JulSrc and *CharMk*, which assemble the machine code and create a compressed character set respectively. *JulSr* must be run with PAGE at &3000. The program *Julia8* is the Basic front-end, and can be used to view the fractal.

When run, you are prompted to enter values for constants 1 and 2 (phone numbers are always the best.) Some good examples to try are: (0.2511, 0.5284), which zooms in where the spiral arms meet; (0.0843, 0.672), which zooms in on the 'bullet-holes'; (-1.2, 0.2); and (0.452, 0.401)

The fractal will then be displayed at the very lowest resolution (and therefore fastest speed); this takes roughly 30 seconds. This can be considered the 'preview mode'; after all, it would be very disconcerting to leave the computer running for 20 minutes on a high resolution, only to find that your constants had produced a very boring fractal.

Once the preview has finished, you are presented with a flickering rectangle cursor. At this stage, if you think the preview looks interesting, the keys H or M can be used to provide high- or medium-resolution respectively (the low-resolution mode can be reselected by pressing L). The number of iterations can also be changed at this stage with the left and right arrow keys; the greater the number of iterations, the greater the accuracy, and the more detailed the display is, but don't hold your breath.

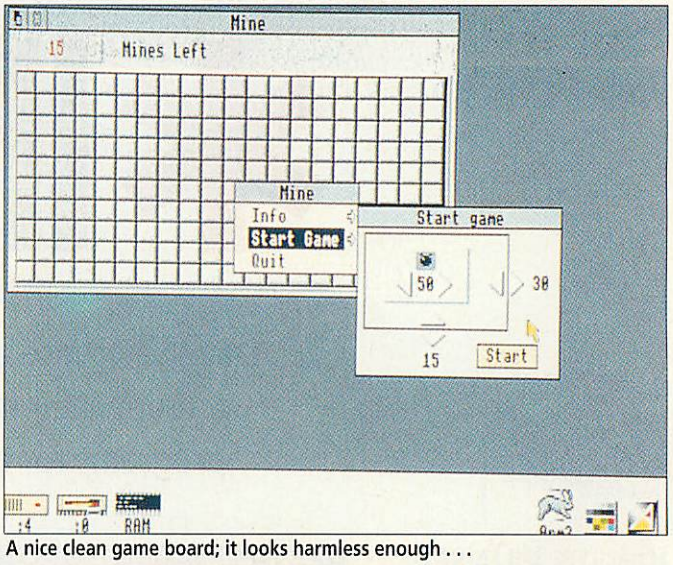
If you now want the fractal to be redrawn using your new settings, just press R. Specific sections of the fractal can be zoomed up and examined in detail, using the rectangle cursor; this can be moved (as normal) with Z,X,: and ?, using SHIFT to accelerate movement. The size of the box – and thus the magnification – can be altered using the up and down arrow keys. When you are satisfied, press RETURN to display the zoomed image.

To enter new values for the constants, just press ESCAPE while the cursor is present. Screens can be saved by pressing S; the filenames always default to *JulPicA*, *JulPicB* and so on during each session; it's therefore advisable to rename precious screens as soon as possible.

MINED BOGGLER

Program: Makemine, Mine8
Description: Single-player thinking game
Authors: D Walters & A Fawcett; eight-bit conversion by DL
Machine: 32-bit desktop version; eight-bit conversion
Listings: 32-bit: 270 + 690 lines Basic, eight-bit: 250 lines Basic

In the words of David Walters, this is a version of the best feature of Windows 3.1; which is, of course, the free game.



A nice clean game board; it looks harmless enough . . .

BEGINNERS' BIT

Last month in our beginners' section we looked at Basic commands for inputting data and detecting keypresses; GET, INKEY and so on. This month, we are looking at the machine-code equivalents.

Machine-code programs often need to receive input from the keyboard. This usually involves either waiting for a key (where you would use GET in Basic) or checking to see if certain keys are being pressed (where you would use INKEY; in Basic). In fact, the calls you use to check keys in a machine-code routine are the same that Basic uses and are very simple.

To GET a character from the keyboard on a 32-bit machine you use the SYS call OS_ReadC. This takes no parameters and returns in R0 the Ascii value of the key pressed. *GETmc32* shows the idea. The small loop at .getmc reads characters and then displays them with OS_WriteC. This continues until the escape character 27 is detected, after which the routine exits. The carry flag C is set if an 'error' occurs, such as ESCAPE.

On eight-bit machines, the routine OSRDCH (at &FFE0) has the same function and *GETmc8* shows how it is used. The character typed is returned in A. The small loop prints the character out by calling OSWRCH – the equivalent of OS_WriteC on 32-bit machines – and continues until ESCAPE is pressed. C is set if ESCAPE or another error occurs.

More complex input routines can be built up using OSRDCH or OS_ReadC. It is up to you to write an input routine to suit the sort of data being entered. There are calls provided to input a stream of characters. On eight-bit machines OSWORD 0 can be used; *INPUTmc8* shows its use. OSWORD is called with A=0 and X,Y pointing to a block. This contains the address and size of the buffer, and minimum and maximum allowable Ascii codes. Characters outside the allowable range are still displayed; they are just not entered into the buffer.

Users of 32-bit machines should use the call OS_ReadLine to get a line of text and *INPUTmc32* demonstrates the SWI. R0 points to the buffer, R1 is the maximum length, R2 and R3 are the lowest and highest allowable values. Risc OS provides many calls to assist the processing of data entered. For example, there are calls to convert strings to numbers. These we will no doubt cover in a future Beginners' Bit. Eight-bit users have to do most of the work themselves, so it is often best to do the fiddly things (like parameter checking) in Basic and use machine-code only when speed is of the essence.

Checking whether or not certain keys are being pressed is a common feature of many machine-code programs. Games rely on quick key detection. Last month we saw how INKEY could be used with either a positive parameter or a negative one. In the first case, a GET is performed but with a time limit. But with a negative value, INKEY will check only whether one particular key is being held down and is therefore very quick.

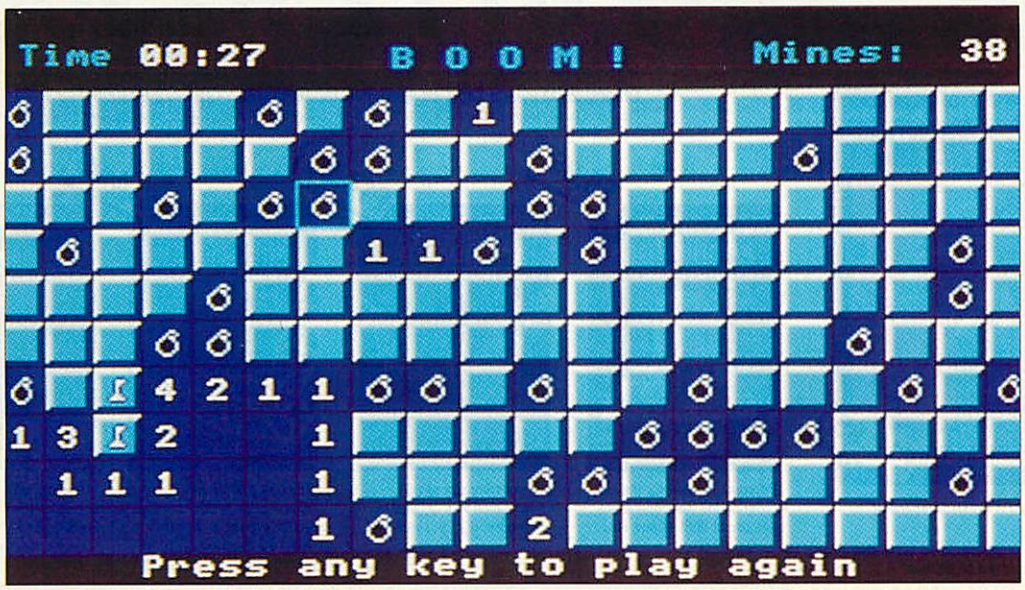
In both cases, the INKEY command actually uses the OSBYTE call 129. *INKEYmc8* and *INKEYmc32* wait for a key with a time delay. On eight-bit machines, A=129, and X and Y contain the low and high bytes of the delay (in centiseconds). If on exit Y=0, X holds the key pressed. Y=&FF means no key was pressed in the time limit. Y=&1B if ESCAPE was pressed. The 32-bit call is exactly analogous. On entry, R0=129 and the delay is in R1 and R2. On exit, R2=&FF if no key was pressed. If R2=0, R1 contains the key.

If R2 (or Y) contains &FF on entry, key R1 (or X) is checked. The correct INKEY- values for keys can be found in the *Advanced User Guide* or PRMs. Alternatively, run the short program *INKEY-*. This will display the INKEY- values for any keys pressed. The first (negative) number is as would be used in a Basic INKEY command. The second is 256 plus this value and is what would be loaded into R1 (or X) prior to an OSBYTE call. After calling OSBYTE R1 and R2 (X and Y) contain &FF if the key is being pressed.

As an example. *Bat32* and *Bat8* show a simple game bat, moved left and right with Z and X. The code for eight and 32-bit machines is very similar. In *Bat32* the bat position is held in R4, whereas it is stored at batpos in the eight-bit version.

The Bat programs also show how to detect that special key; ESCAPE. 32-bit users have a SWI call, OS_ReadEscapeState, that returns with carry set if ESCAPE has been pressed. Eight-bit users will find the easiest way to check for ESCAPE is to look at the top bit of location &FF. It is set when ESCAPE been hit, so use the following:

```
BIT &FF
BMI escape_pressed
BPL carry_on
```



. . . but certain things are lurking under cover

David has supplied us with a fully Risc OS compliant application and I have spent the last day or so beavering away to produce the eight-bit version. The idea of the game is very simple; uncover all the safe squares on the grid and place a flag over each one of the suspected mines.

You have as many flags as there are mines and the game is over when you have used all your flags (or have stepped on a mine). The actual positions of the mines will only be revealed if you successfully locate them all or, *in absentia*, if you have a little accident . . .

Clicking on a square with Select, or pressing SPACE in the eight-bit version, will 'uncover' a square. If it's a mine you're dead; if not, a number is placed in the square telling you how many of the eight adjoining squares contain mines. All adjacent non-mined squares will also be uncovered, as will their neighbours and so on. Clicking in the centre of a clear area will therefore reveal the extent of the safe patch.

