Computing with the AMSTRAD

INC. AMTIX!

$4.50

The independent AUSTRALIAN magazine for Amstrad users

SEPTEMBER, 1987

Registered by Australia Post – Publication No. VBP7811

A Database Publication

3D graphics on the CPC

+ start of a new series on graphics for the PCW

• DIGGER – THE GAME OF THE MONTH •
• SOUND – THE NUMBERS BEHIND THE NOTES •
• UK NEWS • GRAPHICS • CP/M FEATURE •
• BUSINESS SECTION • JOYSTICK REVIEW •

$ O $ % $AVINGS ON $OFTWARE %

Contents

September 1987

Published monthly by Planet Publishing Pty Ltd under licence from Database Publications Ltd.
Mail: P.O. Box 11, Blackmans Bay, Tasmania 7152.
Telex: AA58134, Attn. HT163 General enquiries, phone (002) 29 4377.

Annual subscription (12 issues)

Australia	$ 45
South Pacific (inc NZ)	$A65

Subscriptions – see centre page

Ordering the elements of arrays

By Pete Bibby

Right, no hanging about — type Program I into your micro. If you remember what we covered last time you'll find it easy to understand what's happening.

The DIM of line 20 sets aside memory space for an array, *hiphip()*. This array is made up of 11 elements going from *hiphip(0)* all the way to *hiphip(10)*.

As is my normal practice, the first element of the array, *hiphip(0)*, is ignored. This is wasteful of memory but it's justified by making the program's action easier to understand.

The FOR ... NEXT loop of lines 40 to 60 READs into the array the numbers that the Amstrad finds in the data line. As *loop* goes from 1 to 10, so each element of the array is used to store one of the numbers from line 110.

The second loop prints out the list of numbers in reverse order. Try changing the first line of the loop to:

```
80 FOR control=10 TO 1 STEP -2
```

or:

```
80 FOR control=10 TO 1 STEP -3
```

and see what happens.

By now you should have grasped that combining FOR ... NEXT loops and arrays is a very powerful technique. Using a loop you can read data into a program and store it in an array.

In the programs in this article all the data comes from the data lines at the end of the program. This is just to make the examples clearer.

In practice, you're more likely to use INPUT or take the data from a tape or disk file. Whichever method you use, the fact remains that storing the data in an array allows you to do all sorts of things that you couldn't do with our normal variables.

```
10 REM Program I
20 DIM hiphip(10)
30 REM READs it in
40 FOR loop=1 TO 10
50 READ hiphip(loop)
60 NEXT loop
70 REM PRINTs it out in reverse
80 FOR control=10 TO 1 STEP -1
90 PRINT hiphip(control)
100 NEXT control
110 DATA 0,1,2,3,4,5,6,7,8,9
```

Program I

Try getting Program I to display all the even numbers in the list or all the odd ones or, maybe, all the even numbers greater than 3. Once you've got the numbers in an array, manipulating the data (as it's known) becomes much easier.

Up until now, all the arrays we've DIMmed have been numeric, used (amazingly) for holding numbers. You can also have arrays that hold strings and these are known as string arrays. Program II shows a string array being used to store five names.

```
10 REM Program II
20 DIM name$(5)
30 REM Filling array
40 FOR loop=1 TO 5
50 READ name$(loop)
60 NEXT loop
70 REM Using array for output
80 FOR control=1 TO 5
90 PRINT name$(control)
100 NEXT control
110 PRINT
120 DATA Pete, Eileen, Bodger, Spot,
Eric
```

Program II

This works in exactly the same way as Program 1 except that now strings are being read from the data line and placed in a string array, *name$()*.

It's the dollar sign $ at the end of the array name that indicates to the micro that it's a string array. Be careful not to miss this off as in:

```
50 READ name (loop)
```

If you do this the program halts with the annoyingly misleading message:

```
Syntax error in 120
```

element	name$(1)	name$(2)	name$(3)	name$(4)	name$(5)
contents	Pete	Eileen	Bodger	Spot	Eric

Figure 1: The array name$() and its contents

Notice that the error has occurred because you've tried to READ a string into *name()* — a numeric array. The Amstrad can't reconcile the two, so it stops and blames the data line.

While you can't put strings into a numeric array, you can read numbers into a string array, as changing the data line to:

```
120 DATA 1,2,3,4,5
```

shows. The program happily accepts the numbers, storing them in *name$(1)* through to *name$(5)*. However the numbers are stored as strings. You can't do maths with them, as you'll find if you enter:

```
PRINT 2*name$(1)
```

which gives the:

```
Type mismatch
```

message.

Let's take a closer look at the five elements of the array *name$()*. Figure I shows the elements of the array and the contents of each element.

Notice how not only are the names stored in the array, with *name$(1)* holding Pete and *name$(3)* containing Bodger, but also they are stored in a particular order.

Just as the first element is *name$(1)*, the second is *name$(2)* and so on, the first item of data is Pete, the second Eileen and so on to Eric.

So by putting things into an array we are also giving them an order. This means that when we create an array, not only are we using the elements as pigeonholes to keep things in, we're also saying something about the way these things are related. In posh terms we are structuring the data.

```
10 REM Program III
20 DIM name$(5),mark(5)
30 REM Filling array
40 FOR loop=1 TO 5
50 READ name$(loop),mark(loop)
60 NEXT loop
70 REM Using array for output
80 FOR control=1 TO 5
90 PRINT name$(control),mark(control)
100 NEXT control
110 PRINT
120 DATA Pete,10,Eileen,20,Bodger,30,
Spot,30,Eric,5
```

Program III

In the last program it didn't really matter who came first in the list or who came last, it was just a collection of the inhabitants of my house. However at times the order of an array is important, as you'll see when you run Program III.

As you can see from line 20, this program uses two arrays *name$* and *mark()*. Line 50, embedded in a FOR ... NEXT loop, uses them to store a set of names and marks. Line 80 then uses another FOR ... NEXT loop to display the list of names and marks.

It's a fairly straight forward program that should give you no problems. It does, though, make some interesting points. To see what I mean, take a look at the data line, line 120. From just looking at this you should be able to see that Pete's mark is 10, Eileen's is 20 and so on. The name and the following mark are related.

This relationship still holds in line 50, where *name$(loop)* and *mark(loop)* put the data into the relevant array. When *loop* is 1, Pete is stored in *name$(1)* and Pete's mark, 10, is stored in *mark(1)*. Similarly Eileen is put into *name$(2)* and her mark of 20 is held in *mark(2)*.

Each set of related data, someone's name and their mark, has the same value of *loop*, so each element of *name$()* is linked to the element in *mark()* with the same subscript (the number in the brackets).

Suddenly the order of both arrays becomes important. Figure II shows the elements of *name$()* and *mark()* along with their contents.

Although the two elements *name$()* and *mark()* are completely separate, they do have a relationship.

The first element in each array holds information about the same person. Similarly with the second and so on to the end.

