

character) of the board you want. The program then clears the screen, and places a cursor at the centre of the board. Move the cursor around with the cursor controls, and use the Space key to flip between live points, and dead ones (black/white). When you are satisfied with the pattern you have drawn, press Return and the program will produce the next generations.

If you want to edit the picture press any key and the cursor will be put back on the screen. If you want to start press Escape and type RUN. Be careful of the edges of the board as the program does not check for collisions with them, and some strange patterns can result.





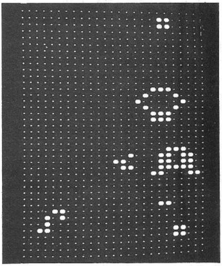


Some typical life formations: Please note that these are not from the program given here!

VARIABLES

Variables	Use
LOOP	Loop to print double height title
W%	Width of screen in characters and then pixels
H%	Height of screen in characters and then pixels
X%,Y%	The position of the cursor
C%	ASCII value of character input
B%	is 0 if the cursor is plotting dead cells is 1 if the cursor is plotting live cells

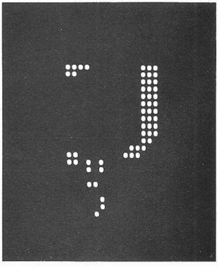
Table 1. A list of variables and their use.



VARIABLES

Location	Content
&70,&71	Address in TABLE of the current cell being examined
&72,&73	Address on the screen of the start of the previous line being examined
&74,&75	Address on the screen of the current pixel being plotted
& 76	The vertical number of the character being scanned
&77	The vertical number of the pixel being scanned
&78	Plotting mask
&79	The horizontal number of the pixel being scanned
&7A	The contents of the current byte being scanned
&7B	The horizontal number of the character being scanned
&7C,&7D	The start address of the current line on the screen
&7E,&7F	The start address of the current line in TABLE

Table 2. Where the program stores its information in memory.



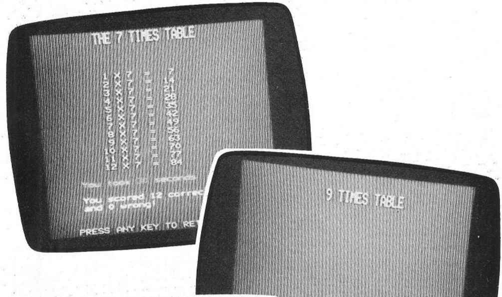
```
   10  MODE7:FOR LOOP=1 TO 2
   20    PRINTTAB(12);CHR$(132);CHR$(157);CHR$(131);CHR$(
141);"LIFE   ";CHR$(156)
   30    NEXT LOOP
   40  INPUT TAB(5,10)"WIDTH IN CHARACTERS (4-39)",W%
   50  IF W%<4ORW%>39 GOTO40
   60 , INPUT TAB(5)"HEIGHT (4-31)",H%
   70  IFH%<4ORH%>31 GOTO60
   80  GOSUB 270
   90  MODE4:VDU23;8202;0;0;0;
  100  *FX4,1
  110  H%=H%*8:W%=W%*8
  120  X%=W%/2:Y%=H%/2
  130  B%=0:REPEAT
  140    REPEAT PLOT69,X%*4,1024-Y%*4
  150      C%=INKEY(2)
  160      PLOT70,X%*4,1024-Y%*4
  170      IF C%=136 X%=(X%+W%-1)MODW%
  180      IF C%=137 X%=(X%+1)MODW%
  190      IF C%=138 Y%=(Y%+1)MODH%
  200      IF C%=139 Y%=(Y%-1+H%)MODH%
  210      UNTIL C%<>-1
  220    IF C%=95 B%=1-B%
  230    GCOL0,B%:PLOT69,X%*4,1024-Y%*4
  240    UNTIL C%=13
  250  REPEAT CALL LIFE:UNTIL NOT INKEY(0)
  260  GOTO120
  270  DIM LIFE 300,TABLE 2000
  280  FORJ=0TO2STEP2
  290    P%=LIFE:[OPTJ
  300    .LIFE
  310    LDA#(TABLE MOD 256):STA&70:STA&7E:LDA#(TABLE DIV
 256):STA&71:STA&7F
  320    LDA#0:STA&72:STA&7C:LDA#&58:STA&73:STA&7D
  330    LDA#&C0:STA&75
  340    LDA#H%:STA&76
  350    .WOP LDA#8:STA&77
  370    .LIN LDA#&7F:STA&78
  380    LDA#W%:STA&7B
  390    LDA#7:STA&79
  400    .CHARLOP LDY#0:LDA(&72),Y
  410    ASLA:STA&7A
  420    .PIXLOP LDA&7A:ASLA:STA&7A:LDY#8:LDA#0:STA(&70),
Y:BCC DEAD
  430    .LIF1 LDA(&70),Y:SEC:ADC#0:STA(&70),Y:DEY:BPL LI
F1
  440    LDY#4:LDA#&7F:CLC:ADC(&70),Y:STA(&70),Y:.DEAD
  450    LDY#0:LDA(&70),Y
  460    CMP#0:BEQ DEDD
  470    CMP#3:BEQ ALIVE
  480    CMP#&82:BEQ ALIVE
  490    CMP#&83:BEQ ALIVE
  500  ; .DEDD LDA&78:AND(&74),Y:STA(&74),Y:JMP PLIT
  510    .ALIVE LDA#&FF:SEC:SBC&78:ORA(&74),Y:STA(&74),Y
  520    .PLIT
  530    SEC:ROR&78:BCS NOCHAR
  540    ROR&78:LDA#8:CLC:ADC&74:STA&74:BCC LL2:INC&75:.L
L2
  550    .NOCHAR
  560    LDA#3:CLC:ADC&70:STA&70:BCC LL25:INC&71:.LL25
  570    DEC&79:BNE PIXLOP:LDA#8:STA&79:CLC:ADC&72:STA&72
:BCC LL3:INC&73:.LL3
  580    LDY#0:LDA(&72),Y:STA&7A
  590    DEC&7B:BEQ P%+5:JMP PIXLOP
  600    LDX&7C:STX&74:INX:STX&72:STX&7C:LDX&7D:STX&75:ST
X&73
  610    LDX&7E:INX:STX&7E:STX&70:BNE LL4:INC&7F:.LL4 LDA
&7F:STA&71
  620    DEC&77:BEQ P%+5:JMP LIN
  630    INC&73:LDA&72:CLC:ADC#56:STA&72:STA&7C:BCC LL5:I
NC&73:.LL5:LDA&73:STA&7D
  640    DEC&76:BEQ P%+5:JMP WOP
  650    RTS
  660    ]:NEXTJ
  670  RETURN
>
```

Listing 1. The game of Life.

Ian Nichols

MULTI TEST

Multiplication is definitely the name of the game.



Many husbands will have attempted to persuade their disbelieving wives (or even ***vice versa***!) that a BBC Microcomputer is just what the family most needs, by emphasising the tremendous educational possibilities that will result for their children. This, then, is the program you have been waiting for. It has been written by just such a husband, for number 2 son who was having trouble learning his tables. They do not seem to teach children tables the way they used to, do they? Never mind, this program is not only instructional, but is fun to use and has an element of competition in it as well. It will run on either a 16K Model A or a Model B.

The first screen is a menu which initially asks what multiplication table you wish to work with. It will only allow tables between 2* and 12*, although the program is easily altered to extend the range. Having chosen a number, you are presented with three choices and asked to press keys 1, 2 or 3:

1 — complete Z times table

This prints out the complete table chosen; pressing any key will then return you to the menu screen.

2 — Z times table for you to fill in yourself

This option prints the first part of each line of the table, such as 4 x 6 = ? and you then have to put in the correct answer. If you give the correct answer, a musical chord is played, and the line is printed. If you give an incorrect answer, a raspberry-like sound is produced, and the correct answer is displayed. After a few seconds the next line is printed. When all 12 lines have been attempted, the amount of time you took (in seconds) is printed, together with the number of correct and incorrect answers. A short but familiar, five note extra-terrestrial tune is played as well. Pressing any key again returns you to the menu.

3 — random tests from the Z times table

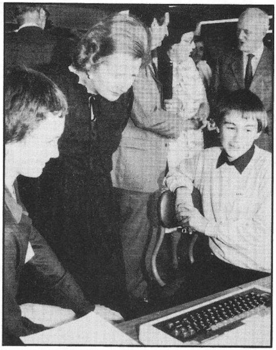
This last option is similar to the second except that the elements of the table are selected in random order.

```
10 REM  Multiplication tests
20 REM          by
30 REM  I G Nicholls Dec 1982
40 REM
50 REM  Initialisation
60 REM
100 DIM N(13)
110 DATA53,61,69,73,81,89,99,101,109,117,121,129,137
120 FOR I=1TO13
130    READ N(I)
140      NEXT:GOTO250
142 REM
143 REM  Arpeggio
144 REM
150 DEF PROCsound3
160 FOR A=1TO8
170      SOUND1,-15,N(A),4
180      SOUND2,-15,N(A+2),4
190      SOUND3,-15,N(A+4),4
200      NEXT
210 SOUND1,-15,N(8),12
220 SOUND2,-15,N(10),12
230 SOUND3,-15,N(12),12
232 REM
233 REM  More Initialisation
234 REM
240 ENDPROC
250 *FX11,0
260 ON ERROR GOTO2150
270 MODE7
280 DIM RAND%(12)
290 G$=STRING$(34," ")
300 REM
310 REM  Main program loop - start
320 REM
330 PROCmenu
340 ON F% GOTO350,370,390
350 PROCtable1
360 GOTO330
370 PROCtable2
380 GOTO330
390 PROCtest
400 GOTO330
410 REM
420 REM  Main program loop - end
430 REM
440 REM
450 REM  Menu screen display
460 REM
470 DEF PROCmenu
480 FOR I%=0TO24
490    PRINTTAB(0,I%)CHR$135;CHR$157;
500    NEXT
510 PRINTTAB(8,3);CHR$141;CHR$129;"Multiplication Tests
"
520 PRINTTAB(8,4);CHR$141;CHR$129;"Multiplication Tests
"
530 PRINTTAB(8,5);CHR$141;CHR$131;"====================
"
540 PRINTTAB(8,6);CHR$141;CHR$131;"====================
"
550 PRINTTAB(3,9);CHR$130;"Which number do you want to
use?"
560 *FX15,1
570 PRINTTAB(3,10);CHR$130;"Type a number between 2 and
12"
580 PRINTTAB(2,11)CHR$132;:INPUTTAB(3,11),Z%
590 IF Z%<2 OR Z%>12 PRINTTAB(3,11)"     ":GOTO580
600 PRINTTAB(3,11)"  "
610   PRINTTAB(11,12);CHR$136;CHR$132;Z%;" TIMES TABLE"
620 PRINTTAB(2,15);CHR$132;"Which option do you want to
try?"
630 PRINTTAB(2,17);CHR$133;"1 - complete ";Z%;" times t
able"
640 PRINTTAB(2,19);CHR$133;"2 - ";Z%;" times table for
you to";TAB(2,20);CHR$133;"     fill in yourself"
650 PRINTTAB(2,21);CHR$133;"3 - random tests from the "
;Z%;TAB(2,22);CHR$133;"     times table"
660 PRINTTAB(2,24);CHR$136;CHR$129;"   Enter your choic
e 1,2 or 3";
670 *FX15,1
680 F%=GET:F%=F%-48:IF F%<1 OR F%>3 GOTO680
690 ENDPROC
700 REM
710 REM  Print complete table
720 REM
730 DEF PROCtable1
740 CLS
750 FOR I%=0TO24
760    PRINTTAB(0,I%);CHR$134;CHR$157;CHR$132;
770    NEXT
780 PRINTCHR$30
790 VDU28,3,24,39,0
800 PRINTCHR$141;TAB(8);"THE ";Z%;" TIMES TABLE"'CHR$14
1;TAB(8);"THE ";Z%;" TIMES TABLE"'''
810 FOR J%=1TO12
820    H%=J%*Z%
840    IF H%>9 AND H%<100 A$=" "
850    IF H%>99 A$=""
860    PRINT,J%;" X ";Z%;"  = ";A$;H%
870    NEXT
880 PRINT''''"  Press any Key to return to menu"
890 Z=GET
900 VDU26
910 CLS
920 ENDPROC
930 REM
940 REM  Step through table
950 REM
960 DEF PROCtable2
970 LOCAL tot%
980 tot%=0
990 CLS
1000 FOR I%=0TO24
1010    PRINTTAB(0,I%);CHR$132;CHR$157;CHR$131;
1020    NEXT
1030 PRINTCHR$30
1040 VDU28,3,24,39,0
1050 PRINTCHR$141;TAB(8);"THE ";Z%;" TIMES TABLE"'CHR$14
1;TAB(8);"THE ";Z%;" TIMES TABLE"'''
1060 TT=TIME
1070 FOR J%=1TO12
1080    H%=J%*Z%
1090    IF H%<10 A$="  "
1100    IF H%>9 AND H%<100 A$=" "
1110    IF H%>99 A$=""
1120    PRINT,J%;" X ";Z%;"  = ";
1130    INPUT,G%
1140    IF G%=H% PRINTCHR$11;CHR$13,J%;" X ";Z%;"  = ";A$
;H%;"      ":PROCsound2:GOTO1200
1150    PRINTCHR$11;CHR$13;"WRONG - the answer is ";H%;"
   ":PROCsound1
1160    FOR Q%=1TO 10000
1170      NEXT
1180    PRINTCHR$11;CHR$13,J%;" X ";Z%;"  = ";A$;H%;"
  "
1190    tot%=tot%+1
1200    NEXT
1210 TT=TIME-TT
1220 TT=TT/10:T%=TT:TT=T%:TT=TT/10
1230 PRINTTAB(3,18);CHR$133;"You took";CHR$136;TT;CHR$13
7;"seconds"
1240 PRINTTAB(3,20);CHR$135"You scored ";12-tot%;" corre
ct"
1250 PRINTTAB(3,21);CHR$135"and ";tot%;" wrong!"
1260 SOUND1,-15,97,10
1270 SOUND1,-15,105,10
1280 SOUND1,-15,89,10:SOUND1,-15,41,10:SOUND1,-15,69,20
1290 PRINTTAB(3,24)"PRESS ANY KEY TO RETURN TO MENU";
1300 Z=GET
1310 VDU26
1320 CLS
1330 ENDPROC
1340 REM
1350 REM  Random tests from table
1360 REM
1370 DEF PROCtest
1380 C%=0
1390 VDU26
1400 CLS
1410 FOR I%=0TO24
1420    PRINTTAB(0,I%)CHR$132;CHR$157;CHR$135;
1430    NEXT
1440 VDU28,3,24,39,0
1450 PRINTTAB(10,2)CHR$141;CHR$131;Z%;" TIMES TABLE"
1460 PRINTTAB(10,3)CHR$141;CHR$131;Z%;" TIMES TABLE"
1470 FOR I%=1TO12
1480    RAND%(I%)=0
1490    NEXT
1500 FOR I%=1TO12
1510    REPEAT
1520      J%=RND(12):FLAG%=0
1530      FOR K%=1TO I%
1540        IF J%=RAND%(K%) FLAG%=1
1550        NEXT
1560      UNTIL(FLAG%=0)
1570    RAND%(I%)=J%
1580    NEXT
1590 TT%=TIME
1600 FOR I%=1TO12
1610    ANS%=0
1620    PRINTTAB(12,10);CHR$141;RAND%(I%);" X ";Z%;" = ";
1630    PRINTTAB(12,11);CHR$141;RAND%(I%);" X ";Z%;" = ";
1640    *FX15,0
1650    ZZ%=GET
1660    IF ZZ%=13 GOTO1700
1670    ZZ%=ZZ%-48:IF ZZ%<0 OR ZZ%>9 GOTO1650
1680    PRINTTAB(POS,10);ZZ%;:PRINTTAB(POS-1,11);ZZ%;:ANS
%=ANS%*10+ZZ%
```


* 830 SEE LINE 1090

```
1690   GOTO1650
1700   IF RAND%(I%)*Z%=ANS% PROCcorrect:C%=C%+1 ELSE PRO
Cwrong
1710   PRINTTAB(3,10);G$;TAB(3,11);G$;
1720   NEXT:TT%=TIME-TT%:TT%=TT% DIV 10:T=TT%:T=T/10
1730 PRINTTAB(3,21);CHR$134;"You took";CHR$136;T;CHR$137
;"seconds,";TAB(3,22);CHR$134;"you got ";C%;" correct and
 ";12-C%;" wrong":PROCsound3
1740 FOR Q%=0TO30000:NEXT
1750 VDU26:CLS
1760 ENDPROC
1770 REM
1780 REM  Procedure for wrong answer
1790 REM
1800 DEF PROCwrong
1810 PROCsound1
1820 PRINTTAB(12,16);CHR$141;CHR$129;"WRONG"TAB(12,17);C
HR$141;CHR$129;"WRONG";
1830 FOR Q%=0TO10000:NEXT
1840 PRINTTAB(3,16);G$;TAB(3,17);G$;TAB(11,16);CHR$141;C
HR$134;RAND%(I%);" X ";Z%;" = ";RAND%(I%)*Z%;TAB(11,17);C
HR$141;CHR$134;RAND%(I%);" X ";Z%;" = ";RAND%(I%)*Z%;
1850 FOR Q%=0TO15000:NEXT
1860 PRINTTAB(3,16);G$;TAB(3,17);G$;
1870 ENDPROC
1880 REM
1890 REM  Procedure for correct answer
1900 REM
1910 DEF PROCcorrect
1920 PRINTTAB(11,20)CHR$141;CHR$130;"CORRECT"TAB(11,21)C
HR$141;CHR$130;"CORRECT";
1930 J%=I%
1940 SOUND1,-11,53+4*J%,10
1950 SOUND2,-11,69+4*J%,10
1960 SOUND3,-11,81+4*J%,10
1970 FOR Q%=0TO10000:NEXT
1980 PRINTTAB(3,20);G$;TAB(3,21);G$;
1990 ENDPROC
2000 REM
2010 REM  Sound for wrong answer
2020 REM
2030 DEF PROCsound1
2040 ENVELOPE1,0,2,-2,2,6,12,6,127,0,0,-127,126,0
2050 SOUND0,1,3,12:SOUND1,0,144,12
2060 ENDPROC
2070 REM
2080 REM  Sound for correct answer
2090 REM
2100 DEF PROCsound2
2110 SOUND1,-11,53+4*J%,10
2120 SOUND2,-11,69+4*J%,10
2130 SOUND3,-11,81+4*J%,10
2140 ENDPROC
2150 *FX12,0
2160 VDU26:CLS:PRINT'':REPORT:PRINT" at line ";ERL
```

Listing 1. The program for Multi Test.



Manchester Home Computer Show
MIDLAND HOTEL
April 22/23/24

Your diary dates are:
Glasgow . . . . May
Birmingham . . . . June
Nottingham . . . . September
Newcastle . . . . October
Bristol . . . . December

Sponsored jointly by:
Personal Computing Today
ZX Computing
Computing Today
Micro Update
Personal Software

At the Home Computer Shows will be a complete cross section of the hardware and software available to the home user. The emphasis is on the lower end of the price bracket with computers from £50-£400.

If you are interested in computers and what they can do for you then come along to our **COMPUTER ADVICE CENTRE:** experts will be on hand to give you impartial advice on equipment available.

Try out the machines in our own demonstration area and see programs running covering educational, games and small business applications.

There is a **COMPETITION** at every show to:

WIN TWO COMPUTERS.

Win a computer for yourself as well as one for the school of your choice: free entry form with advance tickets. Also available at the show with the show catalogue.

ADMISSION £2.00 (CHILDREN UNDER 8 & O.A.P's FREE)
AND IF YOU'RE A PARTY OF 20 OR MORE, THERE'S A 25% DISCOUNT

Friday 22 April '83 (10am-6pm)
Saturday 23 April '83 (10am-6pm)
Sunday 24 April '83 (10am-4pm)

The Manchester Home Computer Show
Midland Hotel. (Opposite Town Hall).

For advance tickets send cheque/postal order to:
ASP Exhibitions
Argus Specialist Publications
145 Charing Cross Rd,
London WC2H OEE
Tel: 01-437-1002

ADVANCE TICKET OFFER
MANCHESTER HOME COMPUTER SHOW
SAVE £1.00

Name Mr/Mrs/Miss . . . .
Address . . . .

J Ruston

BBC BITS

A collection of short ideas to liven up your programs.

SNAKE

This program is a ball bouncer with a difference! Two balls are bounced around a MODE 7 screen (in colour), one following directly behind the other. The forward one leaves a trail of asterisks, and the backward one rubs out the asterisks as it goes. The net effect is to move a snake of asterisks around the screen.

The main claim to fame of the program is the incredible speed with which it runs — evidence of the sophistication of the BBC computer.

```
10 REM Snake
20 MODE 7
30 FOR T%=0 TO 24
40 VDU 31,0,T%,132
50 NEXT T%
60 VDU 23;8202;0;0;0;
70 X%=24:Y%=24
80 L%=0 :M%=0
90 A%=1:B%=1
100 C%=1:D%=1
110 REPEAT
120 VDU 31,X%,Y%,42,31,L%,M%,32
130 IF X%+A%>39 OR X%+A%<1 A%=-A%
140 IF L%+C%>39 OR L%+C%<1 C%=-C%
150 IF Y%+B%>24 OR Y%+B%<0 B%=-B%
160 IF M%+D%>24 OR M%+D%<0 D%=-D%
170 X%=X%+A%:Y%=Y%+B%
180 L%=L%+C%:M%=M%+D%
190 UNTIL FALSE
```

NON-INTERRUPTABLE PROGRAMS

The usual way to make programs un-interruptable is to use on ERROR GOTO to disable the escape keys, and *KEY 10 "OLD RUN" to disable the Break key.