Clicking with Adjust (RETURN on eight-bit) places flags. One click changes to a flag, a second to a '?'(if you're not so sure), a further click changes it back to uncovered. Flagging a square is only for your benefit; you won't know if you're right until the end. The three keys F, M and C can be used on the eight-bit version to place a Flag, a Maybe or RETURN to Clear.

Both versions allow you to alter the size of the field and the number of mines. The 32-bit version uses a menu option, and in the eight-bit version you edit the numbers at the top of the screen with the cursor keys. SPACE or RETURN starts a new game.

There are two listings for 32-bit users. First create a directory called Mine and enter and save RunImage within it. Next, enter and run *MakeMine* in Mine – this will create all the other files required. Double-click on Mine to play. For the inquisitive, *Mine8* is the subject of this month's IN DETAIL box.

MAZE CHALLENGE RESULTS

We had an excellent response to our Maze challenge (September 1992). To put you all out of your misery, the top ten maze solvers are given in the box below. John Atkinson of Blandford Forum in Dorset won the £25 prize. Congratulations to him and the other nine shown, and a big thank you to all who entered and made the standard so high.

The challenge was to write a set of procedures to solve ten unseen mazes, produced using the randomly chosen seed; 8507. John's winning entry managed to solve the ten random mazes in just 1352 moves. Sam Lindley came a good second with 1570 moves.

All but one of the entries tackled all ten mazes successfully. Authors wrote their entries on a variety of machines; some eight-bit, some 32-bit. André Moerenhout of the Netherlands wrote his on a PC using PC BBCBASIC(86) and the winner, John Atkinson, used his Master 128 with Cambridge co-processor (running bas32f).

Many entrants said they felt a total of ten mazes wasn't enough to do their programs justice and, on reflection, we would have to agree. Our original choice of ten was based on a maximum allowable solution time of 20 minutes per maze. To test 100 mazes could therefore have taken up to 33 hours and, when multiplied by the number of entrants, easily outstretched the number of machine-hours available.

Still, a knockout system might have been fairer; testing all on 30 mazes, eliminating half, testing the rest on 60, eliminating some more and so on until a clear winner is found. We'll bear all your comments in mind for next month's challenge.

To give some of the runners up a chance to show off their solutions, we tested the top ten on 100 mazes and the results are also shown in figure 1. As you can see, most of the top ten managed all 100 in less than 19,000 moves and several in less than 17,000. By way of acknowledgement, we are awarding special prizes of £5 each to Mike Cumpstey, Jeffrey Almeida and David Radford.

On the yellow pages you will find John Atkinson's solution together with Mike Cumpstey's routines. Jeffrey Almeida and David Radford both produced rather long sets of procedures – including machine-code – but we have put them on this month's disc for those who are interested.

Also in the yellow pages is *BAU*, our own attempt at the problem which we wrote while testing the original competition program. Incidentally, although we normally edit programs to bring them into 'style', we have left all Maze routines in their original form so that you can see the variety of approaches different programmers adopt when tackling the same problem.

- **First Prize (£25):** John Atkinson. Like many solutions, John's works in two parts. We know the amulet is hidden on the edge of the maze and our man starts in the middle. The strategy is to therefore (a) find the edge and then (b) tour the edge. As with all the better entries, known dead ends are ignored.
- **Special Prize (£5):** Mike Cumpstey. We know very little about how Mike's entry works – only that it works very well and solved 100 mazes in the fewest moves.
- **Special Prize (£5):** Jeffrey Almeida. Jeffrey came ninth in the competition proper but his routines solved 100 mazes in only 16,382 moves. The program rates squares as 'useful' or otherwise. If there are no useful squares in the man's immediate vicinity, the useful square reachable in the fewest moves is found and the man is directed towards it.
- **Special Prize (£5):** David Radford. David's routines eliminate known dead ends by 'walling them off'. He tells us a sort of 'flood fill' is used to mark off more complicated dead ends. As with Jeffrey Almeida's solution, a pathfind-

TOP TEN MAZES

MAZES:	10	100
1 John Atkinson	1,352	18,628
2 Sam Lindley	1,570	18,242
3 Chris Saunders	1,708	19,550
4 Colin Granville	1,736	21,244
5 Gareth Moore	1,766	16,862
6 Phillip Rogers	1,782	20,340
7 David Marples	1,808	17,450
8 Mike Cumpstey	1,816	16,176
9 Jeffrey Almeida	1,824	16,382
10 David Radford	1,914	16,239

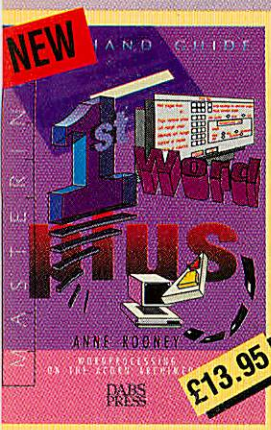
ESSENTIAL READING MATERIAL FOR THE ACORN ARCHIMEDES *from*

DABS PRESS

THE PREMIER ACORN BOOK PUBLISHER

0772 623000

Offices: 250 Leyland Lane, Leyland, Preston. PR5 3HL. Tel: 0772 623000. Fax: 0772 622917
Proprietor: David Atherton. All prices include VAT (0% on books), and UK postage. Access/Visa accepted. Foreign add £6.00 surface, £18 air.

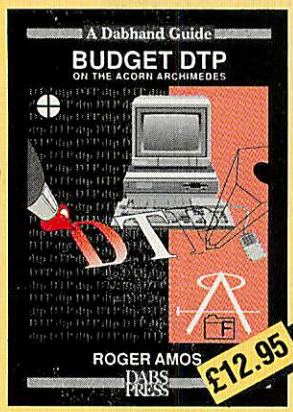


£13.95

step-by-step guide that takes you through all
he features of 1st Word Plus on the
rchimedes, from installation to mail-merge,
Mastering 1st Word Plus is comprehensive,
horough and easy to read.

ncludes: • How to use a word-processor
• How to type, edit, style and layout text
• Using graphics
• Using 1st Mail
• Using 1st Word Plus with other applications

isc available at £7.95 inc. VAT or £21.90 for
ook and disc.



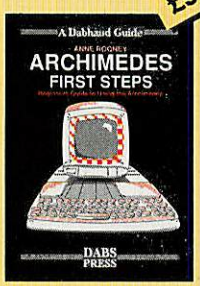
£12.95

"using the bright & brash approach of Roger Amos"

"a book that any impoverished but enthusiastic publisher, should not be without"
Jerry Glenwright - ACORN USER

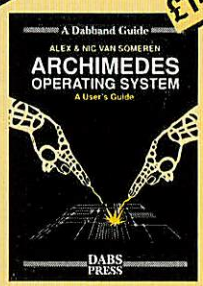
"offers all the help you will need to get you producing DTP documents on a shoestring"

"For the paltry sum of £12.95 this book could well save you over £100"
Paul Gaunt - ARCHIMEDES WORLD



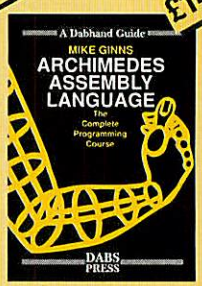
£9.95

Introductory book for the Arc, covering the basic operations and use of !Edit, !Draw, etc.



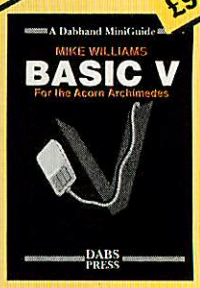
£14.95

Useful summary of OS information with detailed examples. (£21.95) with disc.



£14.95

The only tutorial and reference on ARM assembler on the market, with many RISC OS examples. (£21.95 with disc.)

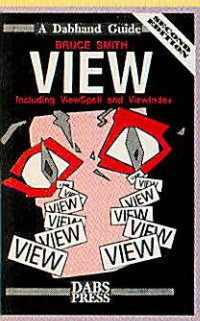


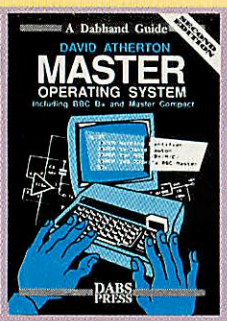
£9.95

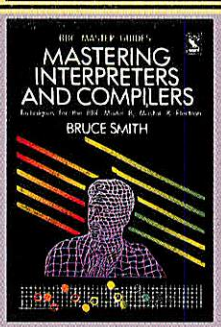
A practical guide to programming in BASIC V on the Acorn Archimedes, with a wealth of easy-to-follow examples.

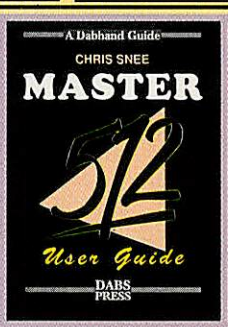
New Titles Coming Soon!!

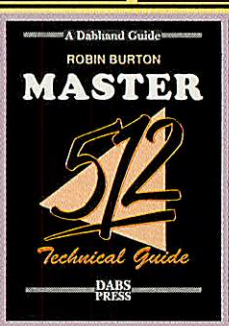
A Guide to IMPRESSION II	Graphics on the Arm	Basic WIMP Programming
by Anne Rooney	by Roger Amos	by Alan Senior

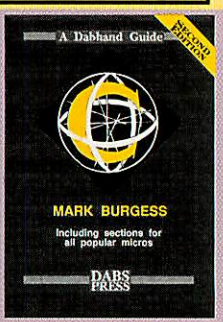












BBC SOFTWARE

lyperdriver The ultimate printer driver
IOM, with over 80 commands, a built-in
ILQ character set, WYSIWIG previewing,
ccess from View, Wordwise, Interword or
ASIC. 100 page manual. For B/B+/E/M/C.
rice £29.95 ROM, £24.95 disc for SRAM.

Minidriver As HyperDriver but for Mini
ffice II. Includes Viewdata terminal. £24.95
IOM. SRAM version £19.95.

los Plus Utility ROM for Master 128s only
xing EDIT and CLOSE#0 bugs, ADFS
ormat, verify and backup in ROM,
ackup/compacting in SRAM, alarm clock
nd configurable startup facility and much
nore. Price £12.95 ROM, £7.95 SRAM.

idewriter Pop-up notepad for SRAM
sers (any SRAM machine), works with any
oftware. Price £7.95.

Fingerprint SRAM or main memory 6502 machine code monitor/disassembler/memory editor. For B/B+/E/M/C/ Price £9.95.