It wouldn't be hard to have a third array such as *age()* which contains the age of each of the members of *name$()*. In this *age(1)* would be my age, *age(4)* Spot's, and *age(5)* would hold Eric's age.

When we have a set of arrays

element	1	2	3	4	5
name$() contents	name$(1) Pete	name$(2) Eileen	name$(3) Bodger	name$(4) Spot	name$(5) Eric
mark() contents	mark(1) 10	mark(2) 20	mark(3) 30	mark(4) 30	mark(5) 5

Figure II: The parallel arrays name$() and mark$()

like this, where the elements of each array are in the same order and each element concerns the same subject (as *name$(3)*, *age(3)* and *mark(3)* all concern Bodger), the arrays are called parallel arrays. And when you've got a set of parallel arrays, it's easy to get information out of them by using the subscript as a pointer. Program IV shows how this is done.

The first part of the program is the same as Program III. It takes the values from the data line and puts them in the arrays *Name$()* and *mark()*. The second part of the program is rather different. It asks you whose mark you'd like to know, asking you to enter the number shown by the person's name (lines 100-120).

The choice of numbers is limited to the number of elements in the array. And, of course, the numbers reflect the position of the various elements in *name$*.

Subject	First	Second	Third
English	Ian	Sue	Tom
Maths	Sue	Ian	Tom
Computing	Ian	Tom	Sue

Figure III: The top three students in each subject

As soon as the Amstrad has the number, say *x*, it can go to the array *mark()* and pick out *mark(x)* knowing that this is the mark obtained by whoever is held in *name$(x)*. The number *x* acts as a pointer to the information you want to retrieve from the array.

Because of the WHILE . . . WEND loop, the process continues until yo give it the number 0. Can you explain why the program prints out a 0 before it stops? What about *mark(O)*, the element we've ignored? Might this have something to do with it?

```
10 REM Program IV
20 DIM name$(5),mark(5)
30 REM Filling array
40 FOR loop=1 TO 5
50 READ name$(loop),mark(loop)
60 NEXT loop
70 REM selecting one item from array
80 pointer=1
90 WHILE pointer>0
100 PRINT "Whose mark do you want?","
Press the relevant number key:"
110 PRINT
120 PRINT "1 Pete":PRINT"2 Eileen":PR
INT"3 Bodger":PRINT"4 Spot":PRINT"5 E
ric"
130 INPUT pointer
140 IF pointer >5 THEN GOTO 100
150 PRINT name$(pointer),mark(pointer
)
160 PRINT
170 WEND
180 DATA Pete,10,Eileen,20,Bodger,30,
Spot,40,Eric,5
```

Program IV

When you've understood how the program works, try adding a few more arrays like *age()* and getting the micro to print out this information as well as the mark. And after that you can strain your brain with Program V.

This has three string arrays, each with three elements (ignoring the first). These elements are used to hold the names of the first three students in English, Maths and Computing. As you'll see from Figure III, it's the same three each time, only the order has changed.

Although heavily disguised with a pair of nested FOR . . . NEXT loops, the first part of the program is really the same as in all the programs so far. It just takes data from the data line and stores it in an array. However in this case it's storing it in three arrays, the outer loop deciding which of the arrays is used by the inner.

```
10 REM Program V
20 DIM english$(3),math$(3),computing
(3)
30 FOR outer=1 TO 3
40 FOR inner=1 TO 3
50 IF outer=1 THEN READ english$(inne
r)
60 IF outer=2 THEN READ math$(inner)
70 IF outer=3 THEN READ computing$(in
ner)
80 NEXT inner
90 NEXT outer
100 PRINT "Press 1, 2 or 3"
110 INPUT choice
120 CLS
130 IF choice=1 THEN PRINT "English r
esults:"
140 IF choice=2 THEN PRINT "Maths res
ults:"
150 IF choice=3 THEN PRINT "Computing
 results:"
160 PRINT
170 FOR loop=1 TO 3
180 IF choice=1 THEN PRINT english$(l
oop)
190 IF choice=2 THEN PRINT math$(loop
)
200 IF choice=3 THEN PRINT computing$
(loop)
210 NEXT loop
220 PRINT
230 DATA Ian,Sue,Tom
240 DATA Sue,Ian,Tom
250 DATA Ian,Tom,Sue
```

Program V

The first time round the outer loop, *outer* is 1. The program then goes on to the inner loop which cycles three times, each time taking a name from the dat list and putting it in an element of a string array. Since *outer* is 1, the array used is *english$* (line 50).

When the inner loop has finished the outer loop starts again. This time *outer* is 2 and so the array chosen is *math$* (line 60). I leave it to you to figure out what happens on the third cycle of the outer loop. Figure IV shows the resulting arrays.

By the time the program gets to line 100, the three arrays have had

Array/Element	1	2	3
English$	Ian	Sue	Tom
Math$	Sue	Ian	Tom
Computing$	Ian	Tom	Sue

Figure IV: The arrays and their elements

their elements filled. You are now asked to press 1,2 or 3 and, depending on your response, the top three in either English, Maths or Computing are displayed.

Can you alter the program so that when you pick a number it gives you the names of all the people who came first, or second or third? Think of parallel arrays and pointers.

No doubt some of you will be looking a bit askance at Program V and quite rightly. After all the message "Press 1, 2 or 3" isn't all that informative. It should really tell you what happens if you do. And the input isn't mugtrapped. Try entering -2 or 5 and see what results. It would be much better to only allow inputs of 1, 2 or 3. I leave this to you.

Another criticism of Program V is that it's too long. Do we really need those two FOR . . . NEXT loops? The answer is no, as you'll see if you add the following lines to Program V.

Here the use of one READ for all the arrays has saved us a loop. However notice that the data in the data list has had to be put into a different order. The data list in Program V mirrored the data in Figure III.

```
10 REM Program Va
40 REM No inner loop
50 REM This reads all values into arr
ays
60 READ english$(outer),math$(outer),
computing$(outer)
70 REM One line saves five
80 REM But the data has to be re-orga
nised
230 DATA Ian,Sue,Ian
240 DATA Sue,Ian,Tom
250 DATA Tom,Tom,Sue
```

Program Va

In Program Va the data is arranged so that all the firsts are together, then the seconds and then the thirds. The program is shorter, but only at the expense of manipulating the information before it's presented to the program. While this is easy in this case, imagine doing it for a class of 40.

The point to grasp is that while we may make the program simpler and more efficient, this has to be balanced against the amount of work that has to be done to the data to get it into the form that this simpler, more efficient program requires.

Now, let's take a look at what we've done so far this month. We've seen how array elements can be used to hold numbers and strings. We've also seen how we can use the arrays to hold an ordered list. We can get at any element of this list by using the subscript as a pointer. And we're not only stuck to one array.

So long as we follow the same order, we can have a number of arrays and refer to the elements concerning the same subject by using just one pointer. And these parallel arrays, as they are known are powerful things.