The problem with this method is that if a 'hard Reset' is carried out (that is, pressing Break in such a way as to print out the memory size, as well as the normal sign on message), the program is protected.

The program given below illustrates an alternative way of approaching the problem. Please note that the last two columns of printout represent the typical output of the program, fustrating huh? First I have defined the Break key to create a dynamic variable. This means that typing OLD will not work.

The next step (line 1000) automatically breaks the machine every time Escape is pressed. The call responsible for this is the one to &DBBE.

Mingled with the program is a demo. While it runs just press Escape or Break.

```
10 *KEY 10 "VAR=PI|M"
20 ON ERROR GOTO 1000
30 REM And so on with rest of program
40 REPEAT
50 PRINT "HELLO"
60 UNTIL FALSE
1000 IF ERR=17 THEN CALL &DBBE
1010 REPORT
1020 PRINT " at line ";ERL
1030 END
```

```
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
HELLO
```

```
VAR=PI
>OLD
Bad program
>LIST
Bad program
>Oh well...
```

PRINTING TEXT UPWARDS

This program is useful for labelling the Y axes of graphs. It draws text from the current graphics cursor position — but directly up, rather than across, the screen. Lines 40 to 70 form a simple demo.

The program is set to work in MODEs 4 and 1, or at a pinch, 0. If you want to use it in MODEs 2 and 5, change the '4' in line 160 (as in 'X% + L% *4') to be '8'.

The program accesses the BBC Micro's built-in character generator directly, and so will not work at the other end of the Tube.

The character generator starts at &C000 with the bytes for a space, and extends up to code 127. It is conceivable that in future versions of the operating system the character generator will be moved. If so, alter the address in line 1030 to be 256 less than the start of the generator. This version works fine on the operating system which gives 'OS Eprom 0.10' in response to *FX 0.

The procedure 'PROCtext_up' takes three arguments. The first is the string to be printed, and the second two are the co-ordinates the printing should start at.

Essentially, the program just accesses individually each bit of the sections of the character generator which make up the text to be printed, and then turn these through 90 degrees.

The second program effectively prints text downwards in the same way. You can see that it is similar.

```
   10 REM Printing text upwards
   20 REM
   30 REM **********************
   40 MODE 4
   50 REPEAT
   60 PROCtext_up("Led Zeppelin",RND(120
0),RND(1000))
   70 UNTIL FALSE
 1000 DEF PROCtext_up(A$,X%,Y%)
 1010 LOCAL A%,C%,L%,M%
 1020 FOR C%=1 TO LEN(A$)
 1030 A%=&BF00+ASC(MID$(A$,C%,1))*8
 1040 FOR L%=0 TO 7
 1050 FOR M%=0 TO 7
 1060 IF (2^M% AND A%?L%)<>0 THEN PLOT 6
9,X%+L%*4,Y%+28-M%*4+C%*32
 1070 NEXT M%,L%,C%
 1080 ENDPROC
```

```
   10 REM Printing text upwards
   11 REM (But back to front this time)
   20 REM (C) 1982 Jeremy Ruston
   30 REM **********************
   40 MODE 4
   50 REPEAT
   60 PROCtext_up("Led Zeppelin",RND(120
0),RND(1000))
   70 UNTIL FALSE
 1000 DEF PROCtext_up(A$,X%,Y%)
 1010 LOCAL A%,C%,L%,M%
 1020 FOR C%=1 TO LEN(A$)
 1030 A%=&BF00+ASC(MID$(A$,C%,1))*8
 1040 FOR L%=0 TO 7
 1050 FOR M%=0 TO 7
 1060 IF (2^M% AND A%?L%)<>0 THEN PLOT 6
9,X%+28-L%*4,Y%+M%*4-C%*32
 1070 NEXT M%,L%,C%
 1080 ENDPROC
```

READ ERROR

This function allows you to recall the last error — in words, rather than via ERR.

The function is based upon the fact that locations &FD and &FE contain one less than the address of the last error message. Given that the message is terminated by a zero byte, it is easy to read it into a string and then exit the function with it

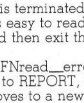
Print FNread_error is identical to REPORT, except that it moves to a new line after printing the message. The difference between the two is exhibited in line 100, where FNread_error is assigned to the string variable.

Note that the actual function definition is only lines 1000 to 1080 — the rest is a simple demo.

```
   10 ON ERROR GOTO 100
   20 REPEAT
   30 UNTIL FALSE
  100 A$=FNread_error
  110 IF A$="Escape" THEN END
  120 RUN
 1000 DEF FNread_error
 1010 LOCAL A$,T%
 1020 A$=""
 1030 T%=(!&FD AND &FFFF)+1
 1040 REPEAT
 1050 A$=A$+CHR$(?T%)
 1060 T%=T%+1
 1070 UNTIL ?T%=0
 1080=A$
```

STRINGY LETTERS

This is a large character printing program with a difference. It doesn't just print out the characters using a filled-in block, it intelligently uses empty boxes, closed off at the right places to give text of the form:



The completely general parts of the routine are lines 60 to 340 and lines 410 to 640. The other parts just form a little demo.

```
  10 REM Stringy letters
  20 REM
  30 M%=0
  40 IF M%=0 THEN MODE 0 ELSE MODE 4
  50 IF M%=4 THEN M%=5 ELSE M%=10
  60 DATA 00,00,00,00,00,00,00,00
  70 DATA FF,00,00,00,00,00,00,00
  80 DATA 80,80,80,80,80,80,80,80
  90 DATA FF,80,80,80,80,80,80,80
 100 DATA 00,00,00,00,00,00,00,FF
 110 DATA FF,00,00,00,00,00,00,FF
 120 DATA 80,80,80,80,80,80,80,FF
 130 DATA FF,80,80,80,80,80,80,FF
 140 DATA 01,01,01,01,01,01,01,01
 150 DATA FF,01,01,01,01,01,01,01
 160 DATA 81,81,81,81,81,81,81,81
 170 DATA FF,81,81,81,81,81,81,81
 180 DATA 01,01,01,01,01,01,01,FF
 190 DATA FF,01,01,01,01,01,01,FF
 200 DATA 81,81,81,81,81,81,81,FF
 210 DATA FF,81,81,81,81,81,81,FF
 220 FOR T%=224 TO 239
 230 VDU 23,T%
 240 FOR G%=0 TO 7
 250 READ A$
 260 VDU EVAL("&"+A$)
 270 NEXT G%,T%
 280 DIM F%(9,9)
 290 FOR T%=0 TO 9
 300 F%(T%,0)=0
 310 F%(T%,9)=0
 320 F%(0,T%)=0
 330 F%(9,T%)=0
 340 NEXT T%
 350 REM ***************************
 360 REPEAT
 370 INPUT LINE ''',A$
 380 UNTIL LEN(A$)<=M%*4
 390 VDU 23;8202;0;0;0;
 400 CLS
 410 FOR T%=1 TO LEN(A$)
 420 A%=&BF00+ASC(MID$(A$,T%,1))*8
 430 FOR Y%=0 TO 7
 440 FOR X%=0 TO 7
 450 F%(X%+1,Y%+1)=(2^X%) AND Y%?A%
 460 NEXT X%,Y%
 470 REM ***************************
 480 FOR Y%=0 TO 7
 490 FOR X%=0 TO 7
 500 IF F%(X%+1,Y%+1)<>0 THEN PROCyes
 510 NEXT X%,Y%,T%
 520 VDU 30
 530 END
 540 REM ***************************
 550 DEF PROCyes
 560 LOCAL H%
 570 H%=0
 580 IF F%(X%+1,Y%)=0 THEN H%=H%+1
 590 IF F%(X%+2,Y%+1)=0 THEN H%=H%+2
 600 IF F%(X%+1,Y%+2)=0 THEN H%=H%+4
 610 IF F%(X%,Y%+1)=0 THEN H%=H%+8
 620 R%=T%-1
 630 VDU 31,((R% MOD M%)*8)+7-X%,((R% D
IV M%)*8)+Y%,224+H%
 640 ENDPROC
```

Ian Nicholls

IN CHORUS

Sing along with your musical micro – with three-part harmony.

The new **User Guide** for the BBC Microcomputer contains three detailed sections on the SOUND and ENVELOPE commands (pp 180-187, 244-248 and 347-353), but only scant attention is given to two very useful features, the synchronisation of more than one note and the noise channel (channel 0). We are going to explore these two features, and use them to produce a tune. The tune will be the Chorale from Cantata No 147 by J S Bach, otherwise known as Jesu Joy of Man's Desiring, and we will play it in full three-part harmony!

PERIODIC NOISE

On page 349 of the **User Guide** we are told that we can have periodic noise on channel 0 of frequency determined by the pitch setting of channel 1, by putting 3 as the value of the third parameter in the SOUND command. Well let's have a go; try entering the following two line program and running it.

```
10  SOUND 0,-15,3,18
20  SOUND 1,-15,172,18
```

What you hear is two sounds, not one, a high pitched sound and one of much lower pitch. If you alter the volume parameter in line 20 to zero, and run it again:

```
10  SOUND 0,-15,3,18
20  SOUND 1,0,172,18
```

You will notice that the higher-pitched sound has disappeared, but the lower pitched sound is still there. The other thing that you will notice is how low the pitch is. One of the few apparent drawbacks of the sound capabilities of the BBC Microcomputer is that it does not, at first sight, appear to be able to produce notes below A# in octave 1. Just to remind you, the table giving the correspondence between the note played and the value for the pitch parameter, is shown in Table 1. The pitch parameter of the sound command (SOUND C,A,P,D), P, can only take positive values between 0 and 255. Most of octave 1 and all of octave 0 seem to be inaccessible. Let's experiment a bit more: try entering the following three commands, to load instructions into the function keys:

```
*KEY0SOUND2,-15,1,18|M
*KEY1SOUND0,-15,3,18:SOUND1,0,188,18|M
*KEY2SOUND0,-15,3,18:SOUND1,0,112,18|M
```

(where | is obtained by pressing Shift and the \ key : it appears as II in Mode 7).

Now press function key 0, and then function key 1. They are the same note, but you will notice that the pitch setting in one case is 1, and in the other case it is 188. To prove that we have found a way of obtaining lower pitched notes than those in the **User Guide**, press function key 2. To appreciate this low a note properly, you really need to feed the sound output to a larger speaker in a properly-designed enclosure. Later on in this article you can see how to do that.

What we need now is a table, similar to Table 1, which shows what pitch settings,P, to use in the:

```
SOUND0,A,3,D
SOUND1,0,P,D
```

pair of commands, in order to obtain particular musical notes, Table 2 is just that. It was obtained using the author's ear, rather than by reference to a formula since, at the time of writing he had not been able to work out the formula!

MULTINOTE SYNCHRONISATION

In any piece of music written for

	Octave number						
Note	1	2	3	4	5	6	7
B	1	49	97	145	193	241	
A#	0	45	93	141	189	237	
A		41	89	137	185	233	
G#		37	85	133	181	229	
G		33	81	129	177	225	
F#		29	77	125	173	221	
F		25	73	121	169	217	
E		21	69	117	165	213	
D#		17	65	113	161	209	
D		13	61	109	157	205	253
C#		9	57	105	153	201	249
C		5	53	101	149	197	245

Table 1. Notes played for pitch parameter values.

an instrument, such as a piano, that can sound more than one note at a time, there is a need to make sure that more than one note *is* sounded at the same time using the sound chip on the BBC Microcomputer. The way this chip (the Texas Instruments SN76489A) is accessed by the machine operating system is that a string of notes for each channel is maintained in a buffer and, as soon as one note has been completed on a channel, the next one is sent along from the buffer.

If you want to make sure that two or three notes sound together then you need to modify the first parameter of the sound command. The full version of the first parameter is a four-digit hexadecimal number (so it has an ampersand, &, in front of it) &HSFN.

H— takes the values 0 or 1, and is used to ensure the release phase of a sound played using an envelope is completed. Normally it is set to 0.

S— is the digit that we are interested in taking the values 0 to 3, one less than the number of notes to be synchronised.

F— takes the values 0 to 1, and controls whether the next note eliminates the queue of notes waiting to be played on any channel, and is played immediately.

N— is the channel number itself, taking values from 0 to 3.

If we want to synchronise two notes on channels 1 and 2, then we would write:

```
1Ø  SOUND&Ø1Ø1,-15,21,18
2Ø  SOUND&Ø1Ø2,-15,33,18
```

In fact, if you entered the following two lines instead, it would sound just the same!

```
1Ø  SOUND1,-15,21,18
2Ø  SOUND2,-15,33,18
```

However, if you add line 5 to these two lines:

```
5  SOUND1,-15,5,18
```

then lines 5 and 20 will sound together, followed by line 10. In order to play line 5, followed by lines 10 and 20 together, you need to write:

```
 5  SOUND1,-15,5,18
1Ø  SOUND&Ø1Ø1,-15,21,18
2Ø  SOUND&Ø1Ø2,-15,33,18
```

The computer will not play the sound on channel two, until another sound is available to be synchronised with it.

PLAY THAT TUNE

Jesu Joy of Man's Desiring is a well known tune, and a good test of any microcomputer's ability to play three-part harmony. Listing 1 shows the program that plays it. This will run on both the Model A and Model B. The essentials of the program are the following five lines:

```
14Ø  RESTORE 32Ø
15Ø  FOR I%=Ø TO 38Ø
16Ø  READ A%,B%,C%,D%
17Ø  SOUND A%,B%,C%,D%
23Ø  NEXT
```

These lines read the contents of the DATA statements note by note and immediately send them into the sound buffer ready to be played. Using Tables 1 and 2 you can see that the first statement plays the note G in octave 0 on the noise channel (pitch value of 124) for a duration of 18 twentieths of a second synchronised with rests, ie silence, of duration 18 on channel 2 and duration 6 on channel 3; then it plays G in octave 4 (pitch value of 129) for duration of 6, followed by A in octave 4 (pitch value of 137) for a further 6 twentieths of a second.

For those brave souls who decide to key in the program in Listing 1 it is worth putting commonly used numbers such as &0300 — &0303 and the word DATA into the function keys, to save your fingers.

The rest of the program produces a title page in Teletext mode, with the name of the piece enclosed in a rectangular box drawn, using Teletext graphics characters, by lines 1300 — 1400 in PROCtitle. When the tune is being played, as each note is read in, line 180 calculates a number between 145 and 150 derived from the loop counter, K%. In Teletext mode these numbers, used as control characters, give the following coloured graphics characters:

145 red graphics
146 green graphics
147 yellow graphics
148 blue graphics
149 magenta graphics
150 cyan graphics

Lines 190 — 220 place these controls in the positions on the screen which control the graphics colour of the rectangular box. As each note is read in, the colour of the box changes. It does look quite pretty, and there is plenty of scope for experimentation with it! How about plotting coloured notes on the screen?

BETTER QUALITY SOUND

I mentioned earlier that it was possible to connect a better

	OCTAVE NUMBER			
Note	0	1	2	3
B	140	188	234	
A#	134	183	231	
A	130	180	228	
G#	127	177	224	
G	124	172	220	
F#	120	168	216	
F	116	164	212	
E	112	160	208	255
D#	108	156	203	252
D	104	152	200	246
C#	100	147	196	243
C	96	144	192	240

Table 2. Pitch settings using periodic noise on channel 0.

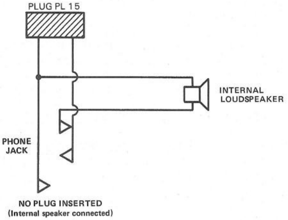
quality loudspeaker to the BBC Microcomputer, in order to appreciate the sounds of which it is capable. The following suggestions will almost certainly be taken by Acorn Computers as invalidating your guarantee, so if you do not understand anything about electronics, or cannot use a soldering iron, or both, don't attempt it! The author cannot be held responsible for any disasters which might ensue. However, having said that, the necessary modification is really quite simple.

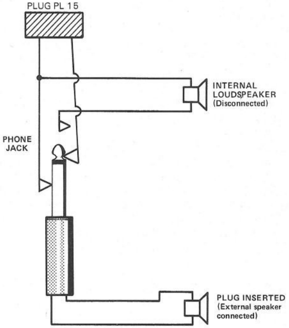
The tiny internal loudspeaker is connected via two wires to plug PL15 on the main circuit board. To obtain access to this you not only have to remove the top of the computer but you also have to remove the keyboard PCB. PL15 is on the left-hand side at the front of the circuit board near the disc drive socket. We could just disconnect the two speaker wires and re-connect them to a new socket at the back of the machine into which an external speaker can be plugged. However, it is more useful to allow the possibility of either the internal loudspeaker being used or an external one. To do this we need to use a socket which automatically connects the internal loudspeaker when no plug is inserted, and disconnects it when one is.

A suitable socket is an ⅛" miniature closed circuit phone jack, available from, for example, Radiospares or Tandy.

The best place to put the new socket is into the redundant Reset button aperture at the rear of the machine. The Break key has replaced this button, so the hole will not be needed.