Conversion Kit Ready-made 6502 assembler routines, for learning or development. Price £7.95

BBC BOOKS

View Dabhand Guide Bruce Smith's comprehensive guide to View word-processor. "For those who want a complete thorough and readable guide to View, then Bruce Smith is your man" (Beebug). £12.95 or £17.95 with disc.

Master Operating System David Atherton's definitive reference work including the famous 'differences between all eight-bit models' section used by countless programmers to ensure compatibility across the full eight-bit range. £12.95 or £17.95 with disc.

Mini Office II Guide Detailed tutorial by Bruce Smith and Robin Burton for the BBC/Master versions of the software. Price £9.95 or £14.95 with disc.

Mastering Interpreters and Compilers Fascinating Bruce Smith title on creating high level languages. £14.95 with free disc.

MASTER 512

Master 512 Shareware Collections Two collections of PC shareware, all tried and tested on the 512. Includes WP's, spreadsheets, databases, games etc. Five full 800k discs in each. Each collection normally £29.95, special offer £25 for both.

Master 512 User Guide Full instruction for using the 512 and DOS Plus, with tips on soft-ware compatibility. £9.95 or £14.95 with disc

Master 512 Technical Guide The companion guide with full 512 programming information and hardware expansion projects. Price £14.95 or £19.95 with disc.

ARCHIMEDES SOFTWARE

Instigator Utility system providing disc sector editor, memory editor, disassembler, command line archiving, and much more. Price £49.95

Arc PC Emulator Shareware Similar to Master 512 Shareware but for the Arc PC Emulator. Two collections of five discs each. Normal Price £34.95. Now on special offer, both collections for £25.

ARCDFS Very popular program to provide full DFS facilities on desktop or !65Host /!65Tube. *(Not A5000 compatible).* Price £29.95

ARCHIMEDES/BBC BOOKS

C: A Dabhand Guide Massive 512-page complete guide to C programming. No previous experience required. Arc & BBC sections £14.95 or £21.95 with disc.

OTHER BOOKS

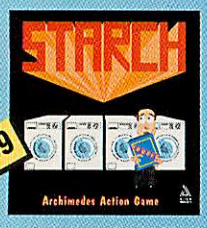
Z88: A Dabhand Guide Introduction to the Z88, by the designers of the machines own software. £14.95

Z88 Pipedream Guide John Allen's detailed work on all aspects of Z88 PipeDream. Good explanations of printing. Price £14.95

Psion LZ OPL Guide Ian Sinclair's guide to OPL programming on the LZ series of Organiser machines. £12.95

Games Action from Alien Images

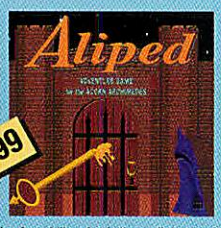
The Ideal Christmas Gift at NEW LOWER PRICES!!



£13.99

arry and Dave manage the laundry firm, but things have been
ng wrong. They work all night but will they put things right?'
ulti level game, high quality graphics, simple key controls for
sy movement of characters. Needs 1MB. RISC OS
mpatible.

great deal of fun" "fiercely competitive"
the accompaniment of decent music, effects and
mourous speech"
Duncan Evans - ARCHIMEDES WORLD



£13.99

"Aliped - wing footed like a bat is the spell that has been put on you by the evil step-mother of your fair princess Natasha". Smooth multi-directional scrolling castle, stunning use of stereo sound, high quality graphics. Needs 1MB. RISC OS compatible.

"Aliped is a sideways scrolling arcade adventure starring you as the bat-winged, red caped crusader."

"Decent looking game" "Worth investigating"
Duncan Evans - ARCHIMEDES WORLD

£13.99

The date is 2143 A.D., the universe has started to contract, causing the most inner galaxies' fine balance to be upset. Their only chance is you in your humble rescue ship. Multi level game with high quality graphics. Needs 1MB. RISC OS compatible.

ALERION Classic shoot-'em-up scroller.RISC OS compatible £14.95.
ARCENDIUM Draughts, Backgammon, Reversi and Quadline, now RISC OS compatible. £14.95.
ALIEN INVASION Classic Space Invaders £14.95
ALL-IN BOXING Realistic boxing game. 1 or 2 player. £14.95.

Orders to Dabs Press Publishing, FREEPOST (PR1327)
Leyland, Preston PR5 3BR (Phone 0772 623000)

Please send me:_____

_____

_____

Amount £ _____ Cheque/POenclosed,

Access/Visa No._____

Expiry date: _____

Name _____

_____

Address _____

_____

AU.1.93

ing routine is employed to take the man by the quickest route to the nearest square that may lead to the edge.

● **Our solution:** *BAU*'s solution performed reasonably well over 100 mazes and – by some fluke – was beaten only by John Atkinson's over the 10 that count. We include it as an example of an 'empirical' approach to the problem. At each stage, a recursive procedure PROCcount is called to weight each of the possible exits to the square according to how quickly they are likely to lead to the edge. If none are promising, you must be up a dead end, so your steps are retraced. The value 1.4 at line 470 was found by trial and error.

PEARLS OF WISDOM

Programs: Pearls, Pearls2
Description: Graphic demos
Author: Jan Vibe
Machine: 32-bit
Listings: 90, 100 lines Basic

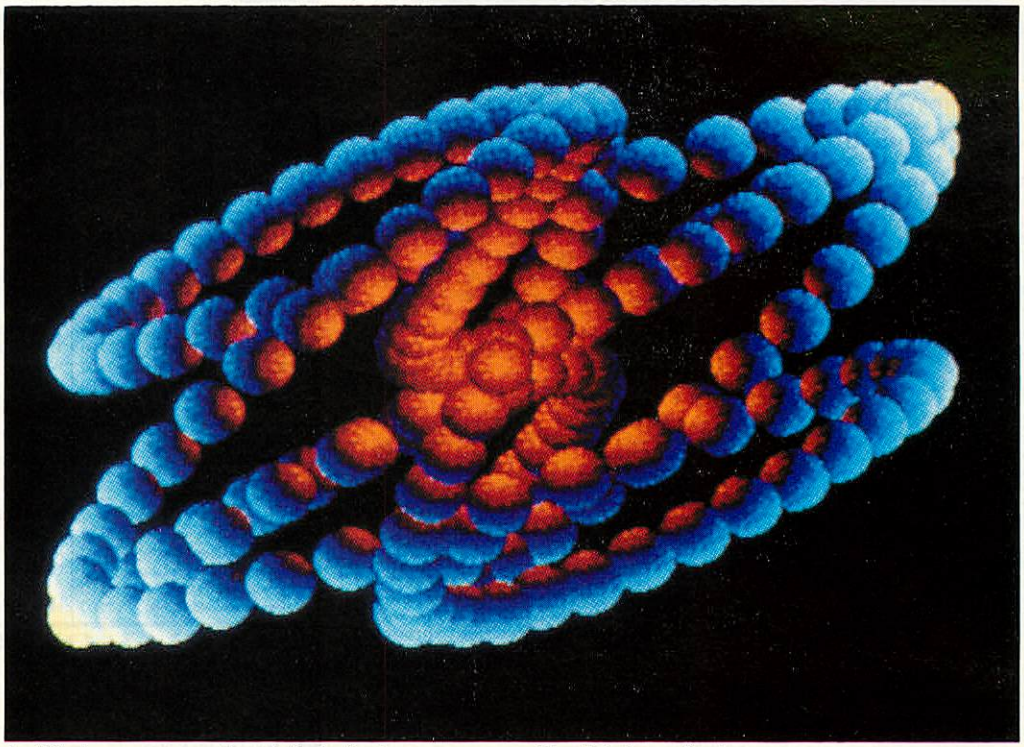
Two graphic quickies now from *INFO regular Jan Vibe. They both use the shaded ball that Jan introduced last month.

Pearls creates a ball and cyclès the colours to produce a dramatic effect. *Pearls2* uses a fixed colour ball but employs several screen banks for its animation.

TWIST AND SHAPE

Program: Parallel, Twist
Description: Sprite demo
Author: DL
Machine: 32-bit, Risc OS 3
Listings: 50, 100 lines Basic

One of the many areas to be enhanced in Risc OS in its latest incarnation is sprite plotting. It is now possible to transform sprites as they are plotted. A new OS_SpriteOp (number 38) allows you to supply either a transformation matrix (in a similar way to drawfile transformations) or a destination parallelogram. According to the PRMs, it may be possible to supply an arbitrary quadrilateral in some future release, but current versions are limited to linear transformations. The output from this call will cover exactly the same area on screen as a call to Draw_Fill would given the same matrix.



Jan Vibe's program goes for relatively simple curves: a case of Pearls before spline?

IN DETAIL

This month we take the lid off *Mine8*, examine it and delve around. Inside we find some rather unusual 'sprites', recursion, boolean algrebra and a few ideas on saving memory.

The first area of technical interest is the way that the program draws the various 'tiles' that make up the game board. Covered – meaning non-touched – squares are represented with a raised bevelled-type effect, complete with a flag or a question mark as appropriate. Uncovered squares appear as either plain blue or they come equipped with a number between 1 and 8.

It would be normal, in a game like this, to store these graphics as bit image sprites somewhere in memory and have a simple piece of machine code to plot them on the screen. *Mine8* neatly sidesteps this set-up for simplicity's sake and, instead, stores the VDU sequences necessary to draw the various tiles in string variables. flat$ and tile$ are initialised in lines 360 and 370 by means of FNread and the small collection of DATA statements at the end of the program.

The process of PRINTing these strings will display, at the graphics cursor, a blue or a bevelled tile respectively. The numbers, flags or question marks are added afterwards in a similar manner. Have a look at lines 380-440 and PROCtile at line 720. A number of functions (gcol, plot, w and b) have been defined to make this initialisation as plain as possible.

The board is stored in memory as a byte array called w. It is defined in line 310 to be two elements bigger in each direction. This allows space for a 'no man's land' border around the outside of the actual play area.

This saves an awful lot of checking in various routines for falling off the board into hyperspace. The minefield is set up in PROCblank. First the play area is cleared and then random positions are chosen for all the mines. As each mine is placed the surrounding eight squares are incremented by one to reflect the number of adjacent mines. Notice that no boundary checking is needed for this because of the screen border.