But they're not as powerful as the two dimensional arrays we'll be dealing with next time. Until then I leave you to ponder program VI which does the same job as Program V but in a different way.

What's happening here? How come there are two numbers inside the DIM? And why is there only one array and not three? All will be revealed next month.

```
10 REM Program VI
20 DIM result$(3,3)
30 FOR subject=1 TO 3
40 FOR place=1 TO 3
50 READ result$(subject,place)
60 NEXT place
70 NEXT subject
80 PRINT "Press 1, 2 or 3"
90 INPUT choice
100 CLS
110 IF choice=1 THEN PRINT "English r
esults:"
120 IF choice=2 THEN PRINT "Maths res
ults:"
130 IF choice=3 THEN PRINT "Computing
 results:"
140 PRINT
150 FOR place=1 TO 3
160 PRINT result$(choice,place)
170 NEXT place
180 PRINT
190 DATA Ian,Sue,Tom
200 DATA Sue,Ian,Tom
210 DATA Ian,Tom,Sue
```

Program VI

New Faces on Display at Show

More than a quarter of the hardware and software suppliers who exhibited at the recent giant Amstrad Computer Show were attended for the first time.

The event has always had a waiting list of firms keen to show their wares at the UK's leading machine-specific show.

Eighteen of them had their patience rewarded when they took their place alongside familiar names from previous shows at the Alexandra Palace, London.

More suppliers than ever attended because it was held in the spacious Alexandra Pavilion where there is 50 per cent more room than at the previous location, the Novotel in Hammersmith.

The popular Amstrad Theatre also benefitted from the move with double seating capacity for the near non-stop presentations that took place during show hours.

A growing number of CPC suppliers launched new products at the show, such as CPC's spreadsheet First Calc by Minerva.

Other products on sale included new word processor Pocket Protext for the CPC6128; two CPC language compilers Maxam II and C; KDS Electronics new disco type Sound to Light Converter for the CPC; and Siren Software's new stereo amplifier system for the CPC called Sound Blaster.

6128 is making a serious impact

Promoted originally as a games machine, the CPC 6128 is carving out a solid future for itself at the serious end of the market, a *Computing with the Amstrad* survey of software houses has revealed.

Arnor was quick to sing the machine's praises in this context.

Managing director Dave Fisk said "We see the 6128 as a damned good machine. It is very flexible; we have used it quite extensively ourselves for word processing, accounting and business software of all kinds. It offers great value for money.

"We have invested a lot of time and effort on the machine and hope to continue to do so even as Amstrad seems to be moving up market".

Fisk hinted that Arnor had some good contracts with High Street retailers for CPC software, reinforcing confidence in the future of the machine.

Robin Thompson of Tasman Software echoed that assurance.

"We are confident enough at the moment to invest effort in preparing new utilities for the machine.

"Our present packages are selling very well, especially our word processing program Tasman 6128", he said.

David Link of HiSoft agreed. "As a software house producing languages, we see the 6128 as being an important machine because of the disc drive and extra memory".

He said his company would probably concentrate on writing new software only for the 6128.

Peter Brunning of Brunning Software said that the word processing program Brunword, with its associated spellchecker and datafile was doing very well.

"It has become clear through our own surveys that there is a huge market for good, serious 6128 software. We definitely believe there is a future in it".

Meanwhile, Cornix has re-released its Simple Accounts package on the CPC, with "significant" upgrading, based on comments from users of earlier versions.

Games Update

For adults only was the reaction from Atlantis Software executives to their latest comedy/fantasy adventure for the CPC series.

Called DAA — for dungeons, amethysts, alchemists — it contains naughty bits involving novice nuns, entangled lovers and busty barmaids.

"The decision to release an adults only game was not taken lightly", said Mike Cole, managing director of Atlantis.

"We've gained a good reputation for providing good games at budget prices and the last thing we want is to be accused of corrupting our young customers — DAA is meant as 'cheeky fun' and we sincerley hope that most people will take it in that spirit".

☆ ☆ ☆

Ariolasoft's newly licensed label Starlight Software has made its debut with three titles for the CPC range.

They are Greyfell, Dogfight 2187 and Deathscape, on cassette.

☆ ☆ ☆

Age-old superhero Flash Gordon has at last made it to the CPC. According to Mastertronic the story goes that the evil Ming has targeted Earth with planet-killer missiles. To stop him destroying life the player assumes the role of Flash Gordon to foil the plot.

Ming must die if Earth is to live. He carries the missile controls with him and only through destroying these can there be any hope of success.

☆ ☆ ☆

Five of the world's bestselling murder mystery writers are trying to bump each other off in the latest CPC release from US Gold.

The Murder Club has convened in the creepy Gargoyle Hotel on a dark and stormy night and the player has the task of helping ace detective Hercules Holmes sort out the mayhem by midnight.

Game that's too realistic

An award-winning computer game that is a best-seller in the UK on the CPC range has been banned from open sale by West Germany for being too realistic.

Silent Service from Micro-Prose, an accurate simulation of a World War II submarine in action, can now only be bought in the federal republic from regulated outlets such as sex shops.

The decision to outlaw the title, known as Das U Boot in Germany, has been made under the Youth Dangerous Publications List.

This controversial law was passed to protect youngsters from products ranging from pornography to material thought likely to incite aggressive behaviour.

This is in fact the second time that publisher Micro-Prose has fallen foul of the list.

A version of the giant software house's first title to be banned in Germany, F-15 Strike Eagle, has just been released for the on cassette and disc.

The package is a combat flight simulator that transports the player into a world of electronic warfare.

Microprose president Bill Stealley

Company president Bill Stealey — a fighter pilot who is an adviser to the US Joint Chiefs of Staff — flew to Munich recently to discuss the situation with lawyers.

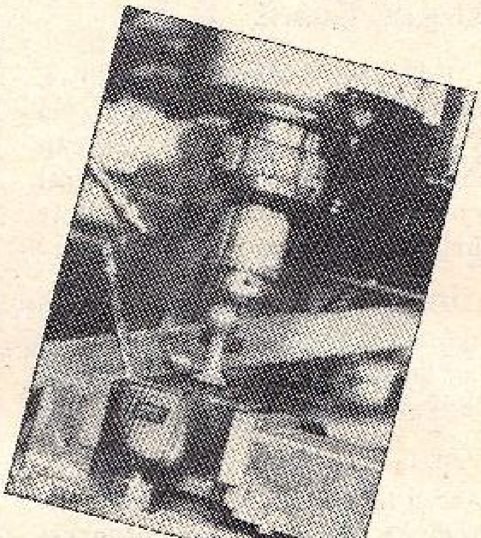
"We are going to take a stand", he told *Computing with the Amstrad* .

"This legislation, wrongly in our opinion, lumps together computer software with pornographic videos.

"To drive sophisticated software into the back streets is not only harmful to companies like ours, but will have a damaging effect on Germany's own software industry".