Having made this modification, you may well now be unhappy with the noisy sound, even when no proper sounds are being played. This noise is coming from signals along the computer's data and address busses — just listen to it when a program is executing! The noise can be considerably reduced by placing a ¼ watt 10k resistor between pins 15 and 16 of the 1 MHz expansion bus plug. The easiest way of doing this is to buy a socket to fit the 1 MHz plug, and to fit the resistor across its appropriate pins. The socket you need is an Insulation Displacement Connector (IDC for short) — Speedblock type — female header socket 2 * 17 pin. These cost about £1.60 each. The improvement is well worth the effort, and does not invalidate your guarantee!






```
>>
>L.
   10 REM  Jesu Joy Of Man's Desiring by J S Bach
   20 REM
   30 REM  Transcribed for BBC Micro
   40 REM  by I G Nicholls Dec 1982
   50 REM
  110 MODE7
  120 REPEAT
  130   PROCtitle
  140   RESTORE320
  150   FOR I%=0TO380
  160     READ A%,B%,C%,D%
  170     SOUND A%,B%,C%,D%
  180     K%=I% MOD 6+145
  190     FOR L%=10TO18
  200       PRINTTAB(2,L%);CHR$K%;
  210       NEXT
  220       PRINTTAB(34,13);CHR$K%;TAB(34,14);CHR$K%;
  230       NEXT
  240   CLS
  250   PROCtitle
  260   PRINTTAB(15,23);CHR$133;CHR$136;"REPEAT ?";:ZZ$=G
ET$
  270   UNTIL (ZZ$="N" OR ZZ$="n")
  280 CLS
  290 VDU23,0,11,255,0,0,0,0,0,0
  300 END
  310 REM bar 1
  320 DATA&0300,-13,3,18,&0301,0,124,18,&0302,0,0,18,&030
3,0,0,6,3,-13,129,6,3,-13,137,6
  330 DATA&0300,-13,3,18,&0301,0,172,18,&0302,-13,109,18,
&0303,-13,145,6,3,-14,157,6,3,-14,149,6
  340 DATA&0300,-14,3,18,&0301,0,160,18,&0302,-14,117,18,
&0303,-14,149,6,3,-14,165,6,3,-14,157,6
  350 REM bar 2
```

```
 360 DATA&0300,-15,3,18,&0301,0,140,18,&0302,-15,129,18,
&0303,-15,157,6,3,-15,177,6,3,-15,173,6
 370 DATA&0300,-15,3,18,&0301,0,160,18,&0302,-15,117,18,
&0303,-15,177,6,3,-15,157,6,3,-15,145,6
 380 DATA&0300,-15,3,18,&0301,0,160,18,&0302,-15,97,18,&
0303,-15,129,6,3,-15,137,6,3,-15,145,6
 390 REM bar 3
 400 DATA&0300,-15,3,18,&0301,0,130,18,&0302,-15,89,12,&
0303,-15,149,6,3,-15,157,6
 410 DATA&0102,-15,101,6,&0103,-15,165,6,&0300,-15,3,18,
&0301,0,140,18,&0302,-15,109,18,&0303,-15,157,6,3,-14,149
,6,3,-14,145,6
 420 DATA&0300,-14,3,18,&0301,0,144,18,&0302,-14,117,12,
&0303,-14,137,6,3,-14,145,6
 430 DATA&0102,-14,97,6,&0103,-14,129,6
 440 REM bar 4
 450 DATA&0300,-13,3,18,&0301,0,152,18,&0302,-13,89,18,&
0303,-13,125,6,3,-13,129,6,3,-13,137,6
 460 DATA&0300,-13,3,18,&0301,0,168,18,&0302,-13,89,18,&
0303,-13,109,6,3,-13,125,6,3,-13,137,6
 470 DATA&0300,-13,3,18,&0301,0,152,18,&0302,-13,89,12,&
0303,-13,149,6,3,-13,145,6
 480 DATA&0102,-13,77,6,&0103,-13,137,6
 490 REM bar 5
 500 DATA&0300,-13,3,18,&0301,0,124,18,&0302,-13,81,18,&
0303,-13,145,6
 510 DATA3,-13,129,6,3,-13,137,6,&0300,-14,3,18,&0301,0,
112,18,&0302,-14,81,18,&0303,-14,145,6,3,-14,157,6,3,-14,
149,6
 520 DATA&0300,-15,3,18,&0301,0,144,18,&0302,-15,117,18,
&0303,-15,149,6,3,-15,165,6,3,-15,157,6
 530 REM bar 6
 540 DATA&0300,-15,3,18,&0301,0,140,18,&0302,-15,129,18,
&0303,-15,157,6,3,-15,177,6,3,-15,173,6
 550 DATA&0300,-15,3,18,&0301,0,160,18,&0302,-15,117,18,
&0303,-15,177,6,3,-15,157,6,3,-15,145,6
 560 DATA&0300,-15,3,18,&0301,0,152,18,&0302,-15,97,18,&
0303,-15,129,6,3,-15,137,6,3,-15,145,6
 570 REM bar 7
 580 DATA&0300,-15,3,18,&0301,0,144,18,&0302,-15,89,18,&
0303,-15,117,6,3,-15,157,6,3,-15,149,6
 590  DATA&0300,-15,3,18,&0301,0,147,18,&0302,-15,117,12
,&0303,-15,145,6,3,-14,137,6,&0102,-14,97,6,&0103,-14,129
,6
 600  DATA&0300,-14,3,18,&0301,0,152,18,&0302,-14,89,12,
&0303,-14,109,6,3,-14,129,6,&0102,-14,101,6,&0103,-14,125
,6
 610 REM bar 8
 620 DATA&0300,-14,3,54,&0301,0,124,54,&0302,-14,97,18,&
0303,-14,129,6,3,-14,145,6,3,-13,157,6
 630 DATA&0102,-13,109,18,&0103,-13,177,6,3,-13,157,6,3,
-13,145,6,&0102,0,0,18,&0103,-13,129,6,3,-13,145,6,3,-13,
109,6
 640 REM bar 9
 650 DATA&0300,-13,3,18,&0301,0,172,18,&0302,-15,97,36,&
0303,-13,129,54,&0100,-13,3,18,&0101,0,168,18,&0200,-13,3
,18,&0201,0,160,18,&0202,-15,101,18
 660 REM bar 10
 670 DATA&0300,-13,3,18,&0301,0,168,18,&0302,-15,109,36,
&0303,-13,137,18,&0200,-13,3,18,&0201,0,160,18,&0203,-13,
129,18,&0300,-13,3,18,&0301,0,152,18,&0302,-15,109,18,&03
03,-13,125,18
 680 REM bar 11
 690 DATA&0300,-13,3,18,&0301,0,160,18,&0302,-15,101,36,
&0303,-13,129,18,&0200,-13,3,18,&0201,0,168,18,&0203,-13,
109,42
 700 DATA&0200,-13,3,18,&0201,0,172,18,&0202,-15,97,18
 710 REM bar 12
 720 DATA&0200,-13,3,108,&0201,0,152,108,&0202,-15,89,18
,3,-13,109,6,3,-13,117,6
 730  DATA&0102,-15,89,18,&0103,-13,125,6,3,-13,137,6,3,
-13,129,6,&0102,0,0,72,&0103,-13,137,6,3,-13,149,6,3,-13,
145,6
 740 REM bar 13
 750 DATA3,-13,149,6,3,-14,137,6,3,-14,125,6,3,-14,109,6
,3,-14,125,6,3,-15,137,6,3,-15,149,6,3,-15,145,6,3,-15,13
7,6
 760 REM bar 14
 770 DATA&0300,-13,3,18,&0301,0,172,18,&0302,-13,97,36,&
0303,-13,145,6,3,-13,129,6,3,-14,137,6
 780 DATA&0200,-14,3,18,&0201,0,168,18,&0203,-14,145,6,3
,-14,157,6,3,-14,149,6
 790 DATA&0300,-14,3,18,&0301,0,160,18,&0302,-14,101,18,
&0303,-14,149,6,3,-14,165,6,3,-15,157,6
 800 REM bar 15
 810 DATA&0300,-15,3,36,&0301,0,140,36,&0302,-15,109,36,
&0303,-15,157,6,3,-15,177,6,3,-15,173,6,3,-15,177,6,3,-15
,157,6,3,-15,145,6
 820 DATA&0300,-15,3,18,&0301,0,160,18,&0302,-15,97,18,&
0303,-15,129,6,3,-15,137,6,3,-15,145,6
 830 REM bar 16
 840 DATA&0300,-15,3,18,&0301,0,144,18,&0302,-15,89,10,&
0303,-15,117,6,3,-15,157,6,2,-15,97,4,3,-15,149,6,2,-15,1
01,4
 850 DATA&0300,-15,3,36,&0301,0,152,36,&0302,-15,97,18,&
0303,-15,145,6,3,-15,137,6,3,-15,129,6
 860 DATA&0102,-15,89,18,&0103,-15,109,6,3,-14,129,6,3,-
14,125,6
 870 REM bar 17
 880 DATA&0300,-14,3,18,&0301,0,124,18,&0302,-14,81,36,&
0303,-14,129,6,3,-14,145,6,3,-14,137,6
 890 DATA&0200,-14,3,18,&0201,0,172,18,&0203,-14,145,6,3
,-13,157,6,3,-13,149,6
 900 DATA&0300,-13,3,18,&0301,0,160,18,&0302,-13,117,18,
&0303,-13,149,6,3,-13,165,6,3,-13,157,6
 910 REM bar 18
 920 DATA&0300,-13,3,18,&0301,0,140,18,&0302,-13,129,18,
&0303,-13,157,6,3,-13,177,6,3,-13,173,6
 930 DATA&0300,-13,3,18,&0301,0,160,18,&0302,-13,117,18,
&0303,-13,177,6,3,-13,157,6,3,-13,145,6
 940 DATA&0300,-13,3,18,&0301,0,160,18,&0302,-13,97,18,&
0303,-13,129,6,3,-13,137,6,3,-13,145,6
 950 REM bar 19
 960 DATA&0300,-13,3,18,&0301,0,130,18,&0302,-13,89,12,&
0303,-13,149,6,3,-13,157,6,&0102,-13,101,6,&0103,-13,165,
6
 970 DATA&0300,-13,3,18,&0301,0,140,18,&0302,-13,109,18,
&0303,-13,157,6,3,-12,149,6,3,-12,145,6
 980 DATA&0300,-12,3,18,&0301,0,144,18,&0302,-12,117,12,
&0303,-12,137,6,3,-12,145,6,&0102,-12,97,6,&0103,-12,129,
6
 990 REM bar 20
1000 DATA&0300,-11,3,18,&0301,0,152,18,&0302,-11,89,18,&
0303,-11,125,6,3,-11,129,6,3,-11,137,6
1010 DATA&0300,-11,3,18,&0301,0,168,18,&0302,-11,89,18,&
0303,-11,109,6,3,-11,125,6,3,-11,137,6
1020 DATA&0300,-11,3,18,&0301,0,152,18,&0302,-11,89,12,&
0303,-11,149,6,3,-11,145,6,&0102,-11,77,6,&0103,-11,137,6
1030 REM bar 21
1040 DATA&0300,-11,3,18,&0301,0,124,18,&0302,-11,81,18,&
0303,-11,145,6,3,-11,129,6,3,-12,137,6
1050 DATA&0300,-12,3,18,&0301,0,112,18,&0302,-12,81,18,&
0303,-12,145,6,3,-13,157,6,3,-13,149,6
1060 DATA&0300,-14,3,18,&0301,0,144,18,&0302,-14,117,18,
&0303,-14,149,6,3,-14,165,6,3,-15,157,6
1070 REM bar 22
1080 DATA&0300,-15,3,18,&0301,0,140,18,&0302,-15,129,18,
&0303,-15,157,6,3,-15,177,6,3,-15,173,6
1090 DATA&0300,-15,3,18,&0301,0,160,18,&0302,-15,117,18,
&0303,-15,177,6,3,-15,157,6,3,-15,145,6
1100  DATA&0300,-15,3,19,&0301,0,152,18,&0302,-15,97,19,
&0303,-15,129,6,3,-15,137,6,3,-15,145,7
1110 REM bar 23
1120 DATA&0300,-15,3,23,&0301,0,144,23,&0302,-15,89,23,&
0303,-15,117,7,3,-15,157,8,3,-14,149,8
1130 DATA&0300,-14,3,28,&0301,0,147,28,&0302,-14,117,18,
&0303,-14,145,9,3,-13,137,9,&0102,-13,97,10,&0103,-13,129
,10
1140 DATA&0300,-12,3,37,&0301,0,152,37,&0302,-12,89,23,&
0303,-12,109,11,3,-12,129,12,&0102,-11,101,14,&0103,-11,1
25,14
1150 REM bar 24
1160 DATA&0300,-11,3,72,&0301,0,124,72,&0302,-11,81,72,&
0303,-11,129,72
1170 REM Fine
1180 DEF PROCtitle
1190 CLS
1200 VDU23,0,11,0,0,0,0,0,0,0
1210 FOR J%=0TO24
1220   PRINTTAB(0,J%);CHR$135;CHR$157
1230   NEXT
1240 FOR J%=10TO18
1250   PRINTTAB(2,J%);CHR$145
1260   NEXT
1270 PRINTTAB(9,1);CHR$130;CHR$141;"Three Part Harmony"T
AB(9,2);CHR$130;CHR$141;"Three Part Harmony"
1280 PRINTTAB(15,3);CHR$130;CHR$141;"on the"TAB(15,4);CH
R$130;CHR$141;"on the"
1290 PRINTTAB(10,5);CHR$130;CHR$141;"BBC Microcomputer"T
AB(10,6);CHR$130;CHR$141;"BBC Microcomputer"
1300 G$=STRING$(20,"_")
1310 PRINTTAB(8,7);CHR$129;CHR$141;G$;TAB(8,8);CHR$129;C
HR$141;G$
1320 H$=CHR$224+STRING$(30,CHR$240)+CHR$176
1330 PRINTTAB(4,10);H$
1340 L$=CHR$234+STRING$(30,CHR$32)+CHR$181
1350 PRINTTAB(4,11);L$;TAB(4,12);L$
1360 PRINTTAB(4,13);CHR$234;CHR$134;" Jesu Joy of Man's
Desiring ";CHR$145;CHR$181
1370 PRINTTAB(4,14);CHR$234;CHR$134;TAB(15,14);"by J S B
ACH.";TAB(34,14);CHR$145;CHR$181
1380 PRINTTAB(4,15);L$;TAB(4,16);L$
1390 H$=CHR$162+STRING$(30,CHR$163)+CHR$161
1400 PRINTTAB(4,17);H$
1410 ENDPROC
```

Listing 1. Jesu Joy of Man's Desiring by J S Bach transcribed for the BBC Micro.

MEMORY SAVER #1

Save yourself memory by using the array of tips here.

If you have a program that contains some sizeable arrays, you may well be able to save quite a lot of valuable memory space by using a different type of array, the byte array. The byte array is a concept that has been carried over from the version of BASIC that Acorn wrote for their ATOM microcomputer: it makes use of the ? indirection operator, itself a concept borrowed from assembly language programming.

BYTEING UP YOUR RAM

A real array in BBC BASIC uses up five bytes of memory for every array element, so an array such as A(100), for example, would use up 500 bytes of memory. An integer array uses up four bytes for each element, so an array such as A%(100), for example, would use up 400 bytes. A byte array such as A 100, however, would only use up 100 bytes, one byte per element. There are drawbacks, of course, as an element of a byte array can only hold an integer value, and that value must lie between 0 and 255. The second drawback is the means of accessing a particular array element. Instead of writing:

```
A(52)   = 36.257
or:
A%(96)  = 2
```

for a byte array we would write:

```
A?14    = 5 or ?(A+14) =5
```

The second method of putting a value into a byte array element gives us a good clue as to how they work. A byte array is dimensioned using a slight variation of the usual DIM statement:

```
DIM A 100
```

The single spaces between DIM and A and between A and 100 are essential: no brackets must be placed around the dimension value. What this statement does is to set aside 101 consecutive bytes in memory and to give the first byte the label A. So the next byte is A+1 and to place a value into it we would use the ? indirection operator as follows:

```
?(A+1) = value (where value
         must be an integer
         between 0 and
         255)
```

The form A?1 is just an alternative to ?(A+1).

A CHANGE OF RANGE

If the numbers that you want to store in the byte array can only take values in a much more limited range than 0 to 255, say 0 to 31, then it is possible to put two or more successive numbers into one element of the byte array. In the example mentioned, eight numbers could be held in one byte. The first number would be entered with no modification, taking a value 0 to 31, the second number would have 32 added to it, giving a value between 32 and 63, and so on up to the eighth number which would have 7 * 32 = 224 added to it giving a value between 224 and 255. To retrieve the correct number from the byte array, we first of all need to note the relationship between our notional array of elements and the elements of the byte array. Continuing with our example, if we had 24 elements in our notional array, (0 to 23), these would be contained in the byte array elements:

A?0, notional array elements 0 to 7
A?1, notional array elements 8 to 15
A?2, notional array elements 16 to 23

To retrieve the Ith element of our notional array we would first have to find which byte array element it was contained in. We would do this with I DIV 8.

I	I DIV 8
0-7	0
8-15	1
16-23	2

We would next have to decide how many multiples of 32 to subtract from the value contained in A?(IDIV8). This number is given by I MOD 8; ie a number between 0 and 7. So to retrieve the Ith element of our notional array, we would use the formula:

```
Ith value = A?( I DIV 8) -
            32* (I MOD 8)
```

If you really need to save array space, and your array elements are positive integers less than 256, byte arrays offers you at least an 80% saving of memory over real arrays and a 75% saving over integer arrays. If the array elements have an even more limited range then you can obtain even greater memory savings. A universal routine to do that is left as an exercise for the reader, based on the guidance above!

HINTS AND TIPS

All sorts of useful hints and tips for use with your micro.

No matter how good the documentation supplied with your new computer there are always one or two new tricks to learn. We've collected some old favourites as well as a few new ones for you to try.

1. USEFUL MACHINE OPERATING SYSTEM COMMANDS

Machine operating system (MOS) commands are prefixed with an asterisk. When such a statement is encountered by the BASIC interpreter it is passed to each of the installed ROMs in turn until one of them recognises it, and deals with it, or it is not recognised by any of them and an error is generated. Most readers will only have the 16K MOS ROM or four 4K MOS EPROMs fitted. However, other ROMs that could be inserted in one of the three vacant slots are (or will be) the disc operating system, Econet, graphics extension, Pascal, LOGO, or the Acornsoft View word processing software. In addition, cartridge ROM packs will soon be available for insertion into the mysterious empty slot to the left of the keyboard.

The new **User Guide** lists all of the operating system commands available for use with the MOS ROM. Some of these are worth becoming familiar with, whereas many of them you will use rarely if ever. Ones that are particularly useful include:

*FX4, *FX18, *FX225, *FX226, *FX227 and *FX228.

In many games you will want to move a character, spaceship, gunsight or similar object about the screen. The most obvious keys to use for this purpose would be the four cursor keys, but in normal operation moving the cursor is their only function. The command *FX4,1 alters their function, however, so that they produce ASCII codes instead. These codes can be detected in the same way as for any other key, by use of GET, or INKEY. The COPY key is similarly affected: the ASCII codes produced are:

COPY	135
←	136
→	137
↓	138
↑	139

*FX4,0 will return the keys to their usual mode of operation. For MOS versions 1.0 onwards *FX4,2 turns these keys into five more function keys. The KEY numbers will then be:

COPY	11
←	12
→	13
↓	14
↑	15

The existing function keys are numbered 0 to 9. The missing function key 10 is, in fact, the Break key. Although its break function cannot be disabled, a string can be stored in it, just like with the other function keys. Command *FX18 (for MOS versions 1.0 onwards) will reset the function keys so that they no longer contain character strings.

There are some further modifications to the operation of the function keys which are available in version 1.0 onwards of the MOS. The first of these is with *FX225 which causes the function keys to generate ASCII codes instead of outputting strings of characters. Entering *FX225,X will cause key f0 to produce ASCII code X, key f1 to produce ASCII code X+1, and so on. *FX225,1 returns the keys to their normal function.

With versions of the MOS other than 0.1, pressing Shift at the same time as a function key causes them to produce ASCII codes in the range 128 to 137. In Teletext Mode 7, these ASCII codes are used for controlling the colour of alphanumeric characters and whether they flash or not.

From the previous paragraph you will recognise that the base value of these codes is 128. *FX226,X allows the user to alter this base value to X.

*FX227,X allows the user to alter the base value for the ASCII codes produced when the Control key is pressed simultaneously with the function keys. The normal base value is 144, which generates the Teletext control codes for the

Function key	ASCII code	Teletext control code
f0	128	no effect
f1	129	red alphanumeric
f2	130	green alphanumeric
f3	131	yellow alphanumeric
f4	132	blue alphanumeric
f5	133	magenta alphanumeric
f6	134	cyan alphanumeric
f7	135	white alphanumeric
f8	136	flashing characters
f9	137	remove flashing effect

graphics colours and flashing.

FX228,X will set a base value of X for the function key, Shift, Control keys combination. Normally, this combination produces nothing.

*OPTA,B

This MOS command is used with the cassette operating system, and is primarily used to control the action taken by the computer when an error is detected during a cassette file operation. *OPT without any values for a and b will reset all the options to their default values. Page 398 of the new **User Guide** supplies the details of what happens for values of A between 1 and 3, and for values of B between 0 and 2. One combination which is given only scant mention is *OPT1,2. The **User Guide** states that this combination will yield "detailed information including load and execution addresses". The reason why this deserves more emphasis lies in the value of knowing these load and execution addresses. When you buy a piece of commercial software on tape, you have a tape which has to be replayed each time you want to run the program you have bought. The tape will wear with use and could even be damaged. In either case you may have a tape which is no longer usable. To avoid having to buy a second tape, it would be prudent to take a copy from the original tape, and only to use the copy. Should any disaster befall the copy, another can be taken from the original master tape. This policy of keeping a master and using copies is common commercial computing practice.

If the program you have bought is written entirely in BASIC, then there is usually no problem in taking a copy: just use SAVE in the normal way. When a program is SAVEd, its load and execution addresses are recorded as well. The load address is the memory location in RAM at which the computer will begin storing the program. The execution address is the address to which the computer will jump in order to begin running a program. For a BASIC program, this is the hexadecimal address &801F, which is the language initialisation address inside the BASIC interpreter ROM. When a BASIC program is LOADed back into memory from tape, it will automatically be stored at the lowest available address in RAM. For most users this will be &0E00: however, if you have Econet, disc operating system or Telesoftware ROMs inside your machine, this value will be higher. The address can be determined by printing the value of PAGE (see page 317 of the new **User Guide**). Occasionally, though, programs will need to be loaded back into addresses other than &0E00. In this case the instructions with the program will tell you to alter the value of PAGE before CHAINing the program in. You must set PAGE to this value when SAVEing the program as well.

Difficulties start to appear when you want to SAVE a machine code program, or one that is mixed BASIC and machine code. If you suspect that the program that you wish to copy is of this type then you need to use the *OPT1,2 command. If the program instructions tell you to CHAIN the program, then type:

```
*OPT1,2
LOAD "FLANGE"
```

assuming the program is called FLANGE.

If the instructions tell you to *RUN the program, then type:

```
*OPT1,2
*LOAD "FLANGE"
```

note that, in this case, we have to use *LOAD. In both cases the resulting messages will look like this:

Name	Length	Load	Execution
FLANGE 26	2670	00000E00	00000E00

All the numbers are in hexadecimal. The one labelled length is the actual length of the program in bytes. The load and execution addresses are eight hexadecimal digits long and will always begin with four zeros. These are, in fact, 32 bit addresses which the BBC Microcomputer uses in its filing systems to ensure compatibility with 16 and 32 bit second processor add-ons. We only need to know the last four digits.

To SAVE this program that has been loaded we need to use the command:

*SAVE"name"load address + length execution address,

ie: *SAVE"FLANGE" 0E00 + 2670 0E00, for our example.

It must be emphasised that unauthorised copying of software is illegal and the above information is intended to show you how to provide back-up copies of commercial software **FOR YOUR OWN PERSONAL USE ONLY.**

*TV

Many domestic televisions are set up so that not all of the transmitted picture can be seen. This is done to avoid showing blank margins at the sides and bottom of the picture, and the Ceefax and Oracle signals at the top of the picture (in the top two lines). The BBC Microcomputer makes very full use of the available picture, and this often means that either the top or bottom parts of the computer generated picture cannot be seen. *TV255 will move the picture down by one character row, *TV254 will move it down two characters, etc. The corresponding commands to move the picture up are *TV0, *TV1, etc. These commands only take effect after a change of mode.

You can also write *TV255,1 where the second parameter, 1, causes the interlace to be turned off. Interlace is the means whereby the TV picture is scanned twice in each frame to give a denser picture, and two sets of scan lines are interlaced with each other. It often causes a flickering picture. In Mode 7 you are stuck with interlace, but in all the other modes you can turn it off to give a steady picture. *TV255,0 would restore the interlace again.

*FX15,1

When you are using INKEY or INKEY$ to wait for a key being pressed at the keyboard, if the keyboard buffer has not been emptied, then whatever is the next character in the buffer will be detected. This could well be a spurious keyboard entry which you do not want the program to detect. If you put the command *FX15,1 on the previous line to the INKEY statement, the keyboard buffer will be automatically flushed.