The uncovering of squares is performed by the recursive procedure PROCtread. This examines the square and if it has already been trodden on (line 1770) nothing happens. Otherwise t will contain 0 (for clear), 1-8 (meaning that many surrounding mines) or the 'magic' number 9 (meaning a mine has been hit).

If this is the case the player is dead and this is flagged (line 1780). Line 1790 reveals the contents of the square to the player and also marks the square as trodden on. If the square was totally clear then the recursion kicks in and calls PROCtread for all surrounding squares.

Boolean algebra crops up twice in PROCmark to shorten some of the lines. In line 1910, the value of marks (the number of flags the player has left) is decremented if a flag is placed (the +(m=flag)) and incremented if a flag is removed (the -(s=flag)).

Variable s is the old 'flag status' of the square and m is the new one. Line 1920 keeps track of the number of bombs discovered so far (this figure is never revealed to the player though). Here t is contents of the square, thus left is decreased if a mine is present and a flag is placed and increased, if one is removed.

The final points worth noting are the two measures taken to increase the available memory. First, PAGE is lowed in line 80 and, second, nine lines are 'stolen' from the screen (lines 240-290) which frees over 5½K. Program lines 260 and 270 move the start of screen address and line 280 blanks off the reclaimed memory so that the screen isn't cluttered up with rubbish.

```
3280 IF b%=2 PROCclosedown
3290 ENDPROC
3300 :
3310 DEF PROCfreeablock(RETURN loc%)
3320 IF loc%>0 THEN SYS &1D,3,heapd%,lo
c%
3330 loc%=-1
3340 ENDPROC
3350 :
3360 DEF FNgetablock(size%)
3370 LOCAL loc%,ok%,heapfull%,largest%
3380 heapfull%=FALSE
3390 REPEAT
3400 SYS &1D,1,heapd% TO ,,largest%
3410 IF largest%>=size% THEN
3420 SYS "OS_Heap",2,heapd%,,size% TO ,
,loc%
3430 ok%=TRUE
3440 ELSE
3450 heapfull%=FNtrytoextendheap
3460 ENDIF
3470 UNTIL ok% OR heapfull%
3480 IF heapfull% THEN ERROR 17,"Could
Not Get Block"
3490 =loc%
3500 :
3510 DEF FNget_line(f%)
3520 IF NOT(EOF#f%) THEN =GET$#f% ELSE
=""
3530 :
3540 DEF PROCinitheap
3550 heapd%=HIMEM
3560 heapsize%=32*1024
3570 SYS "Wimp_SlotSize",-1,-1 TO appsi
ze%
3580 SYS "Wimp_SlotSize",appsize%+heaps
ize%,-1
3590 SYS &1D,0,heapd%,,heapsize%
3600 ENDPROC
3610 :
3620 DEF PROCinit_task(t$)
3630 LOCAL t%
3640 DIM t% 12, err_blk% 255
3650 $t%="TASK"
3660 SYS &400C0,200,!t%,t$
3670 alias$="<Mine$Dir>"
3680 task$=t$
3690 ENDPROC
3700 :
3710 DEF FNtrytoextendheap
3720 LOCAL heapfull%,new%
3730 SYS "XWimp_SlotSize",appsize%+heap
size%+32*1024,-1 TO ,new%;heapfull%
3740 heapfull%=((heapfull% AND 1)=1)
3750 IF new%=appsize%+heapsize% THEN he
apfull%=TRUE
3760 IF NOT heapfull% THEN
3770 heapsize%+=32*1024
3780 SYS &1D,5,heapd%,,32*1024
3790 ENDIF
3800 =heapfull%
3810 :
3820 DEF PROCwimp_poll(mask%)
3830 LOCAL r%
3840 REPEAT
3850 SYS &400C7,mask%,q% TO r%
3860 IF r% PROCaction(r%)
3870 UNTIL FALSE
3880 ENDPROC
3890 :
3900 DEF PROCaction(r%)
3910 CASE r% OF
3920 WHEN 2:PROCopen_window(q%)
3930 WHEN 3:PROCclose_window
3940 WHEN 4:PROCpointer_leaving
3950 WHEN 5:PROCpointer_entering
3960 WHEN 6:PROCmouse_click(q%!8,q%!12,
q%!16)
3970 WHEN 9:PROCmenu_selection
3980 WHEN 17,18:PROCreceive
3990 ENDCASE
4000 ENDPROC
4010 :
4020 DEF PROCload_templates
4030 LOCAL f%,n%,off%,ind%,win%,winsize
%,indsize%
4040 LOCAL start%,end%,pos%,i%,win_loc%
,w%
4050 LOCAL found%,size%
4060 DIM w% 4
4070 SYS "OS_File",5,alias$+".Templates
" TO found%,,,,size%
4080 IF found% THEN
4090 f%=OPENIN(alias$+".Templates")
4100 PTR#f%=16
4110 n%=0
4120 ind%=0
4130 WHILE FNget_word(f%)
4140 PTR#f%=PTR#f%+20
4150 n%+=1
4160 ENDWHILE
4170 PTR#f%=16
4180 DIM win_index% n%*28
4190 win_top%=win_index%+n%*28
4200 n%=0
4210 off%=FNget_word(f%)
4220 WHILE off%
4230 PROCtemp_info(f%,off%,winsize%,ind
size%)
4240 win_index%!(n%*28+12)=winsize%
4250 ind%+=indsize%
4260 win%+=winsize%
4270 PTR#f%=PTR#f%+20
4280 off%=FNget_word(f%)
4290 n%+=1
4300 ENDWHILE
4310 CLOSE#f%
4320 DIM win_store% win%+1000,indir% in
d%+1000,win_buff% &1000
4330 win_loc%=win_store%
4340 SYS "Wimp_OpenTemplate",,alias$+".
Templates"
4350 start%=indir%
4360 end%=start%+ind%
4370 pos%=0
4380 FOR i%=0 TO (n%-1)*28 STEP 28
4390 $(win_index%+i%)="*"
4400 SYS "Wimp_LoadTemplate",,win_buff%
,start%,end%,-1,win_index%+i%,pos% TO ,,
start%,,,,pos%
4410 win_buff%!64=IconSpr
4420 win_index%!(i%+16)=win_loc%
4430 win_index%!(i%+20)=pos%
4440 win_index%!(i%+24)=0
4450 PROCcopy_block(win_buff%,win_loc%,
win_index%!(i%+12))
4460 win_loc%+=win_index%!(i%+12)
4470 NEXT
4480 SYS "Wimp_CloseTemplate"
4490 ELSE
4500 d=FNdialog("File 'Templates' not f
ound")
4510 ENDIF
4520 ENDPROC
4530 :
4540 DEF FNget_word(f%)
4550 w%?0=BGET#f%
4560 w%?1=BGET#f%
4570 w%?2=BGET#f%
4580 w%?3=BGET#f%
4590 =!w%
4600 :
4610 DEF FNgwp(f%,p%)
4620 PTR#f%=p%
4630 =FNget_word(f%)
4640 :
4650 DEF FNgsp(f%,p%)
4660 PTR#f%=p%
4670 =GET$#f%
4680 :
4690 DEF PROCtemp_info(f%,p%,RETURN win
%,RETURN ind%)
4700 LOCAL op%,v%,l%,i%,ics%
4710 op%=PTR#f%
4720 ind%=0
4730 IF FNgwp(f%,p%+56) AND 1<<8 THEN
4740 v%=FNgwp(f%,p%+76)
4750 l%=FNgwp(f%,p%+80)
4760 ind%+=l%
4770 IF v%<>-1 ind%+=LENFNgsp(f%,v%)+1
4780 ENDIF
4790 ics%=FNgwp(f%,p%+84)
4800 IF ics% THEN
4810 i%=p%+88
4820 FOR k%=0 TO ics%-1
4830 IF FNgwp(f%,i%+16) AND 1<<8 THEN
4840 v%=FNgwp(f%,i%+24)
4850 l%=FNgwp(f%,i%+28)
4860 ind%+=l%
4870 IF v%<>-1 ind%+=LENFNgsp(f%,p%+v%)
+1
4880 ENDIF
4890 i%+=32
4900 NEXT
4910 ENDIF
4920 PTR#f%=op%
4930 win%=ics%*32+88
4940 ENDPROC
4950 :
4960 DEF PROCcopy_block(A%,B%,C%)
4970 LOCAL copy%
4980 copy%=FNgetablock(20)
4990 P%=copy%
5000 [OPT 0
5010 LDR  r3,[r0],#4
5020 STR  r3,[r1],#4
5030 SUBS r2,r2,#4
5040 BGT  copy%
5050 MOV  pc,14
5060 ]
5070 CALL copy%
5080 PROCfreeablock(copy%)
5090 ENDPROC
5100 :
5110 DEF FNwin_name_ptr(w$)
5120 LOCAL p%
5130 p%=win_index%
5140 WHILE w$<>$p% AND p%<win_top%
5150 p%+=28
5160 ENDWHILE
5170 IF w$<>$p% p%=0
5180 =p%
5190 :
5200 DEF FNwin_hand_ptr(h%)
5210 LOCAL p%
5220 p%=win_index%
5230 WHILE h%<>p%!24 AND p%<win_top%
5240 p%+=28
5250 ENDWHILE
5260 IF h%<>p%!24 p%=0
5270 =p%
5280 :
5290 DEF FNwin_hand(w$)
5300 =!(FNwin_name_ptr(w$)+24)
5310 :
5320 DEF PROCcreate_game_window
5330 LOCAL x%,y%,temp%,i%,a%
5340 temp%=FNgetablock(rows*cols*32+102
4)
5350 win_ptr%=FNwin_name_ptr("main")
5360 IF win_ptr%!24<>0 THEN
5370 !temp%=win_ptr%!24
5380 SYS "Wimp_CloseWindow",,temp%
5390 SYS "Wimp_DeleteWindow",,temp%
5400 ENDIF
5410 PROCcopy_block(win_ptr%!16,temp%+4
,win_ptr%!12)
5420 i%=8
5430 FOR y%=0 TO rows-1
5440 FOR x%=0 TO cols-1
5450 a%=temp%+4+88+32*i%
5460 a%!0=16+40*x%
5470 a%!4=-112-40*y%
5480 a%!8=a%!0+40
5490 a%!12=a%!4+40
5500 a%!16=%11010+(3<<12)
5510 $(a%+20)="block"
5520 i%+=1
5530 NEXT
5540 NEXT
5550 temp%!88=i%
5560 FOR i%=4 TO 7
5570 a%=temp%+4+88+32*i%
5580 a%!4=a%!4-40*(rows-10)
5590 a%!8=a%!8+40*(cols-20)
5600 NEXT
5610 temp%!(4+4)=temp%!(4+4)-40*(rows-1
0)
5620 temp%!(4+8)=temp%!(4+8)+40*(cols-2
0)
5630 temp%!(4+44)=temp%!(4+44)-40*(rows
-10)
5640 temp%!(4+48)=temp%!(4+48)+40*(cols
-20)
5650 SYS "Wimp_CreateWindow",,temp%+4 T
O hand%
5660 win_ptr%!24=hand%
5670 !temp%=hand%
5680 temp%!28=-1
5690 SYS "Wimp_OpenWindow",,temp%
5700 PROCfreeablock(temp%)
5710 ENDPROC
5720 :
5730 DEF PROCcreate_window(win$,o%)
5740 LOCAL q%
5750 q%=FNgetablock(256)
5760 win_ptr%=FNwin_name_ptr(win$)
5770 IF win_ptr%!24=0 THEN
5780 SYS &400C1,,win_ptr%!16 TO hand%
5790 win_ptr%!24=hand%
5800 q%!0=hand%
5810 ELSE
5820 q%!0=win_ptr%!24
5830 ENDIF
5840 IF o% THEN
5850 SYS "Wimp_GetWindowState",,q%
5860 SYS "Wimp_OpenWindow",,q%
5870 ENDIF
5880 PROCfreeablock(q%)
5890 ENDPROC
5900 :
5910 DEF FNmouse_button
5920 LOCAL q%,b%
5930 q%=FNgetablock(256)
5940 SYS &400CF,,q%
5950 b%=q%!8
5960 PROCfreeablock(q%)
5970 =b%
5980 :
5990 DEF PROCload_menus
6000 LOCAL fp%,f%,f$,ic%,sub%,data%,l$
6010 LOCAL found%,size%
6020 SYS "OS_File",5,alias$+".Menu" TO
found%,,,,size%
6030 IF found% THEN
6040 DIM menu%(10),menu$(10)
6050 fp%=OPENIN(alias$+".Menu")
6060 DIM tp% 1000,menp% 1000
6070 mp%=menp%
6080 menu_cnt%=0
6090 l$=FNget_line(fp%)
6100 WHILE NOT(EOF#fp%)
6110 menu_cnt%+=1
6120 menu%(menu_cnt%)=mp%
6130 l$=RIGHT$(l$,LEN(l$)-5)
6140 menu$(menu_cnt%)=RIGHT$(l$,LEN(l$)-I
NSTR(l$,"<")-1)
6150 $mp%=LEFT$(l$,INSTR(l$,"<")-2)
6160 mp%!12=&00070207
6170 mp%!16=0
6180 mp%!20=44
6190 mp%!24=0
6200 mp%+=28
6210 l$=FNget_line(fp%)
6220 width%=0
6230 ht%=0
6240 WHILE NOT(EOF#fp%) AND LEFT$(l$,4)
<>"MENU" AND l$<>""
6250 ht%+=44
6260 f$=LEFT$(l$,5)
6270 f%=0
6280 ic%=&7000021
6290 sub%=-1
6300 data%=0
6310 bcol%=FALSE
6320 IF INSTR(f$,"t") f%=f% OR 1
6330 IF INSTR(f$,"d") f%=f% OR 2
6340 IF INSTR(f$,"w") f%=f% OR 4
6350 IF INSTR(f$,"m") f%=f% OR 8
6360 IF INSTR(f$,"l") f%=f% OR &80
6370 IF INSTR(f$,"s") ic%=ic% OR &40000
0
6380 IF INSTR(f$,"b") bcol%=TRUE
6390 IF INSTR(l$,">") THEN
6400 sub$=RIGHT$(l$,LEN(l$)-INSTR(l$,">")
-1)
6410 l$=LEFT$(l$,INSTR(l$,">")-2)
6420 IF f% AND 8 THEN
6430 sub%=FNwin_name_ptr(sub$)
6440 ELSE
6450 sub%=EVALsub$
6460 ENDIF
6470 f%=f% OR 8
6480 ENDIF
6490 mp%!0=f%
6500 mp%!4=sub%
6510 l$=RIGHT$(l$,LEN(l$)-5)+CHR$13
6520 len%=LEN(l$)
6530 IF len%>width% width%=len%
6540 IF bcol% THEN
6550 ic%=ic% OR VAL(l$<<28
6560 ic%=ic% OR 8+32
6570 CASE VAL(l$) OF
6580 WHEN 0,1,2,3,9,12,14,15:ic%=ic% OR
7<<24
6590 OTHERWISE:ic%=ic% EOR 7<<24
6600 ENDCASE
6610 width%=len%+2
6620 ENDIF
6630 IF len%>12 THEN
6640 ic%=ic% OR &100
6650 mp%!12=tp%
6660 mp%!16=-1
6670 mp%!20=len%
6680 $tp%=l$
6690 tp%=tp%+len%
6700 ELSE
6710 $(mp%+12)=l$
6720 ENDIF
6730 mp%!8=ic%
6740 l$=FNget_line(fp%)
6750 mp%+=24
6760 ENDWHILE
6770 l$=FNget_line(fp%)
6780 menu%(menu_cnt%)!16=width%*16
6790 IF bcol% menu%(menu_cnt%)!20=40
6800 IF menu_cnt%=1 main_height%=ht%+96
6810 ENDWHILE
6820 CLOSE#fp%
6830 ELSE
6840 d=FNdialog("File 'Menu' Not Found"
)
6850 ENDIF
6860 ENDPROC
```