GIVING IT SOME STICK

An Amstrad CPC 6128 has been employed in the waggle testing of a Konix joystick. The company is at present carrying out a test to see how long a Speed King can last while being operated at 450 waggles a minute.



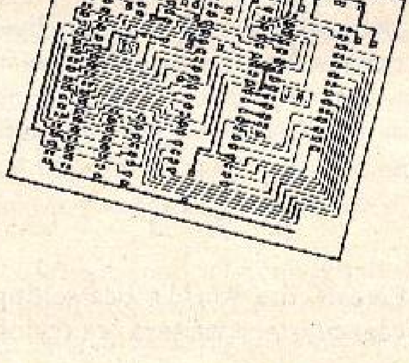
The test is being run in a factory which is open 12 hours a day, six days a week. It was decided not to operate a continuous test so that the computer didn't overheat.

The joystick is operated by a drill linked to a lever which manipulates it at an appropriate rate.

The program driving it was written by Christian Urquart, co-author of Daley Thompson's Decathlon, who is supervising the testing for Konix.

Circuit Designer

Software aimed at helping CPC hobbyists to design printed circuit boards has been published by John Morrison.



Morrison said he has been using the program himself for some time and finally decided to market it.

PCB Designer requires a 6128, or CPC464 with 64k expansion, plus AMX mouse.

Menu driven and featuring zoom editing, the software allows the user to define symbols.

Chess on the line

There was a double bonus for London schoolboy Michael Hennigan when he won the world's first major chess tournament conducted via electronic mail.

Apart from the British Chess Federation's under-18 quick play championship, he also won a CPC6128 for himself and a PCW8256 for his school.

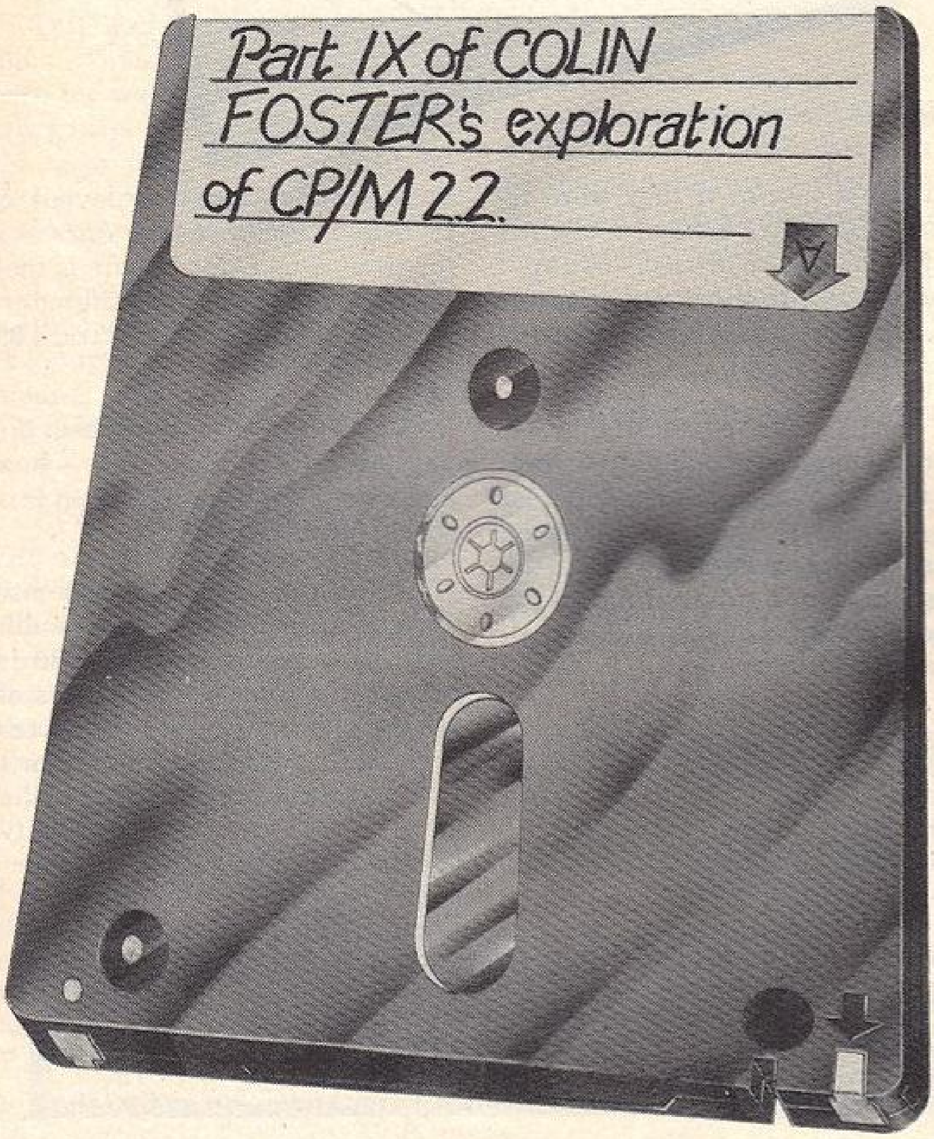
The prizes were donated by CDS Software whose Colossus 4 Chess program was used by the tournament's 560 entrants.

The last three rounds of the competition were staged on MicroLink's electronic mail system and shown move by move on Ceefax.

MicroLink also provided individual electronic mail-boxes for each competitor and linked them to Ceefax. Every move in each game was shown as it happened.

In the final Michael, aged 16 from City of London School, beat 17-year-old Aaron Summersdale from Elliot School, Richmond, in a tense replay after a three hour marathon first match was drawn.

Going round in circles to retrieve information at speed



Part IX of Colin Foster's exploration of CP/M 2.2.

Let's take a closer look at what discs are, how they work and how CP/M stores information and programs on them.

Our Amstrads use 3inch single sided, double density discs. This means that although we can use both sides of a disc our disc drive can only look at one side at a time without us turning the disc over — much like a cassette tape.

More expensive disc drives than the ones Amstrad use are double sided. This means that both sides can be accessed without having to remove the disc and turn it over.