2. AVOIDING TWO CASSETTE FILING SYSTEM ERRORS

There are two important errors present in the MOS version 0.1: one makes SAVEing programs at 1200 baud unreliable, and the other makes writing to files using the PRINT # statement almost impossible. The first error can be avoided by recording programs at 300 baud only. The second can be avoided by using BPUT # to write files instead of PRINT #. If either of these options is unacceptable to you, then you will find Listing 1 essential. It is reproduced here by kind permission of Richard Russell of the BBC, who was involved in preparing the specification of the BBC Microcomputer.

It should be SAVEd itself, after first typing it in and running it, and then CHAINed in every time you use the computer. It will survive a single press of the Break key, but not two presses in sequence, which cause a hard reset. When the program is run it loads a machine code patch into locations &DD0 to &DFF, and can then be overwritten with NEW.

This program is not needed, and should NOT be used with, versions of the MOS other than version 0.1.

```
>L.
  100 REM  Patch for 2 bugs in cassette filing
  101 REM  system. Only needed for v0.1 of MOS
  102 REM  DO NOT USE WITH OTHER VERSIONS
  130 FORPASS=0TO1:P%=&DD0:GOSUB180:NEXT
  140 ?&218=FIX1:?&219=FIX1 DIV 256
  150 ?&20A=FIX2:?&20B=FIX2 DIV 256
  160 *KEY10 ?&218=&D0:?&219=&D:?&20A=&D6:?&20B=&D|M
  170 END
  180 [OPT PASS*2
  190 .FIX1 PHA:JSR &F21:PLA:RTS
  200 .FIX2 CMP #&91:BNE GO:CPX #0:BNE GO
  210 TSX:LDA &102,X:CMP #&F7:BEQ TRAP
  220 LDX #0:.TX LDA #&91:STA &FE09:RTS
  230 .GO JMP (&DB60)
  240 .TRAP PLA:PLA
  250 JSR &F9D8:JSR &FB7B
  260 JSR TX:JMP &F7FB
  270 ]RETURN
```

Listing 1. Richard Russell's Bugpatch.

3. FURTHER TELETEXT SURPRISES

When writing a program using Teletext characters in Mode 7, it can be quite irksome to have to type CHR$X, whenever you want to put a control code into a PRINT statement. Since all control codes have three digits, you have to type in seven characters for each control code. There is a way that you can achieve the same end and only press one key! Try the following line:

```
PRINT CHR$130;"Hello"
```

The message "Hello" will appear on the next line in green. Now try the following two lines:

```
*KEY0 |!|B
PRINT "  Hello"
      ▲
Press f0 here
```

The message "Hello" will again appear on the next line in green. It is also green in the PRINT statement. What you have done is to program the control code for green alphanumeric characters into function key 0. The letters corresponding to each control code (such as B in the example above) are given in the table shown below.

As you can see use of this method of putting control codes on the screen gives colourful text. Each control code leaves a blank space in the listing (remember that they must be placed *inside* the inverted commas), and they may be copied using the COPY key in the usual way.

One control code in the table that you may well not have come across is hold graphics, letter ∧, otherwise CHR$158. This is a very valuable code when you are attempting to draw complex coloured designs involving graphics shapes. Remember that whenever you put a control code in a PRINT

Letter	Effect generated	Letter	Effect generated
A	red alphanumerics	Q	red graphics
B	green alphanumerics	R	green graphics
C	yellow alphanumerics	S	yellow graphics
D	blue alphanumerics	T	blue graphics
E	magenta alphanumerics	U	magenta graphics
F	cyan alphanumerics	V	cyan graphics
G	white alphanumerics	W	white graphics
H	flashing on	X	conceal display
I	flashing off	Y	continuous graphics
L	normal height characters	Z	separated graphics
M	double height characters	\	black background
		]	new background
		∧	hold graphics
		—	release graphics

The Teletext control codes available in Mode 7.

statement there will be a gap left on the screen in the background colour. Suppose that we wanted to have four solid blocks of green followed by four solid blocks of yellow on a black background. The character that gives a solid block in graphics mode is ASCII code 255. The code for graphics green is 146, and the code for graphics yellow is 147. Let's try:

```
PRINT CHR$146;CHR$255;CHR$255;CHR$255
      ;CHR$255;CHR$147;CHR$255;CHR$255
      ;CHR$255;CHR$255
```

This produces the green and yellow blocks but they are separated by a space caused by the control code to give yellow graphics (CHR$147). Now let's try this:

```
PRINT CHR$146;CHR$255;CHR$255;CHR$158
      ;CHR$147;CHR$255;CHR$255;CHR$255
      ;CHR$255
```

The green and yellow blocks are now touching. Control code 158 causes the last graphics character to be repeated in the spaces caused by control codes. So the two spaces which would be caused by CHR$158 itself and CHR$147 have the green graphics block repeated in them. Its effect lasts for the whole line and, in each case where it operates, it is the last graphics character before the respective control code that is repeated. CHR$159 cancels its effect.

WARNING

If you use the function key method, described above, of placing hidden control codes in your program, you may find that they affect your printer, and you are unable to list it!

4. THREE (OR MORE) COLOURS IN MODE 4

Try running the following short program:

```
 10  MODE 4
 20  VDU 19,0,3,0,0,0
 30  VDU 19,1,4,0,0,0
 40  MOVE 0,0:MOVE 600,0
 50  PLOT 85,0,300
 60  PLOT 85,600,300
 70  FOR I%=304 TO 600 STEP 8
 80  MOVE 0,I%
 90  PLOT 21,600,I%
100  MOVE 4,I%+4
110  PLOT 21,600,I%+4
120  NEXT
```

You will see a yellow background with a blue rectangle in the lower left of the screen, with a grey rectangle on top of it. The grey rectangle may have a red and green interference pattern moving across it, caused by the limitations of the PAL encoding system. If you add line 5:

```
5  *TV255,1
```

the grey may be more recognisable (RGB monitor owners will not need this program modification). What this does is, firstly, to select Mode 4 (a two-colour mode) and then set the background to yellow and the foreground to blue (lines 10 to 30). Then a blue rectangle is drawn with two triangles (lines 40 to 60). Lines 70 to 120 draw another rectangle on top of the blue one by plotting dotted lines. Alternate dotted lines are plotted displaced from each other by one pixel to avoid a vertical striped pattern. The dotted lines are alternate blue and yellow pixels, which the eye combines to give grey — three colours in a two colour mode.

The program can be added to, to cycle through all the possible colour combinations, and the technique could be modified (by plotting individual points rather than dotted lines) to give, say, two blue points alternating with one yellow one. This would give yet another colour. There is plenty of potential here for experimentation, so happy dithering (that is what the method is called!).

5. NEGATIVE INKEY FUNCTION

The new **User Guide** gives on page 275 a table of negative INKEY values, ie numbers to be used as the INKEY argument — INKEY (−m). The Table, reproduced below, shows a unique negative number against every key on the keyboard. Stated simply, what the use of negative INKEY values gives is a means of detecting when any key is pressed. Not only that but keys held down simultaneously can all be detected, something which is not possible with positive INKEY values. Also the use of INKEY(−m) is independent of the keyboard buffer, so there is no need to keep clearing it with *FX15,1 (see earlier note).

Lastly, keys which cannot normally be detected such as Control, Shift lock, Delete and the function keys, can all be detected with negative INKEY.

6. BETTER PROGRAMS WITH REPEAT . . . UNTIL

This Pascal-like feature of BBC BASIC is well worth becoming familiar with. Two examples of its use are shown in the drawing listing in the article entitled Multiple Graphics Demo. In this program there is a short main section (lines 570 — 620) which has to be cycled indefinitely. Instead of using a GOTO statement:

```
100 REM beginning of main
      program
         ●
         ●
         ●
     statements
         ●
         ●
         ●
500 GOTO 100
```

It is more easy to follow if we use a REPEAT...UNTIL loop:

```
100 REPEAT
         ●
         ●
         ●
   program statements
         ●
         ●
         ●
500 UNTIL FALSE
```

The condition in line 500 will never be met, so the program cycles indefinitely.

The second use, which occurs a number of times in the multiplication listing in the Multi-test article, is to continue performing a certain action until a particular condition is met, such as Fire button number two being pressed:

```
100 REPEAT
        •
        •
        •
    statements
        •
        •
        •
300 fire% = ADVAL(0) AND 3
        •
        •
        •
    statements
500 UNTIL (fire% = 2)
```

This same idea can be used to continue reading the keyboard until a certain key is pressed, say Y or y for a 'yes' reply:

```
100 REPEAT
110 G$=GET$
120 UNTIL (G$="Y" OR
    G$="y")
```

Selection from menus is an obvious use for this method (see below).

7. USE OF MENUS

In writing a program for someone else to use it is particularly important to make the program as easy to use as possible — user-friendly in the jargon. One way of doing this is by the use of menus, that is a screen with a number of labelled choices for the user to select from. All the user has to do to continue is to type the one letter or number corresponding to his or her choice. If the choices are labelled with one character then the user does not even have to press Return, because the programmer can use GET or GET$ to detect the key pressed, rather than INPUT. A series of menus can take the user a long way into a complicated program in easy stages, since the menu choices themselves remind him or her of the options from which he or she can select. This is the concept behind the use of the massive Prestel database.

With the BBC Microcomputer, the REPEAT... UNTIL structure allows an elegant solution to reading the user's menu choice and discarding incorrect key depressions. Suppose the choices are labelled 1 to 6. The ASCII code for 0 is 48, for 1 it is 49, and so on.

The following statements will do the job.

```
100  REPEAT
110  Z%=GET
120  Z%=Z%-48
130  UNTIL (Z%>0 AND Z%<7)
```

Line 680 of that multiplication tables listing which reads:

```
680  F%=GET:F%=F%-48:IF F%<1 OR F%>3
     GOTO 680
```

could be altered to read:

```
680  REPEAT:F%=GET:F%=F%-48:UNTIL(F%>0
     AND F%<4)
```

Mike James

SOUND AND VISION

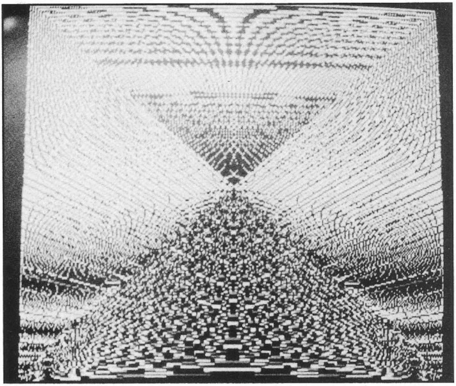
The fundamentals of the BBC Micro's SOUND facility and its powerful graphics laid bare.

The BBC Micro, like many others, uses memory mapped graphics but it uses it in a very different way. Most machines that generate their own video output set aside an area of memory where the ASCII (or similar) codes of the characters to be displayed are stored. As each character's code can fit into eight bits, one memory location is used for every possible display position on the screen. For example, if you have a screen of 40 characters by 20 lines then you need 800 (ie 40 by 20) memory locations.

The way in which these memory locations are made to correspond to positions on the screen varies from machine to machine. It could be that the first memory location corresponds to the character displayed in the top left-hand corner of the screen; subsequent memory locations corresponding to screen locations to the right of the first until the end of the line is reached, when a new line is started at the far left-hand side again (see Fig. 1). The way the memory is associated with the different display positions on the screen is know as the 'screen memory map'. Obviously if you know the screen memory map for a particular machine then you can write programs to change the screen display by going straight to the correct memory location instead of using a PRINT or PLOT statement. This can be the quickest, and sometimes the simplest, way of changing the screen and is often the only way of producing good moving graphics.

As mentioned earlier, the BBC Micro uses a very different method of producing a memory mapped screen. Instead of storing the ASCII code of the character to be displayed, the BBC Micro stores a bit pattern corresponding to the ***shape*** of the character. To make this clear it is worth considering the way other micros convert the ASCII code stored at each memory location into a character displayed on the screen.

A TV picture is built up



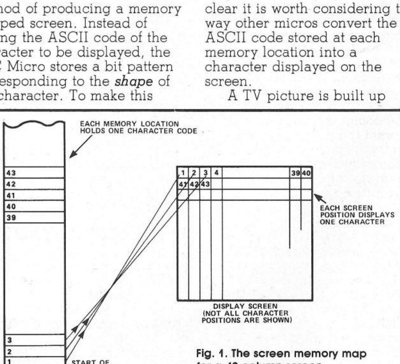


Fig. 1. The screen memory map for a 40 column screen.

from a series of lines and each row of characters requires a number of lines. Each character is formed from a number of dots which may be turned on or off. In this respect the BBC Micro is no different from the rest and uses eight lines of eight dots for each character (see Fig. 2). However, other micros produce this pattern of dots on the screen by using an extra chunk of memory that is accessible only to the video display electronics. This extra memory is normally called a 'character generator' but it is nothing more than a ROM (Read Only Memory) containing the information concerning which dots should be off or on to form the image of a particular character. It is because this ROM memory is available only to the display electronics that it is normally not counted as part of the computer's memory. If you want to know how much memory is involved in a character generator all you have to do is multiply the total number of dots used to make up a character by the total number of possible characters and divide by eight (this is because the ROM has to store the dot pattern of every character that can be displayed and each dot requires one bit). For the eight by eight array of dots used by the BBC Micro, a ROM to generate the character set would have to be 2K in size.

The usual method of displaying characters on a screen using a character generator is simply to use the ASCII code stored in the computer's memory as an 'address' to select the location in the ROM that stores the dot pattern for that character (see Fig. 3). Instead of using this traditional approach to video display, the BBC Micro dispenses with a character generator ROM and stores the dot pattern of the character to be displayed in RAM. The disadvantage of this method is that each screen location needs enough RAM to store all the dots for a single character — in the case of the BBC Micro this amounts to eight bytes per screen location. This means that in MODE 4, for example, with 32 lines of 40 characters, the total RAM required is 32 times 40 times 8, ie 10K, and all this RAM is taken from the user RAM that you might have used to store programs and data.

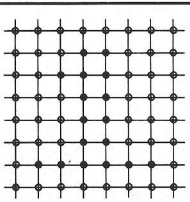


Fig. 2. An eight by eight dot matrix showing the character '1'.

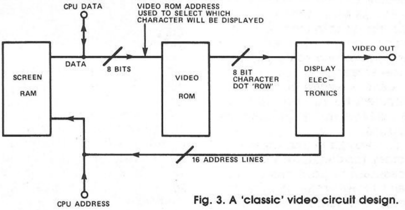


Fig. 3. A 'classic' video circuit design.

In other words, the BBC Micro uses eight times the amount of screen RAM for a given screen size — because it stores an eight bit code instead. The method the BBC Micro uses is often called a 'bit mapped display' because every bit in the screen RAM corresponds to a dot on the video screen.

WHAT ADVANTAGE?

Given the extra memory that the BBC Micro has to use to produce its display you might be wondering what the advantages are. The main advantage is that you can produce high resolution graphics and text characters using the same hardware. Since every dot on the screen corresponds to a bit in the memory location, instead of storing the dot pattern corresponding to a character, you can change individual bits in the memory to produce lines and other shapes. Also, because the same basic method is used to display characters and to produce high resolution graphics, you can mix both ***anywhere*** on the screen. A second advantage is that the character set is not restricted to whatever is stored in the character generator ROM thus allowing you to define new characters.

These two advantages give the BBC Micro a freedom in handling both graphics and characters difficult to match using any other method. For comparison, the Apple uses a bit mapped display for its high resolution graphics but uses a standard character generator for its text modes and so has difficulty in freely mixing text and graphics without extra software (shape tables). On the other hand, the PET uses a character generator for both text and graphics and so can mix them freely but the range of graphics is limited to those already defined in its ROM.

What all this means to the programmer is that, unlike machines such as the PET where POKEing a byte into a memory location causes a complete character to appear on the screen, POKEing a byte to the BBC Micro's display memory causes a pattern of dots on a single line to appear. All that we need to know now is how each memory location corresponds to a screen position and the best way to discover this is via a small test program.

If we start at the lowest screen address and POKE a byte consisting of all 'ones' then a short line of dots will appear somewhere on the screen. If the BBC Micro uses a fairly normal screen memory map, the line should appear in either the top left-hand or bottom right-hand corner. Before we can try this little experiment, however, it is necessary to look at the way the BBC BASIC allows memory to be POKEd. Although I have been using the term POKE to describe storing some data in a given memory location, this is not a term that BBC BASIC uses. To POKE a byte into memory location at 'address', the BBC Micro uses:

```
?address=byte
```

and the '?' isn't a mistake. It means 'treat the number following as an address' (familiar ground for ATOM users but a little strange to the rest of us). The address and byte used in this expression can be variables or constants. If constants are used then it is useful to know that you can specify a hexadecimal constant by using '&'. For example &01 is 1, &0F is 15 and so on.

PRACTICAL EXPERIMENTS

Now we know how to alter a memory location, we can resume the examination of the screen. If you run the following program.

```
10  MODE 4
20  ?HIMEM=&FF
30  GOTO 20
```

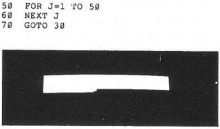
You should now see a short horizontal line in the top left-hand corner. If you don't then it's possible that it's just off part of the screen your TV displays and a slight adjustment of the controls should make the line visible. The program works by first selecting MODE 4 and then (in line 20) storing the Hex value FF in the memory location whose address is stored in HIMEM. The variable HIMEM stores the address of the first screen location in any mode and FF in binary is eight 1s — so producing a row of eight dots.

We now know that the first (lowest) screen address corresponds to the top left-hand corner. To find out how the rest of the screen memory map goes, try the following program.

```
10  MODE 4
20  FOR I=0 TO 7
30  ?(HIMEM+I)=&FF
40  NEXT I
50  GOTO 50
```

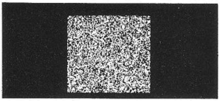
This stores the Hex value FF in eight consecutive memory locations. What is surprising about the result of this program is that instead of producing a thin line eight characters long across the top of the screen, it actually displays a solid block about the same size as a normal character. The screen memory map of the the BBC Micro is such that the first eight memory locations form the dot matrix for the first character. The next eight form the dot matrix for the character to the right of the first and so on to the end of a line. To see the screen memory map in action, try the following:

```
10  MODE 4
20  I=0
30  ?(HIMEM+I)=&FF
40  I=I+1
50  FOR J=1 TO 50
60  NEXT J
70  GOTO 30
```



You should see the screen fill up character position by character position. You can use this program to explore the possibilities of POKEing graphics data directly onto the screen. For example, illustrating that things other than solid lines can be POKEd, try altering line 30 to:

```
30  ?(HIMEM+I)=RND(255)
```



and removing the delay loop formed by lines 50 and 60.

Using this information we can work out a simple equation that will give the address of any screen location:

```
address=HIMEM+(X+Y*40)*8+N
```

This expression gives the address of the Nth line making up the character at the screen location X,Y (N,X and Y all start from zero in the top left-hand corner).

COLOURFUL EXPANSION

The reason why the previous section considered the memory map for MODE 4 is that it is a two-colour mode; this means that each point in a can only be one of two fixed colours and so can be represented by a single bit. If a MODE uses more than two colours — 16 for example — then you need more than one bit to represent each point on the screen. It's a little difficult to explain how many you need in general but two bits can represent up to four colours, three can represent eight and four can represent 16. The question is — how are the extra bits organised in the memory map of the other modes?

The answer is that the fundamental memory map outlined for MODE 4 is used for all the other modes except that each point on the screen now corresponds to a small group of bits in each memory location. For example, in MODE 4 a memory location holding eight bits gave rise to eight dots but in MODE 5 (a four-colour mode), the same memory location only gives rise to *four* dots. In this case each group of two bits determines which of the four colours a point will be (see Fig. 4).

The best way to investigate the memory maps of the other graphics modes is to use the programs given in the last section but change line 10 to give the required mode. In MODE 5, as each block of eight memory locations now corresponds to only eight rows of four dots and each character still needs eight rows of eight dots to be displayed. It should be obvious that the storage of a single character involves two such blocks.

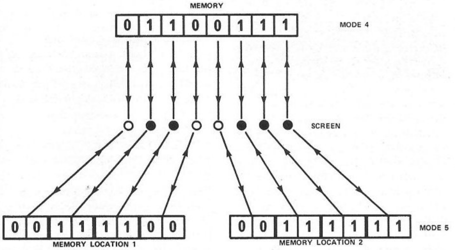


Fig. 4. The correspondence between Modes 4 and 5 for eight dots on the screen.

If all this seems a little complicated then all I can say is that compared to the way other computers work IT IS but, if you want to have the sort of freedom of action that the BBC Micro allows, there is no other way of doing it! In practice, the use of direct memory mapped graphics is limited to either MODE 4 (where it is easy) or involves assembler (where everything is more difficult!!). Seriously though, POKEing the screen is not as useful on the BBC Micro as on other machines — partly because it is more difficult except in two-colour modes and partly because the BASIC provides all sorts of features that make it unnecessary. What is more important though is that a knowledge of the screen memory map allows you to find out quickly what is stored at any screen location.

PEEKING THE SCREEN

This brings us to the topic of PEEKing the screen to see which character is stored at any particular location. This is easy on machines such as the PET — all you have to do is to PEEK the screen location and this returns the ASCII code of the character stored at that position. For the BBC Micro things are not quite as easy.

The first problem is that PEEKing a screen location in a two-colour mode returns the dot pattern of a row of the character stored at the location. This is not as useful as the ASCII code because in general it is not enough to identify the character — for example, it is possible for two characters to have the same dot pattern in every row except one! The second problem is that for the modes which use more than two colours, even a single row of dots from a character is difficult to obtain without a number of PEEKs and quite a bit of logic.