Listing 5 – MakeMine

```
10 REM >MakeMine (Info6)
20 REM Create files for !Mine
30 REM Original files by D Walters &
A Fawcett
40 REM for 32-bit machines
50 REM (c) BAU January 1993
60 :
70 free%=(HIMEM-END-&8000) AND &FFFFF
000
80 DIM q% &100,w% free%
90 REPEAT
```

```
  100 READ file$
  110 IF file$<>"*" THEN
  120 PRINT file$
  130 READ type%,olen%
  140 IF type%=&FFF OR type%=&FFE OR typ
e%=&FEB THEN
  150 out%=OPENOUT(file$)
  160 REPEAT
  170 READ line$
  180 IF line$<>"*" BPUT#out%,line$
  190 UNTIL line$="*"
  200 CLOSE#out%
  210 SYS "OS_File",18,file$,type%
  220 ELSE
  230 line%=0
  240 d$=""
  250 i%=0
  260 WHILE i%<olen%
  270 b$=FNc
  280 IF b$>="a" AND b$<="z" THEN
  290 off%=-EVAL("&"+FNc+FNc)
  300 FOR k%=0 TO (ASC(b$)-ASC"a"+2)
  310 w%?i%=w%?(i%+off%)
  320 i%+=1
  330 NEXT
  340 ELSE
  350 w%?i%=EVAL("&"+b$+FNc)
  360 i%+=1
  370 ENDIF
  380 ENDWHILE
  390 SYS "OS_File",10,file$,type%,,w%,w
%+olen%
  400 ENDIF
  410 ENDIF
  420 UNTIL file$="*"
  430 END
  440 :
  450 DEF FNc
  460 LOCAL c$
  470 IF d$="" THEN
  480 READ d$,ch$
  490 line%+=1
  500 $q%=d$
  510 SYS "OS_CRC",0,q%,q%+LEN(d$),1 TO
crc%
  520 IF crc%<>EVAL("&"+ch$) PRINT"Error
 in data line ";line%:END
  530 ENDIF
  540 c$=LEFT$(d$,1)
  550 d$=MID$(d$,2)
  560 =c$
  570 :
  580 DATA "!Boot",&FEB,&38
  590 DATA | !Boot file for !Mine
  600 DATA
  610 DATA IconSprites <Obey$Dir>.!Sprit
es
  620 DATA *
  630 DATA "!Run",&FEB,&85
  640 DATA | !Run file for !Mine
  650 DATA
  660 DATA WimpSlot -min 128k -max 128k
  670 DATA IconSprites <Obey$Dir>.!Sprit
es
  680 DATA Set Mine$Dir <Obey$Dir>
  690 DATA Run <Mine$Dir>.!RunImage
  700 DATA *
  710 DATA "Menu",&FFF,&45
  720 DATA MENU Mine < main
  730 DATA m    Info > info
  740 DATA m    Start Game > start
  750 DATA 1    Quit
  760 DATA
  770 DATA *
  780 DATA "Sprites",&FF9,&1170
  790 DATA 160000001Øa04741100Ø098,50FB
  800 DATA a0C626C6F636Ba14a170002,9EDF
  810 DATA a1C08e0C07a282Ca2Ca0400,332E
  820 DATA 0Ce20b2730b2C11111111a0,1DB0
  830 DATA 433i0Ci0Cu18z30a5833333,AF85
  840 DATA 333a04b54b9831i74z98f37,807F
  850 DATA b8Cb4388h0C831i0Cj18j24u,9EF4
  860 DATA 18338888b49m60b9832z98x,C7AD
  870 DATA 98939939gA4993399i5Ci0C,71FB
  880 DATA 93h24c23f3Cb3Bf3C99j3Ci,F4E3
  890 DATA 98bF4z98u98A3AA3AgF4A43,29D1
  900 DATA 3AAiF4i0Ci24j18j30j48k9,397D
  910 DATA 834z98x9833BBhF4B3i0CBB,31B8
  920 DATA h18B33Bi0CBBBB3Bs3Cj48k,602C
  930 DATA 9835z98x98CCCCCCCh0Ck183,CE24
  940 DATA Ci68h24i0Ca30h3CC3k30i9,61BD
  950 DATA 836z98x98D3DDhF4c0BdF4D,4049
  960 DATA 33Di0CDDh24D33Dc1Ai0Cf3,58B0
  970 DATA Ci48k9837z98x98EEEEEEiE,9DC2
  980 DATA 8i0CE33Eh68b23eC8c17i0C,39A8
  990 DATA r18k9838z98x98F3FF3FgF4,9BED
 1000 DATA FF33FFi0Ch0Cz24r48k983F,4110
 1010 DATA z98i98fBF30bC81111B1BB1,CBF6
 1020 DATA B1111bF4a0CBBi1BBg0C11B,4FA0
 1030 DATA 1h18b17g18a1Bu18d4830m9,6803
 1040 DATA 8666C6167z98t98B1BB7Bg7,C6AD
 1050 DATA 4i0Ca9871i8Ci0C1177hA47,76A1
 1060 DATA 17717z98d986D696E65z98f,D227
 1070 DATA 98fCFbF4333377aDEa04d18,AC57
 1080 DATA 7777b0Fd1870c0Dd24a18i0,A7AB
 1090 DATA Cg18j30j5419877726F6E67,7F7E
 1100 DATA z98e98BBeD0BBBbA4BBc98BB,233D
 1110 DATA dA4BBa98a1EdB0BB77b1Ce9,D8B3
 1120 DATA 8a1Aq18i30d55c48j60b986,AEFE
 1130 DATA 36C656172z98e98vDCvF4z3,4B52
 1140 DATA 0z48bEC3Ca89726164696F6,30F5
 1150 DATA F6Eg9815eA417e983401000,1663
 1160 DATA 014aF41111111111137c06c0,9C39
 1170 DATA B713713g0D773733g177722,E9A3
 1180 DATA 33f1977272232e19a1722f1,74D6
 1190 DATA 9a17BBe19a2EB2BB2Bd32b1,5E03
 1200 DATA 7BBd19c17BBd19d17d19332,DDF4
 1210 DATA 3e0C0200113133e24001011,B8A9
 1220 DATA e19b1711d19d17d19d17c19,432E
 1230 DATA 22e17c32e17c32f17111F33,DCB0
 1240 DATA 03f17a4B03f17aFE03cFC00,DE6D
 1250 DATA 0000000000FFc07c06F0FF0,4D0F
 1260 DATA Fg12FFFFd07c17FFc19b2Aa,1D14
 1270 DATA 17c19a36b17c19a42c17b32,53DC
 1280 DATA FFe17b32FFf17a4BFFf17a4,441B
 1290 DATA Bg16g1Fe3Fj24j3Ck54j6Ck,D963
 1300 DATA 84j9CkB4jCCkE4fFC3C0200,9F1B
 1310 DATA 00726164696F6F6666bFCa0,E135
 1320 DATA F0015aFE04aFD1BaF92CaFD,9E11
 1330 DATA 3401000014aF91011111111,AF2
 1340 DATA 0113c06c0C0033g0C010033,B075
 1350 DATA b19c18001031b19b2401001,D4E3
 1360 DATA 111b19b30b40b19a3Cb17c1,527C
 1370 DATA 9a48d57a320110d17b32011,802F
 1380 DATA 0d6Eb32a7Bd17a4B0330c73,76C
 1390 DATA b89770710c73a95717701d7,9DED
 1400 DATA 3b1717a9Cd197177ba8d197,7679
 1410 DATA 717cB4bA5a17cC0bA5a17dC,1889
 1420 DATA C3313b17dD83133b17eE433,5A67
 1430 DATA b17eF031b1711710F000000,993D
 1440 DATA 00F0c06c0600FFFFg12F0FF,FFFE
 1450 DATA FFc07b24FFFFc19b2Aa17c1,FE55
 1460 DATA 9b36b17b32a42c17b32a4Ed,1539
 1470 DATA 17a4BF00Fe17a4BF00Ff17F,34ED
 1480 DATA FFFg16h13e3Fi24k3Cj54k6,6731
 1490 DATA Cj84k9CjB4kCCjE4fFC78aF,55D9
 1500 DATA C7074725F64656661756C74,4186
 1510 DATA b26000Ac06000015aF94CaF,5E2E
 1520 DATA Da040008d40FFbFAd49c649,E2CC