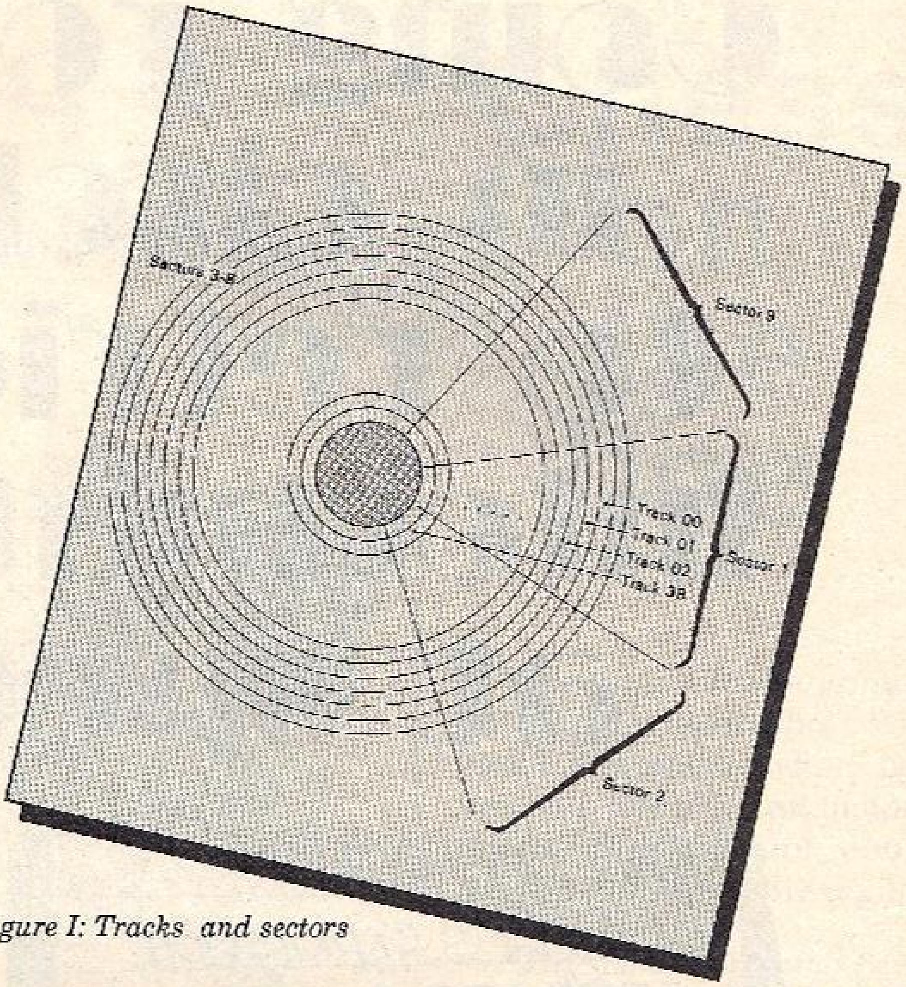


Figure I: Tracks and sectors

Double Density refers to the method used to physically record data on the disc. All modern disc drives use the double density MFM method which gives twice the amount of storage space on a disc than that provided by the original Single Density FM system standardised by IBM when floppy discs were first invented.

So much for the history lesson. How do discs actually work? A floppy disc is simply a disc of plastic — 3in in diameter in our case — coated in the same sort of magnetic material as a tape.

Before a computer can use a disc it must be formatted. This process writes a preset data pattern all over the disc, dividing it up into areas which the computer can then address by number. These areas are called sectors, and on the Amstrad they can each hold 512 bytes of data — 0.5k.

On a record the music is engraved serially in one long line, which spirals in from the outside of the record to the centre to pack as much information in as possible.

We could obviously use a similar system to store our sectors on a magnetic disc. However, unlike a record, we rarely want to listen to the first half of the disc to get to some information stored in the middle.

Also, without some sort of complicated indexing system the computer would not be able to GOTO a particular sector if they were recorded serially. Instead a slightly different system is used — we split the disc up into 40 concentric circles of data called tracks, rather than a single long spiral.

Each track then holds nine sectors of information, and we can get at any one of them effectively as fast as any other simply by stepping the read head of the disc drive out to the required track and waiting till the sector we want passes under it, allowing us to read it.

As the disc rotates at 300rpm, this doesn't take long. This system is called random access to distinguish it from serial access as used in records, compact discs and tapes.

Figure I shows the layout of tracks and sectors on our discs as a diagram. One point to note is that the first track, Track 0, is the outermost circle on the disc and track 39 is the innermost.

So far we have described the physical format of our disc — how and where the information on it is stored physically.

Different computers may use variations on this by having a different number of tracks — 80 is common — and different sizes of sectors — anything from 128 bytes to 1Kbyte — with anything from 4 to 26 sectors per track. This is the basic reason for incompatibility between discs from different computers.

Figure II: Contents of a directory FCB

Byte :	0	1	2	3	4	5	6	7	8	9	10	11
Contents :	user	F	I	L	E	N	A	M	E	T	Y	P

Byte :	12	13	14	15	16	.	.	31
Contents :	ex	0	0	rc	d0	.	.	d15

These differences are, however caused more by the whims of hardware manufacturers than by any inherent requirements of different operating systems. Similar basic recording systems of tracks and sectors are used on all systems from CP/M micros such as ours and IBM PCs running MS-DOS to large mainframes.

This leads us on to logical formats. These have nothing to do with the way data is actually written on the disc but rather control the way sectors are grouped together — purely abstractly — by an operating system to create files.

Logical formats also control the method used to build a directory — a group of sectors which instead of holding data contain an index of files which points to their locations on the disc.

CP/M is designed to run on any Z80 or 8080 based computer, and therefore has to be immune to the chaos caused by the vast variety of physical disc formats used on different micros. To make this possible the BIOS — the machine-specific part of CP/M — does all the worrying about physical format.

CP/M — and all programs written to run under CP/M — treats all discs as having an idealised sector size of 128 bytes. Thus as far as CP/M is concerned we have 36 sectors of 128 bytes per track and our BIOS has to translate this into the physical layout of nine sectors of 512 bytes per track whenever CP/M wants to read or write a 128 byte sector from or to the disc.

The process of buffering carried out by the BIOS to do this is known as blocking and deblocking, and I won't go into it in any more detail here. To avoid confusion, if I haven't caused it already, we'll refer to CP/M's 128 byte logical sectors as records, and treat our disc as having 36 records per track.

On a system format disc tracks 0 and 1 are reserved to hold the code which makes up CP/M itself, and are therefore not used to hold data. The first 16 records on track 2 then become our directory.

This is a bit like the contents page in a magazine — it tells us where to look on the disc to find a particular file of data or program.

To reduce the amount of information the directory needs to hold, and so also its size, CP/M imposes an extra level of organisation on itself by grouping records together into blocks, each of eight records and so 1k in size.

These blocks are an abstract concept and invisible to programmers in the normal course of things — they just exist to make CP/M's job easier. However we have to be aware of their existence to understand how the directory is constructed.

Each record in the directory contains four 32 byte File Control Blocks, each of which is of the form shown in Figure II. Each FCB contains information on the location on the disc of one part, or extent, of a file.

The first byte contains the user number of the file. A value of &E5 means that the file has been deleted. When CP/M deletes a file it doesn't actually wipe out the contents of the file on disc, it simply sets the user byte in all the file's extents to &E5. This allows us to write clever little programs like UNERASE.COM to recover files we've accidentally blown away.