This might make you think that screen PEEKs are not worth the trouble on the BBC machine. However, for MODE 4 things are easier than they look. The general problem of deciding what character is stored at a screen location is difficult even in MODE 4 but in most graphics-based applications this is more than we want to do. Instead of identifying which character from the set of all possible characters is present, it is usually enough to decide which one of two or three characters is there. For example, if you are using 'O' to represent one type of player and 'X' to represent another then we only have to discover if the character stored at a location is one of blank, O or X. This is a much easier problem as it should be possible to find a row of dots different in each character. If this ***is*** possible then you can tell the three characters apart by PEEKing that one row! In the case of blank, X and O, any row will distinguish them but row three corresponds to 0, 24 and 102 respectively.

The BBC Micro uses the '?' instead of PEEK as well as POKE. If you want to PEEK a particular screen location then all you have to do is:

```
?address
```

This will return the contents of the memory location at 'address'. For example:

```
A=?2000
```

stores the contents of memory location 2000 in A. Notice that the '?' represents a POKE if it is on the left of an equals sign and a PEEK if on the right. Now that we know how to PEEK a memory location and we know the screen memory map for MODE 4, we can write a function that will return the contents of a particular row of a screen location:

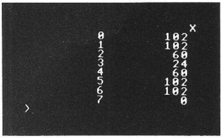
```
100  DEF FNS(X,Y,N)=HIMEM+(X+Y*40)
     *8+N
```

FNS will return the address of the screen location corresponding to character position X,Y and the Nth row of the character.

To give an example of how to use FNS, the program below will print a character on the screen at 20, 10 and will then print the value of the dot pattern making up each row of the character.

```
10  INPUT A$
20  MODE 4
```

```
30   PRINT TAB(20,10);A$
40   FOR N=0 TO 7
50   PRINT N,?FNS(20,10,N)
60   NEXT N
70   END
100  DEF FNS(X,Y,N)=HIMEM+
     (X+40*Y)*8+N
```



This program can also be used to discover how any character is made up — it was used to find out the values of the third row of blank, X and O in the previous example. In practice, the function FNS would be used in IF statements to decide what should be done according to what is stored at a particular location.

USING THE MOS TO PEEK

There is a way of discovering the ASCII code of the character stored at a screen location but it needs a USR call to the MOS (Version 1 and later revisions only) and it is slow (about 120 milli seconds per character). However, if speed is not important then you can use the following function:

```
100  DEF FNASC(X,Y)
110  LOCAL C
120  X%=X
130  Y%=Y
140  A%=135
150  C=USR(&FFF4)
160  C=C AND &FFFF
170  C=C DIV &100
180  =C
```

FNASC(X,Y) will return the ASCII code at screen location X,Y and CHR$(FNASC(X,Y)) will supply the character itself.

The operating system call used in the above function (ie USR(&FFF4)) works by reading the screen memory, assembling the eight bytes representing the character's dot pattern (easy in two-colour modes, not so easy in the rest) and then searches an area of memory in the operating system that is used to generate the dot pattern in the first place. This area of memory is the BBC Micro's equivalent of a character generator. When you PRINT a character to the screen this area of memory — the character table — supplies the dot pattern for the character. This is fast because the table is organised so that the ASCII code of the character leads straight to the correct pattern. However, going back from the pattern to the ASCII code is slower because it involves finding a match for eight bytes somewhere in the table!

THE TROUBLE WITH SCROLL

There is one feature of the BBC Micro that is very surprising and can make use of the screen address map very difficult. When you carry out a MODE command, the screen address map is set up as we have discussed and remains unaltered during the running of a program unless that program prints something that causes the screen to scroll. The action of scrolling is such a common sight on VDUs and computers that it is rare to give it a second thought. However, if you try to write a program from first principles to scroll an entire screen, you will realise what a time-consuming manoeuvre it is. Each text line of the screen must be moved up by one line. The bottom line is cleared and the top line is lost.

In the BBC Micro's case, this screen shift, if done by software for MODE 4, would need 10K of storage to be rearranged — slow to say the least! To overcome this speed problem, scrolling is carried out by hardware which in effect alters the screen memory map so that the memory locations correspond to screen positions one higher. The memory corresponding to the old top line is cleared and is made to correspond to the new bottom line — ie following a single scroll, POKEing data into memory that was the top line produces output on the bottom line. Of course this 're-mapping' of the screen makes a nonsense of the screen mapping functions given earlier! However, the solution is simple — either avoid scrolling the screen following a MODE command or adjust the functions to take account of any scrolling.

To take account of scrolling, it is necessary to keep a count of the number of times the screen has scrolled since the last MODE command. If the scroll count is kept in SC then the following version of FNS will work (for MODE 4):

```
100  DEF FNS(X,Y,N)
110  YT=Y+SC
120  YT=YT-INT(ABS(YT)/32)*32
130  =HIMEM+(X+Y*40)*8+N
```

Notice that YT and SC are global variables and are thus accessible to the main program. Luckily, it is not often that the need to scroll the screen occurs in the same situation as the need to use POKE or PEEK graphics.

A lot of fun using the computers comes from exploring unknown territory — hold on a minute, could this be why Acorn were so long in producing the final version of the BBC Micro's manual!!

Anyway, the territory which forms the subject of this section of the article is so vast that it alone could keep a BBC Micro owner supplied with interesting projects for many months. For, although the BBC Micro has only a single sound generator chip with one noise channel and three tone channels, the software built into the BASIC to handle it makes it more powerful than the hardware specification might lead you to believe. Indeed, it is so powerful and flexible that this article can cover only a fraction of the possiblities!

SOUNDINGS

Before launching into details of how the BBC Micro's sound generator can be used, it is worth taking an overview of the sort of things it can do. There are three tone generators which can be used to produce either single notes or up to three-note chords. There is, in addition, a single noise channel which can produce eight different effects.

This fairly simple hardware is controlled using two extensions to BASIC — SOUND and ENVELOPE. The SOUND command is the only one of the pair which actually causes anything to come out of the tiny speaker just above the keyboard. Among other things, it controls pitch, amplitude and duration of the notes produced. The ENVELOPE command is used to change the characteristics of the notes produced by the SOUND command. Used without the ENVELOPE command, SOUND produces a more or less pure tone with a given frequency, which is fine for most applications — eg beeps during games or playing simple tunes. However, if you want to try to produce more complicated sounds then you have to use the ENVELOPE command to alter the basic sound produced.

There are two general reasons for wanting to produce more complex noises — either you are interested in music and making your BBC Micro sound like a piano, flute, organ, guitar . . . or you want to make especially convincing sound effects such as a police siren, gun shot, etc.

The study of the BBC Micro's sound capabilities, therefore, falls into these two catagories — music and sound effects.

MAKING MUSIC

There are three levels of difficulty involved in making music with the BBC Micro:

1) playing simple tunes,
2) playing music with three-part harmony, and
3) 'synthesising' the sound of

of other instruments.

The first two involve the use of the SOUND command only but the last one also requires a mastery of the ENVELOPE command and, sadly, falls outside the scope of this article. To get very far with either of the three you also need a reasonable understanding of music, but if you feel a little unsure about this then programming sound is a very enjoyable way to learn.

The subject of sound effects is much more limited because all that we are trying to do is to compile a catalogue of 'recipes' to make a few standard noises. However, there are two ways of approaching sound effects: you can either use the SOUND command to control the noise channel or you can use the ENVELOPE command to define basic sounds. With the latter you can produce quite remarkable effects but there's still a great deal of scope for producing a wide variety of noises using the SOUND command.

The rest of this article will concentrate on the use of the SOUND command and will try to convey some of the flavour of the uses of the BBC Micro's sound generator. (The ENVELOPE command is so complex and versatile that it demands an article to itself!) To get us started a brief resume of the SOUND command seems appropriate.

SOUND

The SOUND command has the general form:

```
SOUND C,A,P,D
```

where C controls which channel (0, 1, 2 or 3) produces the sound (channel 0 is the noise channel). A controls the volume and ranges from 0 (silence) to − 15 (loudest); P controls the pitch of the note and ranges from 0 (lowest pitch) to 255 (highest) and D controls the duration of the note and ranges from 1 to 255 in twentieths of a second. There are various extra meanings associated with the parameters C and A. Positive values of A in the range of 1 to 4 cause the pitch and volume of the note to be controlled by the parameters of the ENVELOPE command. The channel parameter C is, in fact, quite complicated and is best thought of as a four-digit hexadecimal number:

```
&HSFN
```

where each of the letters stands for a digit which controls a different aspect of sound production. What exactly each of them does is better left until later except to say that N is the channel number as described earlier.

PROGRAMMING TUNES

Programming tunes is simply a matter of converting notes into numbers. This is easy once you know that middle C corresponds to a value of 53 and going up or down by a whole tone corresponds to adding or subtracting 8. The only thing that you have to be careful to remember is that there isn't always a whole tone between two notes. For example, between the notes of C and D there is a whole tone but between E and F there is only a semitone. The pattern of tones and semitones from C to C, an octave above, is:

```
C - D - E - F - G - A - B - C
  T   T   S   T   T   T   S
```

which is easy to remember because it's the same as the pattern of white and black notes on the piano. Obviously sharps and flats can be produced by adding or subtracting 4. So, you can produce the full chromatic scale by:

```
10  FOR P=53 TO 97 STEP 4
20  SOUND 1,-15,P,10
30  NEXT P
```

This short program can also be used to demonstrate a unique feature of the BBC Micro. If you add line 15:

```
15  PRINT P
```

you will discover that the numbers are printed on the screen and the program finishes but the sound keeps on coming. The reason for this remarkable behaviour is that the BBC Micro maintains a queue of sounds which are produced one after the other as soon as the current sound is completed. The sound queue is processed independently of any BASIC program that is running and each SOUND statement simply adds a note to the end of the queue. This means that a BASIC program isn't held up for the duration of each note. The only time that this fails is when the queue becomes full and a SOUND statement tries to add another note to it. The result is that the program then has to wait until the end of the currently sounding note when the queue is reduced by one and the SOUND statement can add its note.

To make a tune recognisable, not only must each note be at the right pitch, but each note must last for the correct time. The normal system of musical notation is based on repeatedly dividing a time interval by two to obtain shorter notes, so it is a good idea to include a variable in all music programs which set the length of the fundamental unit of time. As an example of programming a simple tune consider the first few notes of 'Hearts of Oak' (see Fig. 5). Translating each note to its pitch and duration value for the SOUND statement gives the two rows of numbers under the music in Fig 5. The best way to convert these numbers to sound is to use a DATA statement thus:

```
5  C=5
10 DATA 69,1,89,1,89,0.75,89,0.25,
   89,1,105,0.75,97,0.25,89,1,85,
   0.75,77,0.25,69,0.75,99,99
20 READ P,D
30 IF P=99 THEN STOP
40 SOUND 1,-15,P,D*C
50 SOUND 1,1,P,2
60 GOTO 20
```

Line 50 has the effect of leaving short silences between each of the notes. Without this line, all the notes run together — try deleting it and re-running the program to appreciate the effect, it is one that you'd want to use to 'slur' notes. You can program any tunes that you have music for in much the same way.

STRIKING A CHORD

Most home computers with a sound generator could manage the simple tune given in the last section. What is special about the BBC Micro is that it is possible to generate three notes at the same time. To see how this sounds, try the following:

```
10  DIM N(13)
20  DATA 53,61,69,73,81,89,99,101,
    109,117,121,129,137
30  FOR I=1 TO 13
40  READ N(I)
50  NEXT I
60  A$=INKEY$(0)
70  IF A$="" THEN GOTO 60
80  A=VAL(A$)
90  SOUND 1,-15,N(A),20
100 SOUND 2,-15,N(A+2),20
110 SOUND 3,-15,N(A+4),20
120 GOTO 60
```

If you RUN this program (by pressing each of the keys 1 to 8) you will be able to hear the eight chords produced by adding a third and a fifth to each of the notes in the scale of C. (A third is a musical interval corresponding to playing a note two notes higher up and a fifth corresponds to playing a note four notes higher up.) This is the simplest kind of chord, called a triad, and is very pleasing to the ear.

Typing in almost any combination of the number keys 1 to 9 will produce something tuneful and it is easy to sit at your BBC Micro and produce 'music'. For example, if you want to hear a snatch of tune which is almost recognisable, try typing in the following sequence:

```
5 5 6 6 4 5 7 7 8 7 6 5
```

No prizes for guessing this one! The array N is used to hold the pitch values for the notes of the scale of C and enough notes higher up to form the triad on B. You can write a program to play a piece of music with up to three-note chords using the same method as given for the single melody in the last section.

There is one thing wrong with the previous program, however, and that is that each note of the chord starts at a slightly different time. In other words, each of the SOUND commands starts its note in the chord as soon as it is reached and, as they are executed one after another, the note on Channel 1 starts a little before that on Channel 2 which starts a little before that on Channel 3. The solution to this problem would be to tell the sound generator to wait for two other notes after the one initiated by line 90 before making any noise at all. This is the purpose of the S part of the channel parameter introduced in the section about the form of the SOUND command. If you use a non-zero value for S, the sound generator will wait for other notes before it starts playing. The number of notes that it waits for is given by the value of S and the SOUND commands which produce them must also use the same value of S. For example, in the case of the triads played by the previous program the SOUND commands would be replaced by:

```
90  SOUND &0201,-15,N(A),20
100 SOUND &0202,-15,N(A+2),20
110 SOUND &0203,-15,N(A+4),20
```

The first SOUND command has a value of S equal to 2 so the sound generator waits for two more SOUND commands with S set to 2 before producing a chord made up of all three notes.

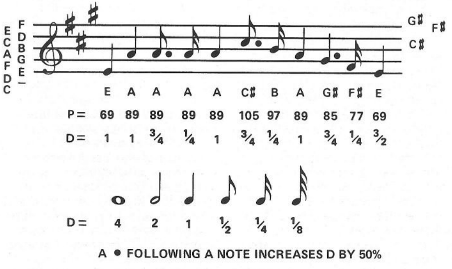


Fig. 5. The first few notes of Hearts of Oak and their digital values for the SOUND command.

The other parts of the channel parameter are also to do with the timing of notes. The H part of the parameter can either be a 0 or a 1 and if it is a 1, it adds a dummy note to the sound queue which allows any previous notes to continue without being cut short by another note. This really only makes any sense when used with the ENVELOPE command. The F part can be either 0 or 1 and if it is 1, it causes any notes stored in the channel's queue to be removed or 'flushed' and the note specified by the current SOUND command to be produced immediately.

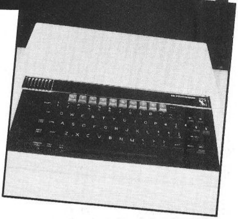
SIMPLE SOUND EFFECTS

The only sound channel that we haven't discussed as yet is the noise channel — Channel 0. The noise produced by this channel depends on the value of the pitch parameter P in the SOUND command:

P	Noise
0	High frequency periodic
1	Medium frequency periodic
2	Low frequency periodic
3	Periodic of a frequency set by Channel 1
4	High frequency 'white' noise
5	Medium frequency 'white' noise
7	Noise of frequency set by Channel 1

The first three noises (P = 0 to 2) are rasping sounds which come in very handy for 'losing' noises in games! Values of P between 4 and 6 produce hissing noises of various frequencies. White noise is a special sort of hissing which is made up by mixing a note of every pitch in much the same way that white light is made up by mixing light of every colour.

There isn't very much that you can do to change the nature of the sounds produced when P has a value of 0, 1, 2, 3, 4, 5 or 6 apart from altering the volume and duration. However, by changing only these two parameters and combining noises, you can still produce a useful range of effects. For example, if you make any noise very short it begins to sound



'percussive' (like something being hit) and if you combine a very short burst of white noise with a very short high pitched tone, you produce a noise like a metallic click. Try:

```
10  SOUND 0,-15,4,1:
    SOUND 1,-15,200,1
```

Similarly, mixing two noise-like sounds produces new effects. The following example:

```
10  SOUND 0,-15,4,1:SOUND 0,-15,3,1
20  GOTO 10
```

produces a sound like a machine gun. Notice that as this example uses the same channel twice, the two sounds follow each other to give a rhythmical pulsing sound. Using this idea with two different pitches of 'white' noise produces a sound like a helicopter:

```
10  SOUND 0,-15,4,2
20  SOUND 0,-15,5,1
30  GOTO 20
```

Notice that one of the sounds has to be twice as long to give the pulsating beat of a helicopter's rotor blades. You can go on experimenting like this for days! The range of sounds which can be produced using Channel 0 alone is so great that discovering new sounds is easy — putting a name to them, however, is quite a different problem!

The pitch values 3 and 7 are special because they produce noises on Channel 0 which are controlled by the pitch on Channel 1. This opens the door to sound effects which involve noises which change in pitch. For example:

```
10  SOUND 0,-15,7,55
20  FOR I=200 TO 255
30  SOUND 1,0,I,1
40  NEXT I
```

produces a noise like a space ship taking off. The pitch of the noise on Channel 0 started by line 10 is continuously changed by line 30. Notice that using a volume of 0 means that the sounds produced by line 30 are silent! Finally, try:

```
10  SOUND 0,-15,7,55
20  SOUND 1,0,200,1
30  SOUND 1,0,255,1
40  GOTO 20
```

which produces a sound like a car engine being started (or rather failing to start!).

Some of the simple sound effects developed here can be considerably improved by use of the ENVELOPE command but for occasional use, it seems like 'overkill' to use anything more than SOUND. However, when it comes to music then the ENVELOPE command has a lot going for it!

Ian Nicholls

MEMORY SAVER#2

Learn to control the 6845 CRT controller chip.

There are two chips which control the display of the BBC Microcomputer, one is the Ferranti Uncommitted Logic Array (ULA), and the other is the 6845 CRT controller chip. The 6845 provides the basic screen layout, determining the number of characters per row and the number of rows in each mode. The ULA handles the production of the very impressive colour graphics. Acorn are not very forthcoming about the instructions needed to control the ULA, to say the least! However, it is possible to send instructions to the 6845 to alter the way it works, and to be confident of knowing what the result will be.

ANATOMY OF A CHIP

The 6845 chip has 18 internal registers which, besides the functions mentioned above, control features such as the shape of the cursor, whether it flashes, and the number of scan lines per row. The BBC Micro automatically puts the appropriate values in these 18 registers when its display mode is altered.

One of the most useful modifications we could make would be to display Mode 6. This mode is text only with 40 characters per row and 25 rows. It would be a likely choice to use when translating a program written for a PET, for example, if we needed to have user-defined characters. It uses up 8K of memory. If you try the following in command mode:

```
MODE 6
VDU 19,0,4,0,0,0
```

This direct command will show how Mode 6 wastes memory.

you will see why Mode 6 would normally be unsuitable for our purpose — there are two blank lines between every character row. If we could remove these two blank lines, then we could avoid the need to use Mode 5, and 10 K of RAM, to obtain a PET-compatible display. The extra 2K of RAM to store program statements would be very useful!

A book which provides the inside information on the control registers of the 6845 chip is Gerry Kane's **CRT Controller Handbook** published by Osborne/McGraw-Hill. Searching through this reveals that register 9 controls the number of scan lines per row, which is the feature that we want to alter in display Mode 6. In Mode 6 it normally contains the value 9, one less than the number of scan lines actually used per row.

We want this figure to be 7, to give eight scan lines per row, since each character is defined within a matrix eight dots wide by eight lines deep.

To alter the contents of register 9 there are basically two methods, one is to access the appropriate memory locations directly using the ? indirection operation (equivalent to PEEK and POKE on other microcomputers). The other method uses the VDU 23 command.

a) Direct memory access. This is a two stage process. Memory location hexadecimal FE00 (written &FE00 in BBC BASIC) has to be loaded with the number of the control register to be addressed; then memory location & FE01 has to be loaded with the new value for the register that has been selected. So, to put the value 7 into control register 9, we need to execute the following statements:

```
?&FE00 = 9
?&FE01 = 7
```

The main disadvantage of this method is that it will not work across the Tube, Acorn's means of adding a second processor to the basic BBC machine. To make the method work with a second processor we would need to use the second approach to accessing the 6845 registers.

b) Using the VDU 23 command. Page 385 of the new **User Guide** gives details of the use of the VDU 23 command to alter values in the 6845 control registers, it is:

```
VDU 23,0,R,X,0,0,0,0,0,0
```

A more compact form is:

```
VDU 23,0,R,X,0;0;0;
```

where R is the register to be addressed, and X is the value to be placed in it. So to put 7 into control register 9, we can use the statement:

```
VDU 23,0,9,7,0;0;0;
```

If you enter this statement in command mode, having first moved into Mode 6, you will probably be disappointed since the picture will be rolling round and round the screen. If you are using a monitor, rather than a domestic TV, you may be luckier, a monitor's ability to lock on to the vertical synchronisation signal and produce a steady picture is much better than a domestic TV's.

However, we obviously have to delve a little deeper into the inner workings of the 6845 chip.