 1530 DATA 9a6F9907a74a04000500005,BD72
 1540 DATA 559a04A905a045A0155A9AA,963B
 1550 DATA 16a0815555969005415A401,62BF
 1560 DATA 000090060000401Ac5AA4aA,E2A9
 1570 DATA 873696768d720000a1Eg780,ADB2
 1580 DATA Bu7801DDDDDDaD4DD00DDaE,CB9D
 1590 DATA 5a04f78eAFaFA80eBAAA2Ac,59D5
 1600 DATA C9A0808002bF80880cC080A,847A
 1610 DATA AAAAAAAi10g20f30g40d95E,5504
 1620 DATA CaFC646563g62bA4b9Cb941,229
 1630 DATA FaFD2CaFD8CaFC0CaFC1111,A704
 1640 DATA 111111110044d0711c08a15,D0DD
 1650 DATA b10b1Ca18c2344d07a20d2A,E1AD
 1660 DATA a30c33b38b3Cf0Df1Ed5011,9A09
 1670 DATA 4444FFcE8FFFFc08a07b10b,D0A1
 1680 DATA 07a18c07FFd0E8d0Ch0Bh18g,8E81
 1690 DATA 28g38g48b58bEC696EzECiE,D45D
 1700 DATA C0041eEF11d09aFEc12bEAa,82FC
 1710 DATA 1B55cEB41e081152eF455dF,85C4
 1720 DATA C55d28a07b38b07a48c0711,4D66
 1730 DATA 00d07h96hEFgF0gF1zF2iF1,CD5E
 1740 DATA 148cEED4aFC646F776EfED0,14F0
 1750 DATA 2aFC06aFA04iEC80eEC01f2,831F
 1760 DATA 5aFCeF6dCCg0D55F5eFEaF9,C361
 1770 DATA 15F1cF9c0B11b0Cb27a0B11,8FE3
 1780 DATA 111CbFB54c0BbE0sBB0FF0h,F295
 1790 DATA C40F00f0D00a0Ce0DaCCa0C,F4B3
 1800 DATA c1AbD0b0Ca27gD47570hADz,E8DD
 1810 DATA D4cC70140gCE0110gCEa0Bf,C5E8
 1820 DATA CEc0BdDBe0BbE8g0B111143,8711
 1830 DATA 4155555555a04b07j8CmA4i,FA6F
 1840 DATA BCiD4iECgF6gFFcFBFF,6D3B
 1850 DATA "!Sprites",&FF9,&2E0
 1860 DATA 0100000010a04E4020000D4,5270
 1870 DATA a04216D696E65a14a170004,8E6C
 1880 DATA e1Cb0F07a282Ca2C80a310C,8DB9
 1890 DATA e20b2730i0CC011B1BB7B11,CA01
 1900 DATA 111133a54888888d46b61c5,DD3
 1910 DATA 003i141011BBBB17111131q,450E
 1920 DATA 14d14b6Fc281111b2883a4F,5404
 1930 DATA 887711433333333333h1471d,3B76
 1940 DATA 1477e143003a3C7777a6473,51BD
 1950 DATA a13d28FF08b50aA1a14f28F,935D
 1960 DATA F08c48a4Di148888bC63377,B016
 1970 DATA 7777o14b15g508888BBBBBB,6368
 1980 DATA f40q14eA0c48c4Dd0AdF1dF,F06F
 1990 DATA 7g0CF4FFFFe18r2DF0FFFFF,89F9
 2000 DATA FFFa04FF0Fq14e14a23q14q,EEF9
 2010 DATA 14j28c58q14q14z28r50f95,ED17
 2020 DATA q14z28z3ChFDeF0n154F041,F6E7
 2030 DATA 8,D201
 2040 DATA "Templates",&FEC,&697
 2050 DATA FFFFFFFF00000000a04c075,713E
 2060 DATA Ca0D5D01000001a156D6169,A1DB
 2070 DATA 6E0D0494E5010040E2B9a14,1FC5
 2080 DATA 3802d1873746172740D00EB,827E
 2090 DATA 011087E2F1030000A6e1869,736B
 2100 DATA 6E666F0Da18EC7484E5b54a,33C4
 2110 DATA 2F006Ca5D46a2458b0De693,DD12
 2120 DATA C0700001200278707020701,AAA
 2130 DATA 03010Cc7F14FEFFFF44a4Cb,3D8A
 2140 DATA 8C3Db7D30d84b8858a90bA8,90C
 2150 DATA 0DaA908aADa04000C8aB9BEa,EB93
 2160 DATA B9F8aC13860000073C556E74,B22F
 2170 DATA 69746C65643E0D7Fb52f20F,F9B6
 2180 DATA 4d2037n20CCaF9BAe203960,1042
 2190 DATA 001B30303000Df40DAe608Aa,6D2F
 2200 DATA F4b60196000C74D696E6573,E262
 2210 DATA 204C6566740D7Fb841CaB03,FBE7
 2220 DATA CaFCC0u80f20BCu8020aF03,9DB5
 2230 DATA 8h2017jC010aFF24a6034a6,EB37
 2240 DATA 0B8dE077jE0b8C0DC4aFDB8,481D
 2250 DATA 02000038a04D4a89aF4b050,C5A1
 2260 DATA 0340700001200058407020?,1539
 2270 DATA 010301bEB0000E4aAD74aCD,E990
 2280 DATA b263Db2B300000e0F005374,5008
 2290 DATA 6172742067616D650D650Fa,7C15
 2300 DATA F92CaFD70aFDCCaE5E0qDDa,60C6
 2310 DATA 3Bg20DCqDDb2074aF5C8h20,DFD7
 2320 DATA kDDbFD3CaF514a90F8aFD1C,A637
 2330 DATA 6000C7jFD60aFDA8aF598e6,FC40
 2340 DATA 01Aa206DbF10D0DcFDDCaF8,45AF
 2350 DATA F4aD04CaD024aA83D9001C7,9573
 2360 DATA cC40DcC47F34aE880aC06Ca,365B
 2370 DATA F0B0aC839a8034300DgC06C,A539
 2380 DATA aFFE8a40A4aFF18q20b807C,10EF
 2390 DATA e80ACq40F0aF8b600Ce01A,56A
 2400 DATA 2000C76465630DfE074eE0A,A0A9
 2410 DATA 2aE86Ce2075700Dg2048e60,17A3
 2420 DATA 64e60n4014eC030i60696E1,249E
 2430 DATA 6014e6044f806F776E0DeE0,B0B0
 2440 DATA 90eC0ACi60j409EaB890020,2734
 2450 DATA 0001A040000643aEDaF0b0,FB2
 2460 DATA 6400700001200138407020?,989B
 2470 DATA 010C0E0Cd19FCFFFF0005d2,1A15
 2480 DATA 43DaF0003000b07b3658a,B980
 2490 DATA 44aFFFF13aF010aF49AaF8C,F31D
 2500 DATA CaF876a5Ca37FF386000073,20D2
 2510 DATA C556E74fF8b2098e20C8q20,ADF0
 2520 DATA 9Ei20C4d4037n20f60F8u20,EA9
 2530 DATA D0aF872e203960001B214D6,2B41
 2540 DATA 96E650DFFFF18aE73Ee20bA,A315
 2550 DATA 8b4019000174E616D653A0,2283
 2560 DATA DcA0FE0EaF29Ce20b80b205,ECE0
 2570 DATA 07527706f73a23a201EaFF6,6E96
 2580 DATA 8e4090e404175746B6F723A,C243
 2590 DATA 0Df4038e6060e6056657273,C315
 2600 DATA 696F6Ec40bE0b60bA0bE039,3504
 2610 DATA 6100176BaECaF3FF1BaFCb9,143D
 2620 DATA 864eE094dE007nE0f20b80a,A970
 2630 DATA 2037n20bA0b60bA0b607De6,212
 2640 DATA 01Ce6030e60bA0r60f205Cu,805E
 2650 DATA 6034eC0b20bC093eC01DaFC,AFE3
 2660 DATA 41626F75742074686973207,B86A
 2670 DATA 0726F6772616D0D4D696E65,3391
 2680 DATA 2D5377656570657220047616,6D9E
 2690 DATA D650D442E57616C74657273,3E54
 2700 DATA 202620412E4661776365747,C9F
 2710 DATA 40D312E3130202832620 4F,7DA5
 2720 DATA 6374203139393229 0D,D076
 2730 DATA *
```