Figure III: Contents of Disc Parameter Block (DPB)

Byte	Contents		Value	
0,1	SPT	Records per Track	36	&24
2	BSH	Block Shift factor	3	&03
3	BLM	Block Mask	7	&07
4	EXM	Extent Mask	0	&00
5,6	DSM	Total Storage Capacity	170	&AA
7,8	DRM	Max. directory entries	63	&3F
9	AL0	Directory Blocks	192	&C0
10	AL1	Directory Blocks	0	&00
11,12	CKS	Checked Records	16	&10
13,14	OFF	Reserved Tracks	2	&02

Bytes 1 to 8 contain the filename in Ascii upper case and bytes 9 to 11 hold the file type, again in Ascii upper case. Note that the "." we normally put into "filename.typ" is not put into the directory.

Bytes 12 and 15 are the extent number and record count values respectively — and we'll return to these later. Bytes 13 and 14 are always 0 — they are reserved for internal use by the BDOS during file operations.

Bytes 16 to 31 are the index, which tells us where to find the data for the file. Each byte is the number of a block which has been allocated to the file.

As we saw previously CP/M on the Amstrad uses a 1k block size. Each FCB has 16 bytes reserved for holding allocation information and so can map up to 16k of a file into each of its extents.

If the file is 16k or less in size then obviously it only requires one extent and its extent number — byte 12 of its FCB — is set to 0. If a file is larger than 16k it will need more than one extent to hold its directory information and subsequent extents will be numbered 1, 2 and so on.

Byte 15 of the FCB, the record count, is the exact number of records which contain valid data held within the blocks pointed to by an extent. As an extent can index up to 16k, and a record is 128 bytes in size obviously the maximum number of records held by an extent is 16x8, or 128 (&80).

Byte	Contents	Value	
15	First (physical) sector number	65	&41
16	(Physical) sectors per track	9	&09
17	Gap length (R/W)	42	&2A
18	Gap length (Format)	82	&52
19	Filler byte	229	&E5
20	$\log_2$(sector size) – 7	2	&02
21	Records per (physical) sector	4	&04
22	Current Track (BIOS variable)	0	&00
23	Not aligned/Aligned (BIOS variable)	0	&00
24	Auto format select (BIOS variable)	0	&00

Figure IV: Contents of Expanded Disc Parameter Block (XPB)

Thus extents of large files which are completely filled will have record counts of &80. If the last extent of a file is not completely filled it will have a count less than this.

You may well come across terms in CP/M literature such as extent folding, and information which suggests that the structure of extents is more complex than I have described here. Unfortunately life is rarely so simple!

The problem here is to do with the fact that most computers use different logical formats on their discs, many of which have block sizes larger than 1k.

This means that each extent can then in theory index more than 128 records, and we have to fold logical extents into each physical extent. Yes my eyes have started to glaze over too. Fortunately on the Amstrad we don't have to worry about this sort of thing.

I said earlier that our directory is 16 records, or 2k, in size. It actually takes up the first two blocks on the disc — 0 and 1. This therefore means that it can hold 4x8x2 entries, which makes 64.

Note that this is the maximum number of 32 byte entries — that is, extents, not files. If the directory gets full we can't put anything else on to the disc until space has been made by deleting something.

This is a separate problem to the more common one of simply running out of data space on disc. The directory size of 2k is an arbitrary one, determined by Amstrad when they designed the disc format. Other CP/M computers use different sizes of directory. For example

128 entries using a 4k directory is common.

So that programs can find out information like this about the system on which they are running the BIOS maintains a table of data called the Disc Parameter Block, or DPB. This contains information on our logical disc format.

Figure III lists the contents of the DPB by the shortened mnemonics by which they are known. I'll now explain what they all mean.

SPT is Sectors Per Track. This however refers to CP/M's logical sectors of 128 bytes — our records — not our real physical sector size of 512 bytes. As we saw earlier CP/M considers us to have 36 (&24) records per track.

BSH — Block SHift factor — and BLM — BLock Mask — are concerned with the size of our data blocks — 1k each on the Amstrad. This requires a Block Shift factor of three and a Block Mask value of seven, but I'm not going to try and explain how these are derived here.

EXM is the EXtent Mask, and is used to control the extent folding I mentioned earlier. On our nice, simple Amstrad this value is just 0.

DSM is one less than the total data storage capacity of the disc in units of blocks. So for us we have 36 records, or 4.5k, per track times 38 tracks — we can't use the reserved system tracks — which gives a total of 171k. As our block size is just 1k we have a DSM value of 170 (&AA).

DRM is one less than the maximum number of directory entries. We saw earlier that we could have 64 entries because of our 2k directory size, so our DRM is 63 (&3F).

AL0 and AL1 make up a 16 bit word — with AL0 as the more significant byte — which tells us how many blocks are allocated to the directory. It does this by setting a bit of the word, starting at the most significant end of AL0, for each block allocated. We use two blocks — 0 and 1 — which gives us a word value of &C000 for AL0/AL1 — binary 1100 0000 0000 0000.

CKS — CHecked Sectors — tells CP/M the number of directory records which must be read and checked when a disc is accessed to discover if a different disc has been inserted into the drive without being logged in — the cause of the infamous BDOS ERR ON A. Obviously with simple removable floppy discs such as ours we would need to check the entire directory on a disc to be able to be sure it hasn't changed, so we have a CKS value of 16 (&10) records.

The last entry in the standard DPB is OFF — OFFset. This tells CP/M the number of tracks which we have reserved on our discs to hold the code for CP/M itself — two in our case.

This is what makes the difference between system and data formats on the Amstrad. Data format discs don't have CP/M on them and so have an OFF value of 0 to give them an extra two tracks, or 9k of data space.

However Amstrad has gone one better than standard CP/M. We have an Expanded Disc Parameter Block, or XPB, which gives us more information than normal on CP/M systems. The XPB follows immediately after the standard DPB in memory.

Our programs can discover the base address of the DPB by executing a BDOS function call 31 — we'll be looking at disc function calls next month. Figure IV shows the contents of the XPB, addressed by offset from the start of the standard DPB.

The XPB gives programs information about the physical format of our discs, whereas the DPB only describes the logical format. We'll look at these values more closely in a future article, when we move on to look at the BIOS and disc controller hardware in more detail.

However you will already recognise the significance of some of the entries – physical sector size, sectors per track and records per sector, for example.

• Next month we'll see what additional function calls are available to use from the BDOS to let us read and write files on disc.

Logically, you should know where to *draw the line*

Part VIII of the Amstrad graphics series by Geoff Turner and Michael Noels

You've probably already noticed that when you DRAW a line with a selected graphics pen, the line is always drawn in the ink colour that is filling that pen.

It seems fairly obvious that whatever the colours already on the screen, our line will always overwrite them. And whatever the background, the line is the colour of the ink filling the selected graphics pen.

Put the Amstrad into Mode 0 and try changing the background colour to any colour of your choice using CLG. Then draw a line across the screen in pen 3 like this:

```
CLG n

DRAW 639,399,3
```

Notice that whatever number (n) you use to clear the graphics screen, the line will always be drawn in red. This is as we'd expect, since we've specified pen 3 with the DRAW command and, until we do something about it, pen 3 is filled with bright red ink — ink number 6.