Press the BREAK key to restore a steady picture, and Mode 7, and program the function keys with the following statements:

```
*KEYØMODE6:VDU19,Ø,2,Ø,Ø,Ø,¦M
*KEY1VDU23,Ø,9,7,Ø;Ø;Ø;¦M
*KEY2VDU23,Ø,4,35,Ø;Ø;Ø;¦M
*KEY3VDU23,Ø,7,3Ø,Ø;Ø;Ø;¦M
```

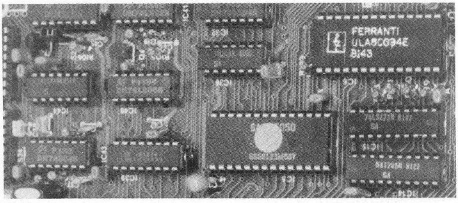
Enter these function key codes for an instant demo.

Pressing function key 0 puts us in Mode 6 and gives a green background, to show the effect of the two extra scan lines. Pressing function key 1 alters register 9 and repeats the problem we have already seen, loss of vertical synchronisation.

Register 4 is called the vertical total register; it controls the number of displayed and undisplayed character rows, and provides a coarse control over vertical synchronisation. It normally contains the value 30 in Mode 6. Typically, values between about 33 and 40 will restore a stable picture with most domestic colour TVs. Pressing function key 2 puts 35 into register 4, and should restore a steady picture. You may need to experiment a little with this value on your own TV.

However, although the display is now steady and no longer has the extra scan lines (the background should be solid green with no horizontal black stripes), it is not centred on the screen. We need to alter the value in the vertical sync position register, which is register 7.

Pressing function key 3 puts 30 into register 7, which should move the display back to the middle of your screen. We now have a PET-compatible display of 40 characters by 25 rows, which allows us to have user-defined characters and which only (!) occupies 8K of video RAM. Those of you who would like to delve further into the 6845 chip will find the book by Gerry Kane useful, and also **The BBC Micro Revealed** by Jeremy Ruston, published by Interface Publications.

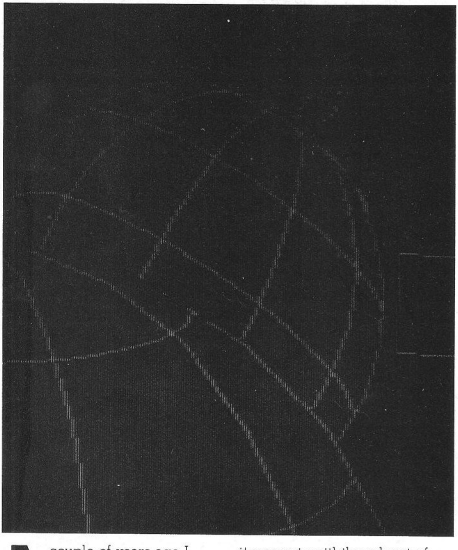


The starring chips on the video circuit!

Simon T Ainsworth

3-D ANIMATION

A program that boldly goes where no program has gone before.



A couple of years ago I went to see the science fiction film. ***Alien***, and was most impressed by the computer graphics sequence in which a spaceship was landed on an unexplored planet with the aid of a rather futuristic navigational computer. This, I thought, was the quality of graphics which I would like the microcomputer I would buy to be capable of producing. However, at the time high-resolution colour graphics were an expensive luxury available on relatively few machines, and it was not until the advent of micros like the BBC Micro that Hi-Res entered the low-cost market.

As an exercise, shortly after buying my Model B BBC Micro, I set myself the task of writing a program which would mimic the display of the Nostromo's navigational computer. On my second attempt I wrote ANIMATION, a typical screen of which is shown above. Four spinning planets are displayed in symbolic form as blue spheres, on which the lines of longitude and latitude are drawn. Several hundred stars drift across the screen in the background, creating the impression that the observer is moving with the planetary system relative to the sidereal frame. Meanwhile, the ship's flight-path is indicated by a series of square boxes shooting from the foreground away into the far distance, changing course and rotating as they go, before finally vanishing at the pole of the destination planet.

THE ANIMATION TECHNIQUE

The first version of ANIMATION took the most obvious, but rather naive approach. The idea was to make, for example, the stars appear to move by repeatedly deleting them and redrawing them in slightly different positions on the screen. The main problem with this was how to deal with stars which go behind a planet, and then re-emerge on the other side. There was also the question of speed! It took so long to shift roughly 300 stars around the screen that, far from being fast and smooth, the star's motion was very slow and jerky. This method might work on the ICL mainframes used by the producers of ***Alien***, but eight-bit micros are simply not fast enough.

I then stumbled across the remarkably elegant animation technique which I was eventually to employ. It makes use of a facility of the BBC Micro to redefine 'logical to actual colour relationships'. To explain, suppose you draw a ball on the screen in logical colour 1, which is red by default. Now, suppose you decide that you want the ball to turn green. You can do this very easily with the command VDU 19,1,2;0; which tells the computer that colour 1 is now green. You can make the ball disappear by turning it black with VDU 19,1,0;0; and make it reappear in red again with VDU 20, which just resets all the default actual colours. Not that the contents of the screen memory have not been changed — the computer has merely been instructed to interpret them in various different ways. This ability to make objects disappear and reappear, without access to the screen memory, is the crux of the technique.

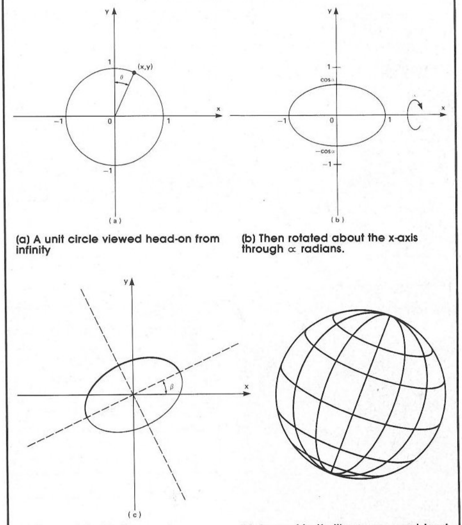
For example, suppose we define logical colours 1-15 to be black, then we draw 15 balls on the screen, each in a different logical colour, as shown in Fig. 1. Now, we define colour 1 to be yellow, pause for a moment, then re-define it as black. The process is repeated with colours 2-15 in turn. If we do this at the right speed we produce the illusion that there is a single yellow ball, moving from left to right. The beaûty of the technique is that once you have set up the screen memory, your program needs to know neither the shapes nor the positions of the objects being animated. A yellow ball, a bowl of petunias or even a surprised-looking sperm whale can be brought to life equally easily!

When you run ANIMATION you will first see how the screen is built up, layer upon layer, using all 16 colours available in graphics MODE 2. Logical colours 3 to 8 are used to animate the stars and the flight-path, whereas the lines of longitude of the planets are drawn in colours 9 to 14. The reason for using two groups of 'cycled' colours instead of one as in the example above, is that the stars and boxes must turn black in order to become invisible. The lines of longitude, on the other hand, must merge into the blue surface of the planets in order to disappear. So two groups of colours are used, the first group cycling between yellow and black, whilst the second group cycles between cyan and dark blue.

	●	●	●	●	●	●	●	. . . ●
Logical colour	1	2	3	4	5	6	7	. . . 15
Actual colour	Black	Black	Black	Black	Black	Yellow	Black	. . . Black

Fig. 1. Animating a moving ball.

Fig. 2. Drawing the planets.



(a) A unit circle viewed head-on from infinity

(b) Then rotated about the x-axis through α radians.

(c) Then rotated in the x-y plane through β radians. The thickened section of the curve is a half-ellipse.

(d) Several half-ellipses are combined to produce the image of a planet.

A LITTLE MATHS

Volumes could be written on the subject of displaying 3-dimensional structures on a 2-dimensional TV screen, but luckily the method which I have used in ANIMATION is the easiest one to explain.

As you read this magazine, your eyes are detecting beams of light travelling in roughly straight lines from all over the page, and the maximum angle between any two such beams is probably about 30 degrees. If you were reading the magazine from a range of 100 yards then this subtended angle would be about 0.3 degrees, and if you had the ultimate in longsightedness and could read this from infinitely far away then the subtended angle would be zero. When drawing distant objects on a computer, it is often easier to pretend that they are infinitely far away, since then all beams of light reaching the observer are parallel. This is the simplification made by ANIMATION when drawing the four planets.

Lines of latitude and longitude on a sphere are circles, and when you view a circle at an angle, from infinity, it looks like an ellipse. We only see half of each of these ellipses, since half of each line of latitude or longitude is on the far side of a planet. Hence, the image of each planet consists of several half-ellipses, as shown in Fig. 2.

CONVERSION NOTES

Owners of micros other than the BBC Micro who wish to convert ANIMATION to run on their machines should first check that their computer:

a) has a 16 colour graphics mode,
b)does not mind if you try to draw shapes which go off screen, and
c) can redefine 'logical to actual colour relationships', as described earlier.

If so, then the following notes will be useful.

MODE 2	160 x 256 graphics in 16 colours, 32 x 20 text. Software regards the screen as 1024 units high by 1280 units wide. Actual colours used by ANIMATION are: 0 black (background) 3 yellow (stars and flight-path) 4 blue (planets) 6 cyan (lines of latitude and longitude) 10 flashing green-magenta (titles)
VDU A,B,C.	Similar to PRINT CHR$(A) + CHR$(B) + (A,B,C, etc, are usually special control codes). If a semicolon follows a number in the VDU list, then it is printed lo-high as a double byte pair. For example.
VDU 5	Used here to make the text cursor vanish
VDU 18,A	Same as GCOL 0,A
VDU 19,A,B;0;	Logical colour A is actual colour B
VDU 29,X,Y;	Graphics origin has co-ordinates (X,Y) relative to the bottom left-hand corner of the screen
PROCfred (A,B)	Calls the procedure called fred and passes the values A,B etc, to the corresponding local variables listed in the procedural definition
DEF PROCfred (P,Q)	Start of procedural definition
ENDPROC	End of procedural definition
REPEAT UNTIL FALSE	An infinite loop
K9=INKEY(5)	Used here as a 50 millisecond delay
MOVE X,Y	Moves graphics cursor to co-ordinates (X,Y) relative to the graphics origin
DRAW X,Y	Draws a line to point (X,Y)
PLOT 85,X,Y	Fills in triangle, the vertices being (X,Y) and the last two points visited
GCOL 0,A	Graphics now in local colour A
POINT (X,Y)	Returns the colour of graphics point (X,Y) or − 1 if that point lies off the screen

Lines	Effect
100-2010	Select graphics MODE 2 and write titles
2020-2050	Draw four planets with various positions, sizes and angles of tilt
2060	Draws the stars and the flight-path
2070-2090	Define actual colours, ready to commence animation
3000-3080	Infinite loop which animates display as described earlier
4000-4160	Definition of the 'sphere' procedure. Set up the graphics necessary to animate a planet, centred on screen co-ordinates (X%,Y%) with a radius R%' and angle of axial inclination = tilt radians
5000-5090	Definition of the 'arc' procedure. Draw part of an ellipse of semi-minor and semi-major axes H% and W% respectively. The ellipse is centred on polar co-ordinates (D%, Alpha) and is orientated so that the minor axis passes through the graphics origin. The elliptic parameter is allowed to vary between − Beta and + Beta
6000-6080	Definition of the 'path' procedure. Set up graphics for the flight path
7000-7070	Definition of the 'square' procedure. Draw a square of side R%, centred on screen co-ordinates (X%,Y%) and rotated through an angle of tilt radians
8000-8090	Definition of the 'stars' procedure. Set up graphics required to animate the stars

```
LIST
  232REM ***  'ANIMATION'  ***
 1010REM *** STA July 1982 ***
 1020
 2000MODE 2:COLOUR 15
 2010PRINT TAB(6)"ANIMATION":VDU 5
 2020PROCsphere(150,900,100,-PI/6)
 2030PROCsphere(1190,945,75,-5*PI/4)
 2040PROCsphere(620,512,400,7*PI/6)
 2050PROCsphere(200,-292,900,0)
 2060PROCstars:PROCpath
 2070VDU 19,1,6;0;19,2,4;0;19,15,10;0;29,0;0;
 2080FOR I%=9 TO 14:VDU 19,I%,0;0;:NEXT
 2090
 3000REPEAT
 3010FOR I%=3 TO 8
 3020J%=I%-1:IF J%=2 J%=8
 3030VDU 19,J%,4;0;19,I%,6;0;
 3040VDU 19,J%+6,0;0;19,I%+6,3;0;
 3050K9=INKEY(5)
 3060NEXT
 3070UNTIL FALSE
 3080
 4000DEF PROCsphere(X%,Y%,R%,Tilt)
 4010VDU 18;2,29,X%;Y%;:MOVE 0,R%
 4020FOR Phi=0 TO 6.4 STEP .15
 4030MOVE 0,0:PLOT 85,R%*SIN Phi,R%*COS Phi
 4040NEXT
 4050Col%=3
 4060FOR Phi=0 TO 3.1 STEP .1
 4070GCOL 0,Col%
 4080PROCarc(R%*COS Phi,R%,0,PI/2+Tilt,PI/2)
 4090Col%=Col%+1:IF Col%=9 Col%=3
 4100NEXT
 4110GCOL 0,1
 4120FOR Theta=.5 TO 2.5 STEP .5
 4130PROCarc(-R%/4*SIN Theta,R%*SIN Theta,R%*COS Theta,Ti
lt,1.5)
 4140NEXT
 4150ENDPROC
 4160
 5000DEF PROCarc(H%,W%,D%,Alpha,Beta)
 5010S=SIN Alpha:C=COS Alpha
 5020X%=W%*SIN Beta:Y%=D%+H%*COS Beta
 5030MOVE C*X%+S*Y%,C*Y%-S*X%
 5040FOR Gamma=-Beta TO Beta+.1 STEP .25
 5050X%=-W%*SIN Gamma:Y%=D%+H%*COS Gamma
 5060DRAW C*X%+S*Y%,C*Y%-S*X%
 5070NEXT
 5080ENDPROC
 5090
 6000DEF PROCpath
 6010X%=1179:Y%=130:R%=100:Col%=9
 6020FOR Tilt=0 TO 1.2 STEP .05
 6030GCOL 0,Col%:PROCsquare(X%,Y%,R%,Tilt)
 6040X%=.9*X%+80:Y%=.5*Y%+440:R%=R%*.92
 6050Col%=Col%+1:IF Col%=15 Col%=9
 6060NEXT
 6070ENDPROC
 6080
 7000DEF PROCsquare(X%,Y%,R%,Tilt)
 7010VDU 29,X%;Y%;
 7020S=SIN Tilt:C=COS Tilt
 7030MOVE R%*(C+S),R%*(C-S)
 7040DRAW R%*(C-S),R%*(-C-S):DRAW R%*(-C-S),R%*(S-C)
 7050DRAW R%*(S-C),R%*(C+S):DRAW R%*(C+S),R%*(C-S)
 7060ENDPROC
 7070
 8000DEF PROCstars
 8010VDU 29,0;0;
 8020FOR I%=0 TO 40
 8030X%=1279:Y%=25*I%:X1%=-8*(2+RND(2)):Y1%=1-RND(3):Col%
=RND(6)+8
 8040REPEAT Col%=Col%+1:IF Col%=15 Col%=9
 8050P%=POINT(X%,Y%):IF P%=0 GCOL 0,Col%:PLOT 69,X%,Y%
 8060X%=X%+X1%:Y%=Y%+Y1%
 8070UNTIL P%<0
 8080NEXT
 8090ENDPROC
>
```

Listing 1. The ANIMATION program.

S Draper

GCOL APPLICATIONS

Using moving graphics in a program can have its problems, unless you make use of GCOL1 and GCOL3.

If you write games with moving graphics in them, and you use GCOL0 then you may find your graphics have an annoying flicker. This is because, when using GCOL0, there is a short time, between plotting the graphic out at its old position and plotting it in at its new position, when there is nothing on the screen. This may be cured by using GCOL3. It is then possible to plot the new graphic in before plotting the old one out (using GCOL3,COL both times where COL is the colour of the graphic):

```
1000 DEFPROCPLOTGRAPHIC
     (COL,OLDX,OLDY,
     NEWX, NEWY)
1010 VDU5
1020 GCOL3,COL
1030 MOVE NEWX,NEWY
1040 VDU 224
1050 MOVE OLDX,OLDY
1060 VDU 224
1070 VDU4
1080 ENDPROC
```

This procedure will plot graphic character 224 in a specified colour at any position on the screen (NEWX,NEWY) at the same time as plotting it out at its old position (OLDX,OLDY). For instance to move graphic 224 across the screen it could be used as follows:

```
100 MODE 4:PROCPLOT
    GRAPHIC (1,2000,2000,0,
    500)
110 FOR X = 4 TO 1280 STEP 4
120 PROCPLOTGRAPHIC
    (1,X—4,500,X,500)
130 NEXT X
```

Similarly GCOL1 can be useful when we wish to plot a background over a foreground so that it only shows where there is a gap in the foreground. For instance, in Mode 4 make the foreground colour 3 and the background colour 1 or colour 2 (or both for a multicoloured background). The foreground may then be plotted first and when the background is subsequently plotted over it, it will only show up in gaps in the foreground.

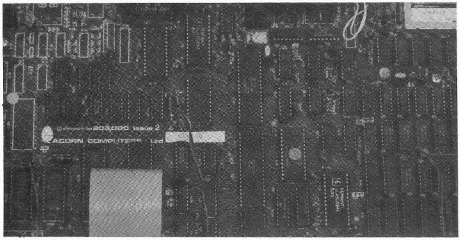
If, on the other hand, we want a two colour foreground over a one colour background we may plot the background first and simply use the ordinary GCOL0 statement for the foreground.

Clearly this technique may be expanded to give more colours and a midground if we use Mode 2, but the basic technique is the same in both cases.

S Draper

MEMORY SAVER #3

Since program storage can be a little tight on your BBC Micro Model A, here are some tips on how to maximise your storage capacity.



When using a model A with 10K of graphics memory there is only 2½K left for program storage. This makes things a little tight for all but the very simplest programs, and so I have attempted here to list a few simple but effective memory saving techniques to help you with your model A programming.

Storage space is used essentially for two things — your program and the variables it uses. We can save memory in both of these categories: I shall deal with program storage first since there is less to be said here.

1) Don't use unnecessary spaces. I know spaces in programs make them that make them that much more readable, but with only 2½K to play with space is at a premium. For instance, replacing: IF A = 10 THEN PROCEQUAL by IF A = 10PROCE saves eight bytes out of 16 — a saving of 50%! Note here I have also not used long variable names or procedure names as this also chews up valuable storage space.

2) The condition IF A is 0 can be replaced by IF A saving three bytes. Similar savings are often possible using logical variables as above.

3) Use REMs on paper when developing your program, but not in the program itself.

4) Put instructions, ENVELOPE definitions, and character definitions (VDU23,...) in a separate program which is loaded first. The first program then prints out the instructions, defines envelopes etc and CHAINs the second program. Note that this could have been done with the Mars-lander program elsewhere in this issue.

5) Don't use brackets where they're not needed, eg PRINT TAB(10);CHR$(65) can be replaced by PRINT TAB10CHR$ 65.

6) If you still get the dreaded 'No room' message, but only need to save a small amount more, it may be possible to do so by the use of larger multi-statement lines rather than lots of smaller lines — this saves four bytes per line.

Next we have variable storage memory. Firstly arrays:

a) If the numbers to be stored in the arrays lie in the range 0-255 (or can be arranged to do so) use the ? indirection operator (see **User Guide** page 411) to set up a byte vector, eg:

```
DIM A(100)
.
.
.
A(B) = C
D = A(E)
```

can be replaced by:

```
DIM A 100
.
.
.
A?B = C
D = A?E
```

saving a phenomenal 300 bytes in variable storage. Notice that there are no brackets in the DIM statement here — this is very important and instead of dimensioning a normal array of 100 elements it reserves 100 bytes for the byte vector A.

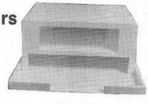
Incidentally this method of storing data is also faster than using arrays, which can be quite an advantage for some games.

b) Failing this use integer arrays where possible as this saves one byte per element over normal arrays.

Having dealt with arrays we now come to ordinary variables. I can't give any real hints here since I don't know any! Contrary to popular belief, the use of integer variables does not save memory since for every byte saved in the variable storage space several are taken up in the program storage space by % signs. However, integer variables are slightly faster than their real counterparts.

One final point — when writing model A programs on a model B it is useful to be able to simulate the model A memory situation: this can be done by adding the line HIMEM = &1800 after every Mode change.

Consoles for the BBC, Oric-1 and other Microcomputers



Console 'A' illustrated.

Professional desktop Console 'A' for the BBC and other microcomputers up to maximum dimensions of: Width 410mm, Depth 460mm and Height 96mm. The Console raises the TV/Monitor to minimise eyestrain, holds in place two tape recorders or two disk drive units side-by-side or one tape recorder and one disk drive and hides most of the wiring.

Professional desktop Console 'B' for the ORIC-1 and other microcomputers up to maximum dimensions of: Width 280mm, Depth 370mm and Height 52mm. The console raises and tilts the TV/Monitor to minimise eyestrain, holds in place one disk drive unit and tape recorder on top of disk drive or two disk drive units one on top of the other and hides most of the wiring.

The TV/Monitor stand of Console 'A' and 'B' is removable for access to add-ons and wiring.

Date........................

YOUR NAME AND ADDRESS ....
(In Capital Letters)

....

....

....

Qty	Description	Item Price	Total
	Console 'A'	£39.95	
	Console 'B'	£36.95	
		Sub Total	
		Carriage	£3.00
		Total Due	

Consoles' colour finish is matt cream.