Listing 6 – Mine8

```
   10 REM >Mine8 (Info7)
   20 REM 8bit conversion by DL
   30 REM For all machines
   40 REM (C) BAU January 1993
   50 :
   60 MODE 1
   70 arc=(HIMEM>&10000)
   80 IF (NOT arc) AND PAGE>&1100 THEN P
AGE=&1100:CHAIN "Mine8"
   90 VDU 23;10,32;0;0;0;0;
  100 PROCinit
  110 IF NOT arc THEN HIMEM=screen
  120 REPEAT
  130 done=FALSE
  140 PROCblank(FALSE)
  150 PROCadjust
  160 PROCplay
  170 UNTIL FNquit
  180 MODE 1
  190 *FX229,0
  200 *FX4,0
  210 END
  220 :
  230 DEF PROCinit
  240 lines=23
  250 screen=&8000-lines*&280
  260 VDU 23;12,(screen DIV 8) DIV 256;0
;0;0;0;
  270 VDU 23;13,(screen DIV 8) MOD 256;0
;0;0;0;
  280 VDU 23;6,lines;0;0;0;0;
  290 top=32-lines
  300 VDU 19,1,4;0;19,2,6;0;
  310 DIM t$(13),d(8,1),vx(2),v$(2),v(2)
,max(2),w 22*12
  320 FOR i=0 TO 2
  330 READ vx(i),v$(i),v(i),max(i)
  340 NEXT
  350 ad=0:trod=&30:mine=9:wrong=10:dead
=1:quit=2:end=3:clear=0:flag=1:maybe=2
  360 flat$=FNread
  370 tile$=FNread
  380 FOR i=1 TO 8
  390 t$(i)=FNb(48+i,0,3)
  400 NEXT
  410 t$(mine)=FNb(130,3,0)
  420 t$(wrong)=FNplot(0,0,-4)+FNgcol(0)
+CHR$5+CHR$130+FNplot(0,-32,0)+FNgcol(2)
+CHR$131+CHR$4
  430 flag$=FNplot(0,-44,-12)+FNb(128,1,
3)
  440 maybe$=FNplot(0,-44,-16)+FNgcol(0)
+CHR$5+CHR$129+CHR$4
  450 VDU 23,128,&0E,&0C,&08,&08,&08,&1C
,&3E,&00
  460 VDU 23,129,&1C,&22,&02,&04,&08,&00
,&08,&00
  470 VDU 23,130,&06,&08,&1C,&3E,&3E,&3E
,&1C,&00
  480 VDU 23,131,&41,&22,&14,&08,&14,&22
,&41,&00
  490 *FX4,1
  500 *FX229,1
  510 ENDPROC
  520 :
  530 DEF FNread
  540 a$=""
  550 REPEAT
  560 READ n$:z=VALMID$(n$,2):n$=LEFT$(n
$,1)
  570 IF n$="c" a$=a$+FNgcol(z)
  580 IF n$="p" READ x,y:a$=a$+FNplot(z,
x,y)
  590 UNTIL n$="x"
  600 =a$
  610 :
  620 DEF FNgcol(c)=CHR$18+CHR$0+CHR$c
  630 DEF FNplot(p,x,y)=CHR$25+CHR$p+FNw
(x)+FNw(y)
  640 DEF FNw(a)=CHR$((a+65536) MOD 256)
+CHR$((a+65536) DIV 256)
  650 DEF FNb(c,o,i)=CHR$5+FNgcol(o)+CHR
$c+FNplot(0,-32,-8)+CHR$c+FNplot(0,-36,4
)+CHR$c+FNplot(0,-24,0)+CHR$c+FNgcol(i)+
FNplot(0,-36,0)+CHR$c+CHR$4
  660 DEF FNpeek(px,py)=?(w+px+py*22)
  670 DEF PROCpoke(px,py,pp,f)
  680 ?(w+px+py*22)=pp
  690 IF f PROCtile(px,py,pp)
  700 ENDPROC
  710 :
  720 DEF PROCtile(x,y,t)
  730 MOVE (x-1)*64+4,(y-1)*64+4
  740 s=t DIV 16
  750 IF s=flag PRINTtile$;flag$;:ENDPRO
C
  760 IF s=maybe PRINTtile$;maybe$;:ENDP
ROC
  770 IF s=clear PRINTtile$;:ENDPROC
  780 PRINTflat$;t$(t MOD 16);
  790 ENDPROC
  800 :
  810 DEF PROCblank(new)
  820 xsize=v(0):ysize=v(1):mines=v(2)
  830 IF new VDU 28,0,31,39,top+2,12,26
  840 blx=(20-xsize)*32:bly=32+(10-ysize
)*32
  850 VDU 29,blx;bly;
  860 FOR i=0 TO 22*12-1:w?i=trod:NEXT
  870 FOR y=1 TO ysize
  880 FOR x=1 TO xsize
  890 PROCpoke(x,y,0,TRUE)
  900 NEXT
  910 NEXT
  920 FOR m=1 TO mines
  930 REPEAT
  940 x=RND(xsize):y=RND(ysize)
  950 UNTIL FNpeek(x,y)<>mine
  960 PROCpoke(x,y,mine,FALSE)
  970 FOR x1=x-1 TO x+1
  980 FOR y1=y-1 TO y+1
  990 z=FNpeek(x1,y1)
 1000 IF z<>mine PROCpoke(x1,y1,z+1,FALS
E)
 1010 NEXT
 1020 NEXT
 1030 NEXT
 1040 ENDPROC
 1050 :
 1060 DEF PROCadjust
 1070 PRINTFNv(0,0);FNv(1,0);FNv(2,0);
 1080 oad=-1
 1090 REPEAT
 1100 REPEAT
 1110 IF ad<>oad PRINTFNv(ad,1);
```

```
1120 oad=ad
1130 k=GET
1140 IF k=136 PRINTFNv(ad,0);:ad=(ad+2)
MOD3
1150 IF k=137 PRINTFNv(ad,0);:ad=(ad+1)
MOD3
1160 IF k=138 AND v(ad)>1 v(ad)=v(ad)-1
:oad=-1
1170 IF k=139 AND v(ad)<max(ad) v(ad)=v
(ad)+1:oad=-1
1180 UNTIL k=13 OR k=27 OR k=32
1190 UNTIL v(2)<v(0)*v(1) OR k=27
1200 COLOUR 128
1210 PRINTTAB(0,top);SPC40;
1220 IF k=27 done=quit:ENDPROC
1230 IF xsize<>v(0) OR ysize<>v(1) OR m
ines<>v(2) PROCblank(mines=v(2))
1240 ENDPROC
1250 :
1260 DEF FNv(v,b)
1270 COLOUR 128+b
1280 COLOUR 2:PRINTTAB(vx(v)-1,top);" "
;v$(v);" ";
1290 COLOUR 3:PRINTRIGHT$("  "+STR$v(v)
,2);" ";
1300 ="
1310 :
1320 DEF PROCplay
1330 xp=xsize DIV 2:IF xp=0 xp=1
1340 yp=ysize DIV 2:IF yp=0 yp=1
1350 IF done=quit ENDPROC
1360 marks=mines:left=mines:start=TIME:
tick=start
1370 COLOUR 2:PRINTTAB(1,top);"Time";TA
B(29,top);"Mines: ";FNmarks;FNtime;
1380 REPEAT
1390 REPEAT
1400 IF TIME>tick+100 tick=TIME:PRINTFN
time;
1410 PROCcurs(xp,yp,2)
1420 k=INKEY20
1430 PROCcurs(xp,yp,0)
1440 IF k=-1 k=INKEY20
1450 UNTIL k<>-1
1460 IF k=136 xp=xp-1:IF xp=0 xp=xsize
1470 IF k=137 xp=xp+1:IF xp>xsize xp=1
1480 IF k=138 yp=yp-1:IF yp=0 yp=ysize
1490 IF k=139 yp=yp+1:IF yp>ysize yp=1
1500 IF k=32 now=TIME:PROCtread(xp,yp):
TIME=now
1510 IF k=13 PROCmark(xp,yp,-1)
1520 IF k=ASC"F" OR k=ASC"f" PROCmark(x
p,yp,flag)
1530 IF k=ASC"M" OR k=ASC"m" PROCmark(x
p,yp,maybe)
1540 IF k=ASC"C" OR k=ASC"c" PROCmark(x
p,yp,clear)
1550 IF k=27 done=quit
1560 UNTIL done
1570 IF done=dead OR done=end PROCrevea
l
1580 ENDPROC
1590 :
1600 DEF FNtime
1610 time=(TIME-start) DIV 100
1620 s=time MOD 60
1630 m=(time DIV 60) MOD 60
1640 COLOUR 3:PRINTTAB(6,top);RIGHT$("0
0"+STR$m,2);":";RIGHT$("00"+STR$s,2);
1650 ="
1660 :
1670 DEF FNmarks
1680 COLOUR 3:PRINTTAB(37,top);RIGHT$("
  "+STR$marks,2);
1690 ="
1700 :
1710 DEF PROCcurs(x,y,c)
1720 GCOL 0,c:MOVE (x-1)*64,(y-1)*64:PL
OT 1,64,0:PLOT 1,0,64:PLOT 1,-64,0:PLOT
1,0,-64
1730 ENDPROC
1740 :
1750 DEF PROCtread(x,y)
1760 t=FNpeek(x,y) MOD 16
1770 IF FNpeek(x,y)>15 ENDPROC
1780 IF t=mine done=dead
1790 PROCpoke(x,y,t OR trod,TRUE)
1800 IF t>0 ENDPROC
1810 PROCtread(x-1,y-1):PROCtread(x-1,y
)
1820 PROCtread(x-1,y+1):PROCtread(x,y+1
)
1830 PROCtread(x+1,y+1):PROCtread(x+1,y
)
1840 PROCtread(x+1,y-1):PROCtread(x,y-1
)
1850 ENDPROC
1860 :
1870 DEF PROCmark(x,y,m)
1880 t=FNpeek(x,y) MOD 16:s=FNpeek(x,y)
DIV 16
1890 IF s=3 ENDPROC
1900 IF m=-1 m=(s+1) MOD 3 ELSE IF m=s
m=clear
1910 marks=marks-(s=flag)+(m=flag)
1920 left=left-(s=flag AND t=mine)+(m=f
lag AND t=mine)
1930 PRINTFNmarks;
1940 PROCpoke(x,y,t+16*m,TRUE)
1950 IF marks=0 done=end
1960 ENDPROC
1970 :
1980 DEF PROCreveal
1990 IF left=0 PRINTFNm("All mines foun
d!"):ENDPROC
2000 IF done=dead PRINTFNm("B O O M !")
2010 FOR y=1 TO ysize
2020 FOR x=1 TO xsize
2030 a=FNpeek(x,y) MOD 16:b=FNpeek(x,y)
DIV 16
2040 IF b=0 AND a=mine PROCtile(x,y,min
e OR trod)
2050 IF b=flag AND a<>mine PROCtile(x,y
,wrong OR trod)
2060 NEXT
2070 NEXT
2080 IF done=dead PROCcurs(xp,yp,2):END
PROC
2090 m$=STR$left+" wrong flag":IF left>
1 m$=m$+"s"
2100 PRINTFNm(m$+"!")
2110 ENDPROC
2120 :
2130 DEF FNm(m$)
2140 VDU 29,(1280-32*LENm$)/2;32*lines-
8;5,18,0,1
2150 MOVE 0,4:PRINTm$;:MOVE 0,-4:PRINTm
$;
2160 MOVE 4,0:PRINTm$;:MOVE -4,0:PRINTm
$;
2170 MOVE 0,0:GCOL 0,2:PRINTm$;
2180 VDU 29,blx;bly;
2190 =CHR$4
2200 :
2210 DEF FNquit
2220 PRINTTAB(6,31);"Press any key to p
lay again";
2230 k=GET
2240 PRINTTAB(0,31);SPC39;
2250 VDU 28,0,top+1,39,top,12,26,29,blx
;bly;
2260 PROCcurs(xp,yp,0)
2270 =(k=27)
2280 :
2290 DATA  3,Width,12,20
2300 DATA 15,Height,8,10
2310 DATA 28,Mines,15,99
2320 :
2330 DATA c1,p0,0,56,p81,56,-56,p81,0,5
6,p0,-44,-12,x
2340 DATA c3,p1,0,56,p1,56,0
2350 DATA c1,p33,0,-56,p1,-56,0,p1,4,4,
p1,48,0,p1,0,48
2360 DATA c3,p1,-48,0,p1,0,-44
2370 DATA c2,p0,4,0,p0,0,40,p81,40,-40,
p81,0,40,p0,8,8,x
```