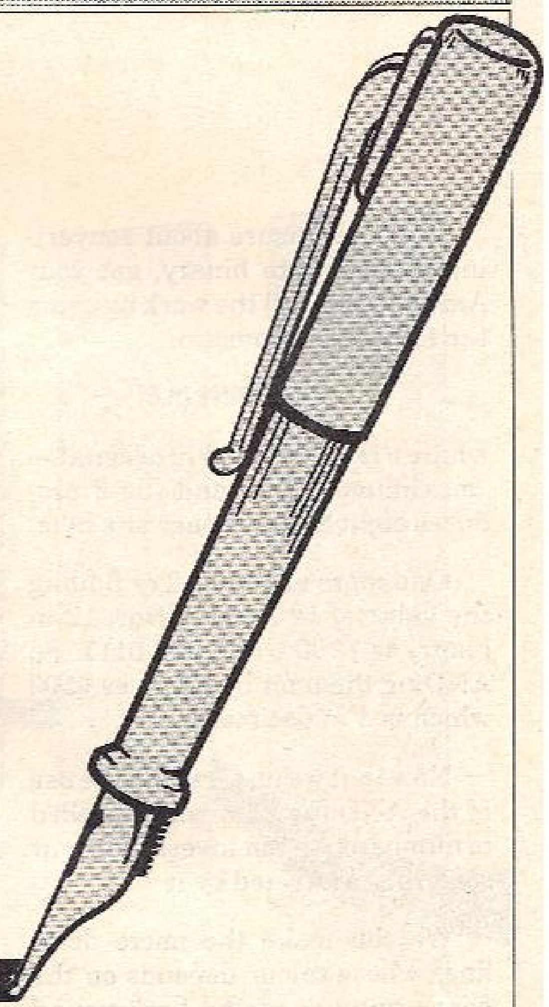
For the moment, only enter lines 10 to 140 from Program I plus line 300 which prevents the ready message reappearing after the drawing is complete. We can add the other lines later.

In this shortened form, Program I plots 16 strips of colour down the screen, then draws a single line across them with graphics pen 1. You'll see that the line is drawn in bright yellow — pen 1, filled with ink number 24 — across all 16 strips of colour. This is much as we'd expect from what we said earlier.

It appears, then, that the graphics pen completely disregards the colour of the background when it's used to draw lines. The same thing happens when using the graphics pen to plot individual points on the

```
10 REM PROGRAM I
20 BORDER 0
30 MODE 0
40 FOR colour=0 TO 15
50 FOR strip=1 TO 40 STEP 4
60 MOVE 40*colour+strip,0
70 DRAW 40*colour+strip,399,colour
80 NEXT
90 NEXT
100 LOCATE 1,4
110 PRINT"NORMAL"
120 PRINT CHR$(23);CHR$(0)
130 MOVE 0,320
140 DRAW 639,320,1
150 LOCATE 1,9
160 PRINT"AND"
170 PRINT CHR$(23);CHR$(2)
180 MOVE 0,240
190 DRAW 639,240,1
200 LOCATE 1,14
210 PRINT"OR"
220 PRINT CHR$(23);CHR$(3)
230 MOVE 0,160
240 DRAW 639,160,1
250 LOCATE 1,19
260 PRINT"EOR"
270 PRINT CHR$(23);CHR$(1)
280 MOVE 0,80
290 DRAW 639,80,1
300 WHILE INKEY$="":WEND
```

Program I



screen. The point plotted overwrites the background.

However, it doesn't have to be like this — it is possible to change the way that the graphics pen writes over the background colours.

The background can be allowed to interact with the graphics pen so that the line drawn isn't always the colour of the ink filling the pen used. In other words, the background can affect the colours that are drawn on it.

There are in fact four different ways that the graphics pen can interact with the background colours. The first we've already comes across — it's where the background is ignored by the line being drawn.

For the other three ways in which the colours are affected, we need to study some rules of logic. This is because the way that the background affects the drawn line is determined by three logical operations, AND, OR and EOR.

So, before we see how pen and paper colours interact, we'll use some simple numbers to illustrate the rules of logic.

Now, enter this command into your computer:

```
PRINT 2 AND 1
```

The micro should respond with the answer 0. If you're unfamiliar with logic functions, you may well have expected the answer to be 3, as you've probably always been taught that 2 and 1 equals 3.

The truth is that 2 PLUS 1 equals 3, but in our examples we have used the logical AND function unstead of adding the two numbers together. It's just maths using a different rule. You already know the rule for adding — in the next paragraph you'll learn the rule for ANDing.

0 AND 0	0
0 AND 1	0
1 AND 0	0
1 AND 1	1

Figure 1:AND logic truth table

The AND function examines the binary values of the two numbers, and then compares each corresponding pair of bits. If both bits are set to the value 1 then the result of the AND function will also be 1.

You see the AND function is saying "If the bit from the first binary number is 1 and the corresponding bit from the second binary number is 1 then the resulting bit will be 1. Otherwise the result will be 0".

If you can't see that, take it step by step applying the rule to each of the bits in turn. The number 2 in binary is 10, while the number 1 in binary is 01. If we consider each bit of the two numbers and apply our AND logic rule, then the result will be 00 in binary, which is 0 in decimal.

Compare the last bit or figure in the two numbers. The first is 0, the second 1 so ANDing them gives us 0. The second pair of bits are 1 and 0 so again we get 0 from the AND. We soon arrive at the answer. 10 AND 01 is 00.

To simplify things we use what is known as a truth table. This shows evey possible combination of bits for a particular logical function. Figure 1 shows how the AND function affects each combination of two binary bits.

Notice that in the above example we're only using two bits for simplicity, but the rules can be applied equally well to any number of bits. Usually bits are dealt with in bytes of eight at a time.

You might like to try your hand at ANDing some other numbers together. What is 7 ANDed with 9? Remember, change the numbers to binary first — 0111 and 1001 in this case. Then apply the AND rule to each successive pair of bits. Finally change the binary result (0001) back into decimal (1).

If you're unsure about converting decimal into binary, get your Amstrad to do all the work by using the following command:

```
PRINT BIN$ (n,8)
```

where n is the number in decimal — maximum 255 — and the 8 produces eight bits of binary or a byte.

One more example. Try finding the value of 12 AND 7. Now 12 in binary is 1100 while 8 is 0111. So ANDing them bit by bit gives 0100 which is 4 in decimal.

Now that we understand the use of the AND function when applied to numbers, we can investigate how colours are affected by it.

We can make the micro draw lines whose colour depends on the paper number of the background and the pen number of the graphics pen being used. These are ANDed together to give the number of the pen that actually draws the line.