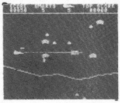
Dimensions: Console 'A': Width 510mm, Depth 520mm and Height 300mm. Console 'B': Width 660mm, Depth 442mm and Height 172mm. All Prices inclusive of V.A.T.

Delivery: Allow 28 days for Console 'B' and 60 days for Console 'A'. Cheques/P.O. Made payable to:

COMPUTERLOCK, 2 Wychperry Road, Haywards Heath, West Sussex RH16 1HJ. Telephone: (0444) 451986

NEW! ATOM NEW!

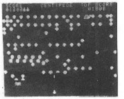
ALL ACTION PACKED M/CODE ARCADE GAMES



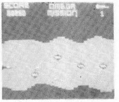
PROTECTOR £7

The most realistic version of this amazing Arcade game available for the Atom. Defend your humanoids from mutation and destroy the Aliens in this hyper fast, action packed game. Moving planetary surface, repeating lasers and smart bombs, thrust, space, increasing Attack Waves, 6 types of Aliens, Sound Effects, Top score and excellent mode 4 graphics are some of the features in this exciting game.

CENTIPEDE £6



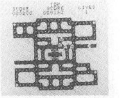
The first and only version of this popular Arcade game for the Atom. Shoot down the splitting centipede as it swirls through the mushroom field. Also inhabiting the game are Spiders, Bugs and Snails. The action increases until only skill and quick thinking can save you. Excellent high speed mode 4 graphics, Sound Effects and Top score.



OMEGA MISSION £7

The first and only version of the superb Scramble Arcade game. Moving landscape! fly over mountains, through caves and tunnels. 5 different stages, Ground to Air Missiles, Fire Balls, Space Craft, Mutants and the narrow twisting Tunnel. Movement in 8 directions, laser cannon, Excellent COLOUR graphics (give Black & White on monochrome T.V.) mode 3a, Sound Effects, Top score.

PUCKMAN £5



One of the best versions of this popular Arcade maze chaser game. Eat all the dots in the maze but watch out for the hungry ghosts! Eat an Energy Blob and the chase reverses. Each maze cleared brings a new one with faster ghosts. Excellent high speed mode 4 graphics, Sound Effects and Top score.

ALL 12K RAM

PRICES INCLUDE P&P. FAST 2 DAY DESPATCH!

micromania

14 LOWER HILL RD. EPSOM. SURREY.

Electronequip

(Authorised BBC Dealer, and service centre)

SPECIAL OPENING OFFER DUE TO MOVE TO NEW PREMISES ALL ORDERS RECEIVED DURING NEXT MONTH QUOTING REF. COMT/C WILL BE ENTERED INTO A WEEKLY DRAW AND 2 CUSTOMERS IN EVERY 100 WILL RECEIVE THEIR GOODS FREE OF CHARGE

THIS MONTHS SPECIAL OFFERS

BBC36 High Quality 14" RGB Colour Monitor/TV. Colour monitor suitable for 80 columns with ability to receive TV.................... **£244.95**

BBC45 New improved cassette recorder for BBC. Has monitor facility, counter, remote.................... **£35.88**

BBC48 Dual 800K disc drives for BBC micro with free Z80 second processor card.................... **£897.00**

Large stocks of Software for many machines as well as BBC. Acornsoft, Bug-Byte, CP/M, Program/Micro Power, Computer Concepts etc.

Business systems enquiries welcome. Systems & Software available from 500 to 100,000.

Torch Colour Machine 800K floppies ex. VAT.................... £2795.00
Torch Colour Machine Hard Disc ex. VAT.................... **£4995.00**

BBC54 Daisy Wheel printer for BBC 12cps.................... £558.90

BBC

BBC1	BBC Micro Model A	£299.00
BBC2	BBC Micro Model B	£399.00
BBC3	BBC Model A Micro with 32K	£333.50
BBC4	BBC Model A Micro 32K & VIA	£339.50
BBC21	Upgrade Model A to B	£99.82
BBC27	Disc Upgrade for BBC B	£109.25
BBC30	14" Colour Monitor for BBC	£286.25
BBC33	BMC12A 12' Black/Green Monitor	£90.85
BBC34	12" Black/Green Monitor for BBC	£113.85
BBC35	12" Black/Ambre Monitor for BBC	£129.95
BBC36	14" Monitor/TV. 80 columns	£244.95
BBC40	Cassette Recorder for BBC	£29.90
BBC41	Single 5.25" Disc Drive 100K	£265.00
BBC42	Dual 5.25" Disc Drive for BBC	£447.00
BBC48	Dual 800K low profile disc drives	£897.00
BBC49	5.25" Discs for BBC 40/80 tracks	£2.20
BBC50	Epson MX80T type 3 for BBC	£373.75
BBC54	Daisy Wheel printer for BBC	£558.90
BBC70	Plinth/Stowage for BBC	£29.90
BBC80	Cassette lead for BBC	£4.60
BBC95	Printer lead for BBC	£17.25

Large stocks. Prices inclusive of VAT
All prices inclusive of postage except micros 3.00

All Upgrades etc. are fitted free of charge and the computer fully re-tested. Access and Barclaycard Welcome.

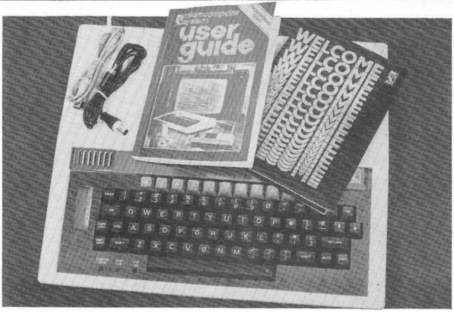
ACORN COMPUTER **Electronequip** BBC

36-38 West Street, Fareham, Hants TO16 0GN (0329) 236 670

Mike James

BBC USER REPORT

If you want to know just how good the BBC's computer system really is, read what our reviewer thinks after many weeks of testing and usage.



The story of how the BBC came to the decision to adopt a micro and how they found their way to the particular machine they eventually adopted, is a tale that will become part of the folklore of computing. Put simply, what happened was that the BBC decided that they would produce a series of programmes about the microcomputer and computing in general and felt that it would be desirable to link the series to the use of a particular micro. It should be obvious that the chosen micro would suffer severe sales problems — namely, they would keep running out of stock! A specification was drawn up around the end of 1980 and manufacturers were invited to tender the contract to produce the BBC micro.

At about the same time as the BBC were developing their specification, Acorn Computers were developing a successor to their very popular ATOM. Although they had only reached the prototype stage, the machine impressed the BBC sufficiently for them to drop one of their specifications (for a Z80 CPU) and accept Acorn's machine, 6502 CPU and all!

AN OVERVIEW

Before I go any further, I should say that I think the BBC micro is the most exciting and versatile micro I have seen to date. High resolution colour graphics and sound effects are standard features in a machine which costs less than £250! Of course it has faults (doesn't everything?) and I will point these out as I go along but all in all it is the machine at the top of my list of 'best buys'! To find out why read on . . .

The BBC micro is sold in two different forms: the Model A, a basic 16K machine costing £299 (including VAT), and the Model B, an extended 32K machine retailing for £399 (including VAT). The Model A machine as just stated, comes with 16K of RAM and a sound effects chip. However, as mentioned earlier, high resolution graphics in colour are also a standard feature (ie you don't have to buy any extra ROMs or colour boards), so even the basic Model A out-performs other machines in the same price bracket — for more details see the section on graphics. It is important to realise, however, that these two models are entirely a sales convenience and that the 'A' can be converted to the 'B' by the addition of the extra chips (at a cost of about £135). There could be hundreds of versions of the BBC micro depending upon which options are installed. In the Model B, for example, there is (in addition to the extra 16K of RAM) a serial printer interface, a parallel printer interface, an eight-bit user port and a four channel A to D convertor.

The overall appearance of the BBC Micro is smart — as can be seen from the photos. The case is made from lightweight plastic and is adequate for most environments (but don't try standing heavy weights on it, eg TV monitors). One of the most amazing things about the unit is its size and weight. For a machine with the expansion capabilities outlined above, it is very small and light, measuring 16" by 13", about 2.½" thick and weighing approximately 9 lbs. If you're interested in getting inside the case, then Acorn have made it easy — just four screws and the whole top lifts off giving very good access.

The machine is a pleasure to use. The keyboard feels good and has an auto repeat facility and three separate keys to provide upper case characters;

Shift and Shift Lock giving upper case on all the keys; and Caps Lock giving upper case on letters only. An additional row of user definable function keys are included and these are very easy to control from the software. Five keys are included for screen editing, the usual four cursor keys and a key marked Copy. My one complaint is the layout of the 'arrow' keys. It would have been nice if they could have been positioned like the points of the compass rather than left/right, up/down. However on a keyboard of this size I don't see how it could be done. Other people who have used my BBC Micro have had trouble with the Space bar. It is rather fussy about where it is hit and declines to throw a space if it is hit too close the the end rather than in the middle. Personally, I've never had a problem in this respect.

The display quality is remarkably sharp, both on monitors and on ordinary domestic TVs. One small problem is that on some sets the top line of the display vanishes outside the frame and on others the bottom line does the same. This is due to the rather complete use that the machine makes of the screen; however, it is fairly easy to remedy this 'fault' by using the *TV command. The cassette system is very easy to use and keeps you informed of exactly what is going on. In use it is about as good as a cassette system can be and has the handling characteristics of a very slow disc! (In case anyone is in doubt this *is* a compliment.)

HARDWARE

I have already said how much I like the mechanical construction of the machine and the sight of the internal layout should be enough to please even them most discriminating. All the chips on the main (only) board are socketed and neatly placed. The power supply is the small black box to the left of the case. The keyboard is fitted at an angle and slightly covers the main board. This should cause no problems as the keyboard can be removed by undoing two bolts and unplugging a short ribbon cable. Also mounted on the keyboard is a small loudspeaker for sound effects and the CHR$(7) 'bell'. The PCBs are well made; the main board is double-sided and printed with the names and locations of all the components. A slightly worrying problem is the poor support of the main board. It is fixed at four points and flexes if you try to remove or insert a chip into its socket. This may not sound like much of a problem until you notice that all but one of the I/O connectors are also mounted on the main board, so plugging and unplugging causes a similar flexing of the board. However, I should point out that I have had no problems in this respect during a nine month period of heavy use.

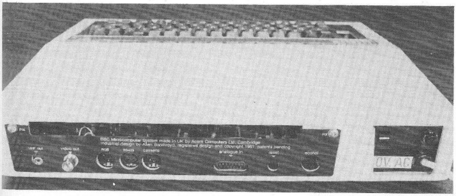
THE MAIN BOARD

The main board is divided into a number of functional areas (see Fig. 1). The RAM area contains eight or 16 dynamic RAM chips (4816) socketed so servicing should be easy. The ROM area on my machine contained not five ROMs, but one ROM and four 2732 EPROMs. The BASIC is contained in the massive 128K bit ROM. The four EPROMs currently contain the Machine Operating System (MOS). In later versions this will be put into another 128 Kbit ROM. What becomes of the three spare sockets, I hear you ask? The answer is that four of the ROM sockets are paged and can be used for 'alternative' software. For example, a disc operating system ROM could be installed and could be switched in to replace the BASIC ROM under software control. The idea of paging is a simple one which can be used to extend address space by allowing the micro to select which of a number of ROMs occupies a given address area.

Moving away from the memory area we come to the video processor ULA. ULA stands for Uncommitted Logic Array and is essentially a method of producing a large-scale integrated circuit for a reasonable cost. Put another way, this means that there are two chips inside the BBC Micro which have been designed by Acorn (and produced by Ferranti). The video ULA is responsible for most of the clever colour graphics the machine is capable of. It is certain that the use of this ULA is what makes the BBC Micro able to offer such good graphics for such a reasonable price.

GRAPHICS

This is certainly the single most interesting feature of the BBC Micro. There are, as always, two aspects of graphics — the hardware used which determines resolution, etc, and the software provided to make use of the hardware. Discussion of the software is left until later. The graphics hardware can work in eight distinct modes. Examining the table reveals a number of details. The highest resolution graphics is a remarkable 640 by 256 plotting



The rear of the machine showing the general I/O facilities.

points — this sort of resolution would have cost more than the entire machine a year ago! A standard (commerical) format 80 characters by 25 line screen is available only on the Model B. The memory used by each mode is taken from user RAM, not a special display memory. Notice that only the last four are available to the Model A because of the memory requirements.

As mentioned earlier, the graphics are produced mainly with the help of the custom built ULA chip. However, as it works, it must be receiving data from the user RAM and then rearranging it to represent the required screen format. For example, in Mode 0, each bit of the user memory corresponds to one screen location (pixel), but in Mode 1, you need two bits to determine the colour of each pixel. The ULA is responsible for collecting the number of bits each pixel requires and then determining which colour it should be. An area of memory inside the ULA is used as a 'palette' in the sense that it associates the codes stored in user memory with 'real' colours. For example, in the two colour mode, zero could be black and one could be white — but by reprogramming the palette, you could have blue and cyan! One last detail about the graphics ULA is that it accesses the user memory in between the read/write cycles of the 6502, so graphics display doesn't slow anything down.

In use, these colours are clear and the overall display effect is stunning. Plotting coloured lines in Hi-Res graphics couldn't be easier — just select your colour and plot the line!

The trouble with having all of these advanced graphics options is that it's all too easy to miss commenting on the less exciting things. So let me say, before I forget, that upper and lower case characters are present on both models; the text characters can be user defined (except for Teletext Mode 7) and text and graphics can be freely mixed on the screen. From the point of view of the hardware, text is just predefined graphics! It is worth pointing out that as this true then the BBC Micro is capable of being used to display the block graphics characters (or at any rate something close) of other machines. This would make converting programs which make use of specific graphics features very easy.

There are three video outputs on the back of the machine: one mixed video (BNC connector), one RGB (6-pin DIN) and one UHF modulated output (Phono connector).

SOFTWARE

As should have been clear from the hardware section, the BBC micro has its memory space divided into two 32K regions. The bottom 32K is used for RAM and the top 32K is used for ROMs and memory mapped I/O (see the memory map). This may seem like rather a lot of ROM for one machine but it is all used to good effect. As well as the superb BASIC, there is an assembler and all the routines necessary for cassette handling, etc. The trouble with having all this excellent software in 28K of ROM is that it does reduce the amount of user RAM. In the worst possible case, with Mode 0 graphics and a disc system, the user might only have 8K to play with! Don't let this put you off — in practice you could always move to lower resolution graphics. It does, however, point to a weakness of the machine — insufficient address space.

Mode	Graphics	Colours	Text	Memory	Model
0	640 by 256	2	80 by 32	20K	B
1	320 by 256	4	40 by 32	20K	B
2	160 by 256	16	20 by 32	20K	B
3	—	2	80 by 25	16K	B
4	320 by 256	2	40 by 32	10K	A and B
5	160 by 256	4	20 by 32	10K	A and B
6	—	2	40 by 25	8K	A and B
7	Teletext	16	40 by 25	1K	A and B

Table 1. The eight modes of the graphics hardware.

THE BASIC

The BASIC to be found inside the BBC Micro is brand new. It's not Microsoft BASIC but something produced by Acorn themselves. The only other successful micro which has left the Microsoft school is the ZX81 which has a BASIC which comes close to the standard set by Microsoft. The BBC BASIC is the first version ***better*** than Microsoft.

BBC BASIC is fast, as the results given show:

Test 1	0.8
Test 2	3.1
Test 3	8.2
Test 4	8.7
Test 5	9.1
Test 6	13.9
Test 7	21.3
Test 8	5.3

Results of Benchmark tests.

Along with the BBC hardware specification came a detailed specification for the BASIC their machine should run. The solution was to have as much Microsoft-compatible BASIC as possible and extend it to include the extra statements needed for structured programming.

If you're not too clear what all this talk of structured programming is about, then some of this discussion may be a little meaningless. There isn't enough space to explain the ideas of structured programming here but basically a structured program is one which uses only statements like IF some condition THEN. . .ELSE some condition, WHILE some condition DO. . ., and DO. . .UNTIL some condition (the '. . .' is taken to mean a collection of other program statements). Also, the use of full subroutines or procedures is required. It is true to say that the resulting product has lost a lot of the original specification but it is still too good to complain about.

FILE HANDLING

The reason why I've singled out the file handling commands is that this is one of the main areas where things might get difficult if you have to convert a Microsoft BASIC program. The cause of the trouble are the OPENIN and OPENOUT commands which are distinctly different from the better known OPEN command. OPENIN and OPENOUT are functions which return the logical file number as opposed to OPEN which is a command to assign a given logical file number to the file. I leave it to the reader to think of the fun this slight difference could cause.

GRAPHICS AND SOUND

The graphics commands of the BBC Micro are far too versatile and subtle for me to be able to give you anything other than a flavour of the subject. The first clever thing about the graphics is that no matter what mode you are in, the graphics screen is made to appear 1280 pixels wide by 1024 pixels high. This allows you to write graphics programs ignoring the resolution at which they will finally be used. I've had quite a lot of fun trying out the same program at various resolutions and comparing the differences.

The workhorse graphics command is PLOT. It has very many different functions including plotting a point, a line, a dotted line and even a solid triangle (!), either in absolute co-ordinates or relative to the last plotted point. I hope you noticed the bit about plotting a solid triangle because it's the most powerful part of the Hi-Res graphics commands. The triangle can be plotted in any valid colour and it appears very quickly on the screen. Why triangles? Surely rectangles are more useful? No — if you think about it, any shape can be made up out of triangles. In this sense the triangle is to drawing solid shapes what the line is to line drawings!

Before rounding off the description of the BASIC, I should mention the sound command. Its syntax is:

```
SOUND channel, volume,
frequency, duration
```

Channel can be from 0 to 3 with 0 as the noise channel, volume from 0 to − 15, frequency from 0 to 255 and the same for duration. The fun part of this command is that the three tone channels can be used at the same time.

Once you tire of 'pure' notes, you can always use the envelope command to change the characteristics of the note produced by the sound generator. However, the number of parameters involved in the envelope command means that it is not easy to work out how to produce any given or desired sound. After some practice, things do get a little easier and it is possible to produce some very impressive effects. The sound generator in the BBC Micro may be simple but the software used to drive it is very sophisticated!

There are many other features of the BASIC that make it enjoyable to use, such as long variable names, good (Microsoft style) strings and string functions, a renumber command, etc.

THE ASSEMBLER

One of the best features of the ATOM was the way in which assembly code could be mixed with BASIC. The BBC Micro has carried on this tradition by including an even better assembler in ROM. It's so easy to mix assembler and BASIC that I have a feeling that in the future I will be switching from one to the other without making my usual fuss.

EXPANSION

The BBC Micro has extensive expansion capabilities ranging from the entirely expected to the decidedly unexpected. But even those expansion interfaces which are normally taken for granted are rather special in this system.

The cassette system used by the BBC Micro is, as I have said before, very easy to use. It is also ***very*** reliable. The secret of this good natured storage is the second ULA in the machine — the serial processor. The serial processor is responsible for handling the coding of the cassette data and contains a digital clock/signal separator making it a complete signal processor. The use of a digital separator makes data recovery fairly independent of speed and volume fluctuations produced by low costs cassette recorders. Two record speeds are available: 30cps using a standard CUTS format, and 120cps using a CUTS related but non-standard format. Both work!

The cassette recorder is connected to the back of the machine via a standard seven pin DIN audio socket. Acorn don't provide a cable to connect to the recorder on the basis that they could only cover 30% of the types of connector with one lead. This is a pity, because it means that it is not possible to unpack and run the demonstration programs without first soldering on at least one plug.

The software used to control the cassette is clearly based on the ATOM cassette system. Named programs (names up to 10 characters) can be saved and loaded. The format used for writing the tape is such that if an error occurs it can be isolated to a particular block. The tape can be rewound and the read restarted at any earlier time. The first complete block found gives the name of the program and the block number. This information is used to continue the load. This means that it is not necessary to go right back to the start of a bad load — just re-read the blocks in error.

In order to use disc drives with your BBC Micro, you'll need some extra chips and an operating system ROM fitted. Once this modification has been made, you'll be able to use 40 or 80 track disc drives from a wide range of manufacturers. Alternatively, disc drives, in the BBC Micro's colours, are available from Acorn.

There are a number of 'odd' interfaces that I think of as falling into the category of 'user' interfaces. Starting with what is usually referred to as a user interface, the BBC Micro has an eight-bit parallel port. This is simply an unbuffered 'B' side of a 6522 PIA chip so it should be very familiar to anyone with a PET. Not all of the lines are available for unrestricted use in that the CB1 line is also used as a light pen input. Connection is made to the user port by a 20 pin ribbon cable plug mounted under the cabinet.

The other half of the PIA is used as a parallel printer port. The standard seven data and two handshake (busy and strobe) connections are provided on a 26 pin ribbon cable plug mounted under the cabinet. Presumably Acorn will provide cables for most printers.

A serial printer interface is also available using a standard 6850 ACIA. The only control lines provided are RTS and CTS, and these may be found on a five pin DIN socket at the back of the machine along with (of course) data-in and data-out. The use of a five pin DIN socket may cause some trouble if you're trying to connect a standard (RS232 or V24) piece of equipment which uses a 24 pin D connector. (But then it wouldn't be fun if they made it too easy!) The only other fact which might cause concern is that the serial interface is labelled RS423 rather than the more friendly and usual RS232. Have no fear — the RS423 is just a 'better' version of RS232 and may be used as if it were RS232 in 99% of cases.