Listing 7 – GetMC32

```
  10 REM >GETmc32 (Info8)
  20 REM Machine code GET
  30 REM for 32-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 link=14:pc=15
  80 FOR pass%=0 TO 2 STEP 2
  90 P%=code%
 100 [OPT pass%
 110 .getmc
 120 SWI     "OS_ReadC"
 130 SWI     "OS_WriteC"
 140 CMP     r0,#27
 150 BNE     getmc
 160 MOV     pc,link
 170 ]
 180 NEXT pass%
 190 PRINT"Press some keys..."
 200 CALL getmc
 210 END
```

Listing 8 – GetMC8

```
  10 REM >GETmc8 (Info9)
  20 REM Machine code GET
  30 REM for 8-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 osrdch=&FFE0
  80 oswrch=&FFEE
  90 FOR pass%=0 TO 2 STEP 2
 100 P%=code%
 110 [OPT pass%
 120 .getmc
 130 JSR osrdch
 140 JSR oswrch
 150 CMP #27
 160 BNE getmc
 170 RTS
 180 ]
 190 NEXT pass%
 200 PRINT"Press some keys..."
 210 CALL getmc
 220 END
```

Listing 9 – INPUTmc8

```
  10 REM >INPUTmc8 (Info10)
  20 REM Machine code INPUT
  30 REM for 8-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 osword=&FFF1
  80 FOR pass%=0 TO 2 STEP 2
  90 P%=code%
 100 [OPT pass%
 110 .inputmc
 120 LDA #0
 130 LDX #block MOD 256
 140 LDY #block DIV 256
 150 JSR osword
 160 RTS
 170 :
 180 .block
 190 EQUW buffer
 200 EQUB 16
 210 EQUB 32
 220 EQUB 255
 230 :
 240 .buffer
 250 EQUS STRING$(16," ")
 260 ]
 270 NEXT pass%
 280 PRINT"Enter a string (up to 16 cha
rs)"'
 290 CALL inputmc
 300 PRINT"The string you typed was '"$
buffer"'"
 310 END
```

Listing 10 – INPUTmc32

```
  10 REM >INPUTmc32 (Info11)
  20 REM Machine code INPUT
  30 REM for 32-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 link=14:pc=15
  80 FOR pass%=0 TO 2 STEP 2
  90 P%=code%
 100 [OPT pass%
 110 .inputmc
 120 ADR     r0,buffer
 130 MOV     r1,#16
 140 MOV     r2,#32
 150 MOV     r3,#255
 160 SWI     "OS_ReadLine"
 170 MOV     pc,link
 180
 190 .buffer EQUS STRING$(16," ")
 200 ]
 210 NEXT pass%
 220 PRINT"Enter a string (up to 16 cha
rs)"'
 230 CALL inputmc
 240 PRINT"The string you typed was '"$
buffer"'"
 250 END
```

Listing 11 – INKEYmc8

```
  10 REM >INKEYmc8 (Info12)
  20 REM Machine code INKEY
  30 REM for 8-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 oswrch=&FFEE
  80 osbyte=&FFF4
  90 delay=50
 100 FOR pass%=0 TO 2 STEP 2
 110 P%=code%
 120 [OPT pass%
 130 .inkeymc
 140 LDX #delay MOD 256
 150 LDY #delay DIV 256
 160 LDA #129
 170 JSR osbyte
 180 CPY #&FF
 190 BNE key_pressed
 200 LDX #ASC"."
 210 .key_pressed
 220 TXA
 230 JSR oswrch
 240 CMP #27
 250 BNE inkeymc
 260 RTS
 270 ]
 280 NEXT pass%
 290 PRINT"Press some keys..."
 300 CALL inkeymc
 310 END
```

Listing 12 – INKEYmc32

```
  10 REM >INKEYmc32 (Info13)
  20 REM Machine code INKEY
  30 REM for 32-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 link=14:pc=15
  80 delay=50
  90 FOR pass%=0 TO 2 STEP 2
 100 P%=code%
 110 [OPT pass%
 120 .inkeymc
 130 MOV     r1,#delay MOD 256
 140 MOV     r2,#delay DIV 256
 150 MOV     r0,#129
 160 SWI     "OS_Byte"
 170 CMP     r2,#&FF
 180 MOVEQ   r1,#ASC"."
 190 MOV     r0,r1
 200 SWI     "OS_WriteC"
 210 CMP     r0,#27
 220 BNE     inkeymc
 230 MOV     pc,link
 240 ]
 250 NEXT pass%
 260 PRINT"Press some keys..."
 270 CALL inkeymc
 280 END
```

Listing 13 – INKEY-

```
  10 REM >Inkey- (Info14)
  20 REM Find INKEY- values
  30 REM for all machines
  40 REM (c) BAU January 1993
  50 :
  60 PRINT"Press keys now:"
  70 REPEAT
  80 FOR i%=1 TO 127
  90 IF INKEY-i% PRINT -i%,256-i%
 100 NEXT
 110 UNTIL FALSE
 120 END
```

Listing 14 – Bat32

```
  10 REM >Bat32 (Info15)
  20 REM Machine code INKEY-
  30 REM for 32-bit machines
  40 REM (c) BAU January 1993
  50 :
  60 DIM code% &100
  70 sp=13:link=14:pc=15
  80 left=256-98
  90 right=256-67
 100 FOR pass%=0 TO 2 STEP 2
 110 P%=code%
 120 [OPT pass%
 130 .batmove
 140 STMFD   (sp)!,{r0-r12,link}
 150 MOV     r4,#19
 160 .batloop
 170 MOV     r0,#19
 180 SWI     "OS_Byte"
 190 MOV     r1,#left
 200 BL      inkey
 210 BNE     notleft
 220 CMP     r4,#0
 230 SUBGT   r4,r4,#1
 240 .notleft
 250 MOV     r1,#right
 260 BL      inkey
 270 BNE     notright
 280 CMP     r4,#37
 290 ADDLT   r4,r4,#1
 300 .notright
 310 SWI     &100+31
 320 MOV     r0,r4
 330 SWI     "OS_WriteC"
 340 SWI     &100+20
 350 SWI     &100+ASC" "
 360 SWI     &100+ASC"="
 370 SWI     &100+ASC" "
 380 SWI     "OS_ReadEscapeState"
 390 BCC     batloop
 400 LDMFD   (sp)!,{r0-r12,pc}
 410
 420 .inkey
 430 STMFD   (sp)!,{link}
 440 MOV     r2,#255
 450 MOV     r0,#129
 460 SWI     "OS_Byte"
 470 CMP     r1,#255
 480 LDMFD   (sp)!,{pc}
 490 ]
 500 NEXT pass%
 510 MODE 7
 520 VDU 23,1,0|
```