The sound generator chip is sort of a user interface (computer to air!) so I will deal with it in this section. It is a fairly standard SN7 6489 sound effects chip containing some noise channel and three independent oscillators. This means that the BBC Micro can 'play' up to three note chords and make a wide variety of other bangs and pops.

The only other interface that comes into the general category of 'user' is the 'paddle' or analogue input interface. Connection to the on board A to D convertor (a uPD7002) is made via a 15 pin D socket (why use a D socket here and not on the serial port?) Apart from the four analog input channels there is also a five volt supply and a reference voltage. These are obviously used to feed the two BBC X, Y joysticks.

FACT SHEET	BBC Micro	
CPU	6502	
Clock	2MHz	
ROM	16K BASIC plus 16K MOS	
RAM	Model A 16K Model B 32K	
Language	BBC BASIC	
Keyboard	73 key QWERTY keyboard 10 user-definable keys, cursor control keys	
Display	Both models include: 320 by 256, 2 colour graphics and 40 by 32 text 160 by 256, 4 colour graphics and 20 by 32 text 40 by 25,2 colour text 40 by 25, Teletext display Model B only includes: 640 by 256, 2 colour graphics and 80 by 30 text 320 by 256, 4 colour graphics and 40 by 32 text 160 by 256, 16 colour graphics and 20 by 32 text 80 by 25,2 colour text	
Cassette	300/12,00 baud	
I/O	Model B only incorporates: Serial and parallel interfaces four channel A to D eight-bit user port 1 MHz expansion bus Tube	
Costs	Model A	£299.00 inc VAT
	Model B	£399.00 inc VAT
Supplier	BBC Microcomputer Systems c/o Vector Marketing Dennington Estate Wellingborough Northamptonshire NN8 2RL	

LOOKING AHEAD

For the BBC Micro, the future must surely be good. Without looking too far ahead, there are going to be lots of exciting extras. The speech synthesiser is now ready and a Teletext interface has been recently launched. There is a strange slot in the front of the case that will be used to take plug-in ROM packs for extra words for the speech synthesiser, etc (and maybe even prepackaged software). The Econet interface, already in use in a number of locations, including Acorn's own headquarters, will open up new ways of using home computers by providing the first low cost way of linking up a number of machines in a local network. I've already mentioned some of the other planned extras — a Prestel interface, a second processor connected via the Tube, either another 6502 or a Z80 running (ugh) CP/M, a 16-bit processor . . . All in all the BBC Micro is quickly turning into a powerful and exciting system.

DOCUMENTATION AND THE WELCOME TAPE

One of my main criticisms of the BBC Micro when it first appeared was the paucity of the information contained in the provisional **'User Guide'**. There was rather a long wait before this was replaced by the 'real thing', which was either frustrating or a challenge depending on how one felt about finding things out by trial and error. Eventually, the **User Guide** arrived. It resembles **War and Peace** in its size, but even so it does not tell the users all they want to know and there is room for lots more information. To be fair, however, the **User Guide** is arranged in a way that makes it easy to use and it does explain the things it covers well.

The word 'Welcome' in the heading may lead you to believe that there has been a printer's error and this bit should have come first. In fact, 'Welcome' is the title of a package of programs which comes with the BBC Micro just to show you what can be done. It comprises a cassette tape and a booklet and is really excellently produced to show off many of the features of the machine. In all, there are 16 programs including 'keyboard', which teaches you to type; 'alphasort', an amazing demonstration of a sort program; 'poem', an interactive, colour poem by Roger McGough; 'bat 'n ball', a simulated squash game — but rather a poor one; 'music', which allows you to play simple tunes; and 'kingdom', a decision-making game. It is remarkable that compilation of such extent and quality should be given away free with every BBC Micro.

CONCLUSIONS

As far as I'm concerned Acorn's new micro is an exciting departure for the BBC. A feature which it shares with the BBC is that it is an all-British product!

At this point, it is important to remember that there are two versions of the BBC Micro — the Model A and the Model B. The Model A is not really big enough, in terms of its memory, to take advantage of its own potential. With only 16K, it's rather hampered and its performance is nothing special. Its saving grace is, however, the fact that it can be up-graded to a Model B! It's not really possible to come to any conclusion about the Model B BBC Micro, other than it's well ahead of all currently marketed machines and has a clear price advantage — in short, it's excellent and certainly worthy of its prestigious name.

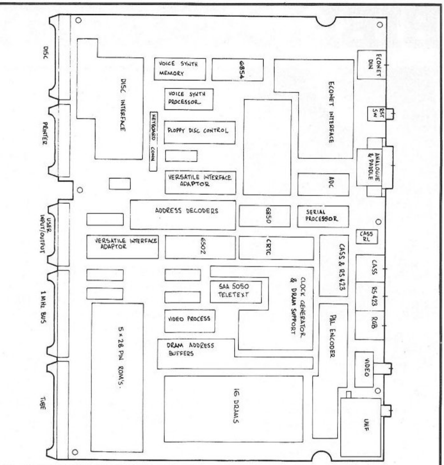


Fig. 1. What lives where on the main PCB.

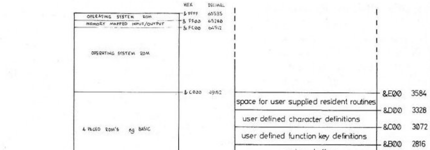
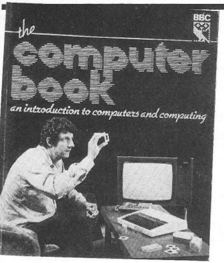


Fig. 2. What goes where in the memory map of the BBC Micro.

BIBLIOGRAPHY



When a new machine becomes available on market there inevitably follows a deluge of books telling us how to use the machine to its best advantage, The BBC Microcomputer is certainly no exception and we take a look here at some of those books,

No doubt more books will keep appearing on the publishers' lists as more people investigate and experiment with the BBC Micro so don't assume that this list ends here!

The BBC Micro Revealed: If you've mastered the contents of the manual that came with your BBC Microcomputer and now want to continue your exploration of the computer's functions and capabilities, this book is for you. The author a 17 year old student, spent months delving into the computer's internal operations in order to reveal a large number of sophisticated techniques to help the reader improve his or her programming skills. The book includes the following features: Details of how to construct our own display modes; A way to scroll the display in any direction (up, down, sideways and even diagonally); A visual analogue of the computer's memory transactions; Information on the way in which the computer stores its programs and line numbers; A technique for increasing the speed of your programs by up to 10%; Instructions on how to pass arrays and matrices to user-defined functions and procedures; and much, much more. If you're serious about developing your programming skills on the BBC Micro this book will prove an invaluable aid.

The BBC Micro Revealed by Jeremy Ruston is published by INTERFACE at £7.95 for 144 pages.
ISBN 0 907563 15 5.

Learning to use the BBC Microcomputer: This is one of a new series of 'Learning to use', books designed to provide potential users, established users, teachers, students and businessmen with standardised introductions to the use of popular microcomputers. This beginner's guide really does begin at the beginning: it assumes that you want to learn how to use the BBC Microcomputer in your work or leisure, not become a theorist in computing. The book provides a simple, down-to-earth, jargon-free introduction to the machine and its software.
Many applications of the BBC Microcomputer are described including business, educational and hobby uses; the micro's ability to produce and draw pictures and diagrams is explored and explained, and programs for a large number of graphics applications are presented. The book will not only appeal to the new BBC Microcomputer owner but also to the potential buyer since it will tell him how the BBC Microcomputer operates and performs and will help him assess whether the machine will suit his needs.

Learning to use the BBC Microcomputer by P N Dane is published by Gower Publishing Company Limited at £4.95 for 84 pages.
ISBN 0 566 03452 2.

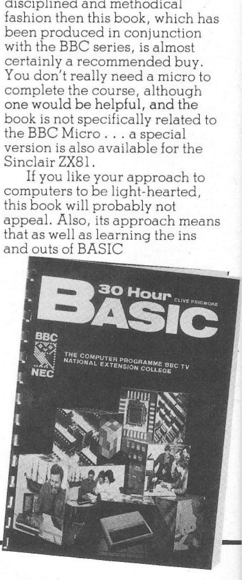
The Computer Book: This publication, although produced by the BBC, has no direct connection with either the BBC Micro or The Computer Programme. It simply attempts to introduce the 'computer shy' individual to the wide and diverse subject of computing, covering the ground in a relaxed and friendly fashion. The book doesn't introduce any radical new ideas and certainly cannot be regarded as a 'text' on computers but then that isn't really its object.

The features which really make this book stand out are its excellent production and layout and the clever use of photographs, illustrations and cartoons to keep the reader both interested and amused. If the quality of the editorial content was to the same standard. . . .

The Computer Book: This Bradbeer, Peter de Bono and Peter Laurie is published by BBC Publications at £6.75 for 254 pages.
ISBN 0 563 16484 0.

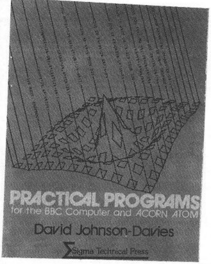
30 Hour BASIC: If you want to approach the subject of computer programming in a disciplined and methodical fashion then this book, which has been produced in conjunction with the BBC series, is almost certainly a recommended buy. You don't really need a micro to complete the course, although one would be helpful, and the book is not specifically related to the BBC Micro . . . a special version is also available for the Sinclair ZX81.

If you like your approach to computers to be light-hearted, this book will probably not appeal. Also, its approach means that as well as learning the ins and outs of BASIC



programming, you will learn to write clear and logical programs, something that happens all too seldom in many books.

30 Hour BASIC by Clive Prigmore is published by The National Extension College at £5.50 for 256 pages.
ISBN 0 80682 269 9.



Practical Programs for the BBC Computer and the Acorn ATOM: This somewhat slimmer volume contains four chapters based around a number of simple programs which are reproduced for both the BBC Micro and the ATOM. The presentation and layout is excellent and the structure of each of the examples is clearly explained. However, the real meat is to be found in the fifth chapter which presents SPL, Simple Programming Language, a new compiler for both types of micro. As well as providing a second high-level language, this chapter demonstrates how to go about writing a simple compiler.

Practical Programs for the BBC Computer and the Acorn ATOM by David Johnson-Davies is published by Sigma Technical Press (distributed by John Wiley & Sons Ltd) at £5.95 for 120 pages.
ISBN 0 905104 14 5.

BASIC Programming on the BBC Microcomputer: To have produced this introductory book in so short a time was a remarkable achievement by both the authors and the publishers. The sad fact, however, is that this is an introductory book, and as such tends to leave you waiting for more. The volume was put together with the assistance of Acorn, the company who designed and produced the BBC Micro, so it is very specifically related to that product.

The book is practical in nature with lots of small examples to try out and problems to solve. The actual information content is not significantly more than that in the early version of the **User Guide** except that the facts have been arranged in a more readable form. Although the book has areas of weakness, it does stand up as an introduction but one hopes that the second volume will not be too long in coming.

BASIC Programming on the BBC Microcomputer by Neil and Pat Cryer is published by Prentice Hall International at £5.95 for 205 pages.
ISBN 13 066407 3.

Assembly Language Programming for the BBC Microcomputer: Every BBC Micro comes equipped with an immensely powerful and very fast assembler; assembly language statements and BASIC statements can be freely mixed which enhances the programmer's potential control over the machine. Containing 73 listings of programs, the book is completely self-contained, and has various appendices on the 6502 instruction set, floating point and the user port, and a section on combining programs in the BBC computer using PAGE and *LOAD.
Two companion tapes are available with the book if you feel you do not want to type in all the programs yourself.

Assembly Language Programming for the BBC Microcomputer by Ian Birnbaum is published by Macmillan Press at £8.95 for 305 pages. The cassettes are £9.00 each or £16.00 for two.
ISBN 0 333 34585 1.

The Book of Listings. Fun Programs for the BBC Microcomputer: This first BBC book of listings contains a host of games and other programs, ranging from arcade-like action programs, through board games which will tax your wits, to some startling graphics demonstrations.

The authors have tried to make the most of the colour and sound potential of the BBC Micro, and have written most programs so that they will run on both Model A and B machines. The programs were developed on both the A and B Model machines with the 0.1 Operating System.

Structured programming techniques have been used as far as possible. Although programs may thus be a little longer than strictly necessary, they do tend to be relatively easy to debug and modify. Many of the program notes include suggestions as to how you can adapt the programs to make them your own and to develop them further. The programs are intended to entertain and to teach useful programming techniques.

The Book of Listings. Fun Programs for the BBC Microcomputer by Tim Hartnell and Jeremy Ruston is published by the British Broadcasting Corporation at £3.75 for 156 pages.
ISBN 0 563 16534 0.

Easy Programming for the BBC Micro: This book is explicitly a beginner's guide to working with the BBC Microcomputer. Starting with an explanation of what a computer is, the author takes you through the complexities of BBC computing including animation, strings, the use of flowcharts, editing, arrays, the comprehensive sound capabilities of the BBC Micro and includes a case history of a bugged program. Included in the text are 28 complete and ready to run programs and another 12 'additional programs' are listed at the end of the book which can be copied and RUN at any stage.

The book was written before the full BBC Manual was available but it is suggested that the two should be used in conjunction.

Easy Programming for the BBC Micro by Eric Deeson is published by Shiva Publishing Limited at £5.95 for 128 pages.
ISBN 0 906812 21 6.

TIRED OF TYPING?

Why type in thousands of bytes of BBC BASIC when you can buy all these programs ready to LOAD?

Ultima

It might look like a chessboard to you but it's like no game of chess that you've ever played! Supported by a superb Hi-Res graphics display of the board and the pieces Ultima challenges anyone to try this subtle variation on a theme.

It doesn't matter if you don't know the rules, the micro acts as gamesmaster while the two human players battle it out. You cannot buy an Ultima chess set, so if you want to learn how to play this is the only way you'll find out how much fun it is!

Gomoku

The classic strategy game of five in a row. Simple to play, easy to learn and very difficult to win!

The computer displays and manages the games board as well as taking the position of your opponent in this BBC implementation of the game that used to be played with stones on the floor!

St George & The Dragon

A high-speed graphics game in which you must manoeuvre bold St George through the woods and across the stream into the dragon's territory. Once there the dragon must be quickly slain before he can breathe fire on you.

If you manage to kill off the monster it becomes a race against the clock to get to the castle and free the villagers who have been sheltering there.

It's all fast and furious fun with excellent graphics that will appeal to the whole family.

In Chorus

The SOUND and ENVELOPE Facilities on the BBC Micro make it a very powerful music machine, provided you can figure out how to make them perform. The program given here is the supporting material for the article and generates the well-know Bach cantata, Jesu Joy of Man's Desiring. We've included it on the tape to save you keying in all those DATA statements and ending up with something that doesn't sound quite right!

Multitest

If you've ever wondered how you are going to justify your claims that you can write educational programs for the kids on your BBC Micro don't worry. We've done it for you. The idea is simple, it's just a multiplication test program but to give a more competitive edge the questions are timed and, for practice, there is the option of doing simple maths drills as well.

Life

Every computer deserves its game of Life and, as the BBC Micro is no exception, here is our version of the famous Conway simulation. So, get your gliders gliding and barberpoles spinning with our stunning new implementation.

Mars Lander

We must have all played with Lunar Lander type games at one time or other so why another? Well, it's easy to land on the Moon, there is only one sixth the gravity compared to Earth so we thought that we'd make the gravity variable. Not really satisfied with that we've added a full colour, Hi-Res Display, fully controllable spaceship and instrument read-outs too.

The Maze

Ever since *Computing Today* first published a three-dimensional maze game we have sought to go one better. Here is the ultimate in mind boggling mazes, a three dimensional three dimensional maze. You not only see the walls on either side but the floors and ceilings as well and you can move in any direction you like — even up and down.

Envelope Design

This utility program is fully documented in the text but, briefly, it allows you to design and listen to sound envelopes drawn on the screen. Once you are satisfied that you've got it right it prints out all the necessary parameters for the SOUND and ENVELOPE commands so you can simply insert them into your BASIC program.

A Graphic Demo

How often have friends and visitors looked at your BBC Micro and said, "Well, show us what it can do!"? And, equally, how often have you wished that you had a simple yet effective demonstration program to hand? Wait no longer for here is a complete demonstration program which will show off the BBC's graphics to the full. You could, perhaps, even combine it with the **In Chorus** program to show off the music as well.

Joystick Suite

A set of three, inter-related programs from the text of this issue which enable you to calibrate and use any of the many joysticks currently on the market for the BBC Micro. The first program is simply to identify which stick is which and whether it is working correctly, the second computes the operating parameters of the sticks and the third provides an interactive drawing and graphics package. If you want to use joysticks for games or other purposes these will prove invaluable utilities to posess.

Disassembler

One of the strong features of the BBC Micro is the way that you can add assembly language routines to BASIC programs. However, there comes a time when you get a program with machine code built in and you simply can't work out what it does. At this point you need a simple disassembler and that is just what we've provided here.

BBC TAPE 1
(St George, Ultima, Gomoku, In Chorus) ☐

BBC TAPE 2
(Mars Lander, The Maze, Life, Multitest) ☐

BBC TAPE 3
(Joystick suite, A Graphic Demo, Disassembler, Envelope Design) ☐

Tapes are priced at £9.99 each (all inclusive) or you can order two tapes for £15.99 or all three for £21.99.

ASP Software
ASP Ltd
145 Charing Cross Road
London WC2H 0EE

I enclose my Cheque/Postal Order/International Money Order for: (delete as necessary)
£. . . . .(made payable to ASP Ltd)
OR debit my Access/Barclaycard
(delete as necessary)

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Please use BLOCK CAPITALS
Name (Mr/ Mrs/ Miss) . . . .
Address . . . .
. . . . **Postcode** . . . .
Signature . . . . **Date** . . . .
Please allow 21 days for delivery

PROGRAMPOWERMICROPOWER

GREAT BBC PROGRAMS FROM BRITAIN'S LEADING SOFTWARE HOUSE

SWOOP

SWOOP

ALIEN DESTROYERS

CHESS

SWOOP (B) £6.95 — the NEW GALAXIANS IT'S HERE AT LAST!! Galaxian-style, machine code arcade game. THIRTY screaming, homing, bomb-dropping, explosive egg-laying BIRDMEN, swooping down in ones and two's to destroy your laser bases. The exploding eggs feature makes a normally difficult game into a challenge 'par excellence.' Each new screen means increased difficulty. Bonus bases, score display, high-score and rankings are, of course included.
YOUR WAITING IS OVER!!

ALIEN DESTROYERS (B) £6.95
Sensational, high speed 'INVADERS' program with an abundance of features. Brilliant use of sound and graphics. 48 strong Alien Fleet of three different types plus Mothership scoring mystery bonus. Choice of six alien speeds and three bomb speeds. Vertical, angled and exploding missiles. Options to replace defences and suppress new fleet advances. Bonus bases awarded each new sheet. Scoring according to overall difficulty level. Ongoing display of score and hi-score. End of game rankings of top five scores.

This program has many unique extras e.g. 'battle analysis' showing the number of each alien type shot down, how many motherships destroyed, the number of sheets cleared, the shots fired, the percentage of hits made and the number of bases lost.

CHESS (B) £6.95
Our excellent machine code program — now with superb MODE 1, colour graphics. Six skill levels, play black or white, illegal moves rejected, 'en passant', castling, take-back of moves, and display of player's cumulative move-time. Options include Blitz Chess where you must move in 10 seconds, set-up of positions for analysis, replay of a game just played and saving of part completed games on tape. On loading, a 1972 Spassky/Fischer game can be replayed.
NOTE: Model A Version still available at only £4.95.
If you wish to upgrade your Model A version please return your tape, together with £2.50 plus V.A.T.
(Special Offer does not apply for Model A upgrade)

Other B.B.C. programs available:
Galactic Commander (B) £6.95/
Laser Command (B) £6.95/
Adventure £6.95/
Cowboy Shoot-Out (B) £5.95/
Filer £8.95/Micro Budget £6.95/
World Geography (B) £5.95.
Timetrek (B) £6.95/Spacemaze (B) £5.95/Martians (B) £5.95/Astro Navigator (B) £4.95/Star Trek £4.95/Maze Invaders (B) £4.95/Footer (B) £6.95/Munchyman £5.95/Seek £5.95/Eldorado Gold (B) £5.95/Cat & Mouse £4.95/ Mastermind £3.95/Reversi 1 £4.95/Reversi 2 (B) £4.95/ Roulette (B) £4.95/Gomoku £3.95/Zombies £3.95/ Dissassembler £5.95/Constellation (B) £5.95/ Where? (B) £5.95/Junior Maths Pack (B) £5.95.

WE *Guarantee* THAT ALL OUR ADVERTISED PROGRAMS HAVE BEEN COMPLETED AND ARE READILY AVAILABLE

WRITTEN ANY PROGRAMS!
WE PAY 20% ROYALTIES
FOR DRAGON, SPECTRUM
BBC, ATOM PROGRAMS

WE ARE AUTHORISED DEALERS FOR ACORN ATOM, BBC MICRO & DRAGON 32

SPECIAL OFFER
Deduct £1 per cassette when ordering two or more.

MICRO POWER LTD.
8/8a REGENT STREET,
CHAPEL ALLERTON,
LEEDS LS7 4PE.
Tel: (0532) 683186

Please add 55p order P & P + VAT at 15%
PS3

Please Note:
All programs are now available at all good dealers or direct from MICRO POWER LTD.

PROGRAMPOWERMICROPOWER