To instruct the graphics pen to draw with AND logic, we need to use another of our control codes. Character number 23 is used, followed by a number in the range 0 to 3 — 0 is the default value which is used for normal drawing. Parameter number 2 is used to specify the AND function, so we need to enter:

```
PRINT CHR$(23);CHR$(2)
```

to see AND in action.

Add lines 150 to 190 to Program I. Now when you run the program, a second line will now be drawn across our coloured strips. This is because line 170 has told the micro to draw using AND to interact the background colour with the graphics pen.

Notice that although we have used pen 1 — bright yellow — the drawn line actually changes colour as it passes over each strip. Instead of the paper being ignored and overwritten as before, it's now affecting the resultant line.

0 OR 0	0
0 OR 1	1
1 OR 0	1
1 OR 1	1

Figure II: OR logic truth table

The second line has been ANDed with the background colours. The rules for ANDing the paper and pen colours are exactly the same as for ANDing numbers.

In fact, all we need to do is find the numbers of the background and foreground pens and AND them together. When we know the result we have the number of the pen used to draw the line when the graphics pen crosses that particular background colour.

As an example of this, let's take a background colour of cyan as produced by pen 2. This is the third strip along from the left. We are going to draw over this in bright yellow with graphics pen 1.

If we AND the numbers 2 and 1, we end up with the result 0 and the resultant line is drawn in pen 0 which is filled with blue ink. So you will see our second line is drawn in blue when it passes over the cyan strip.

Looking at the next strip along, you'll see that it's bright red, drawn in pen 3. ANDed with pen 1 this gives 1 — 11 AND 01 gives 01. So the bit of the line that passes over the red background is drawn using the yellow ink of pen 1.

Try figuring out what's happening for the other strips. You'll see that in each case the colour of the line is that of the ink filling the pen number found by ANDing the numbers of the paper and graphics pen. As you can see, you might pick a yellow graphics pen but you don't always get a yellow line.

The second function that we are going to examine is the logical OR. Once again we need to convert our numbers to binary before applying the OR rule.

The OR rule says that "If either the bit from the first binary number

0 XOR 0	0
0 XOR 1	1
1 XOR 0	1
1 XOR 1	0

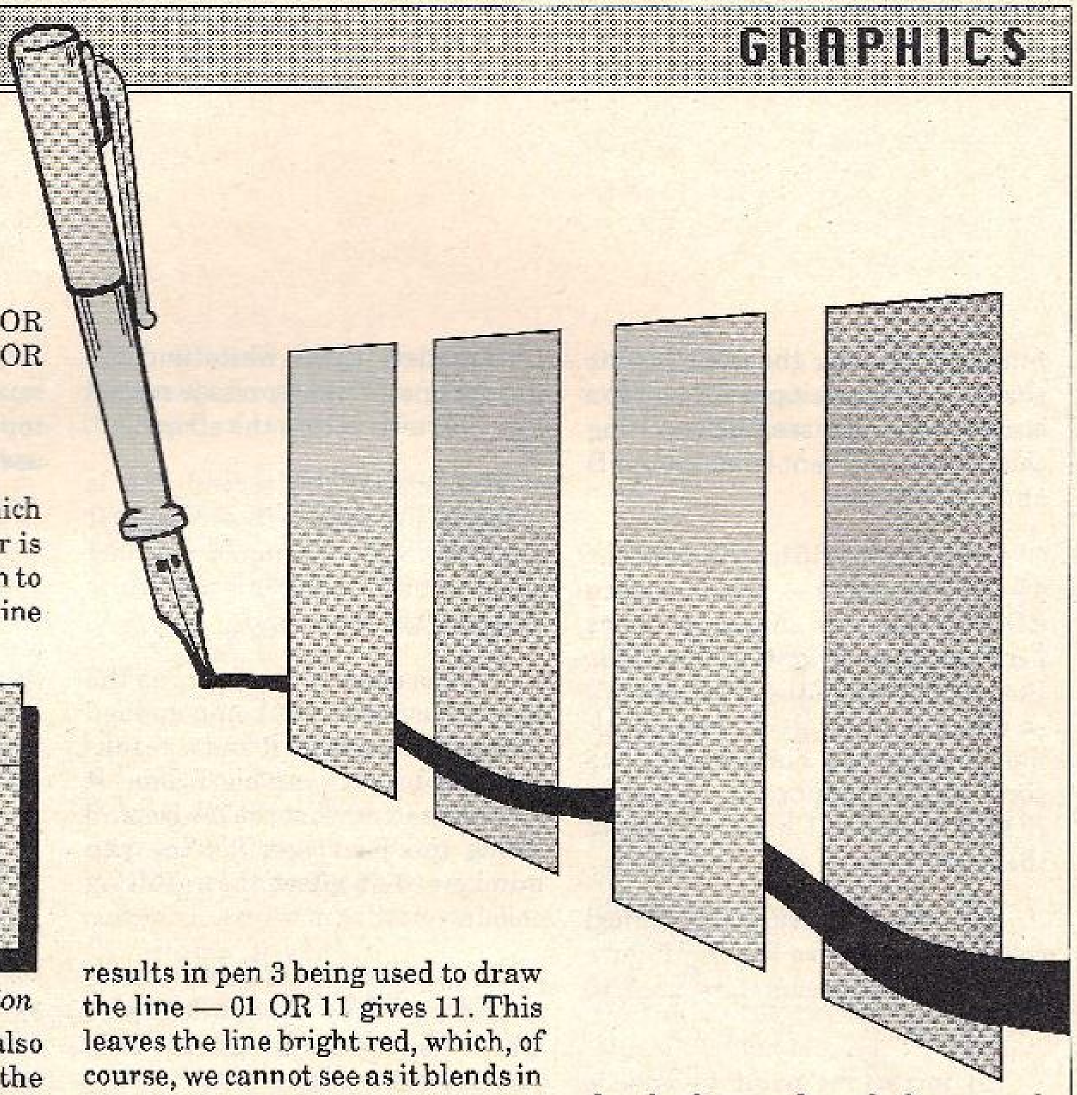
Figure III: XOR logic truth table

is 1 or the corresponding bit of the second is 1 then the result will be 1. Otherwise the result will be zero".

Figure II is our truth table for the OR function.

Notice that the result is 1 if one or both of the bits is 1.

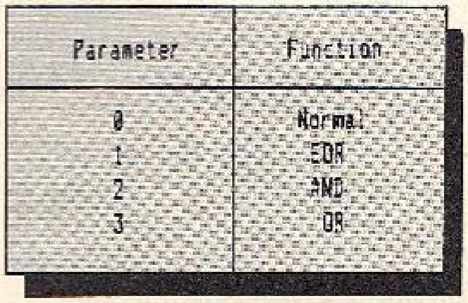
Once again we can apply the OR function to the pen and paper colours on screen. If you now add lines 200 to 240 on to the end of Program I we will end up with a



third line drawn this time using OR logic. The command for the OR function to be used is:

```
PRINT CHR$(23);CHR$(3)
```

You'll find this in line 220 which specifies that the paper number is to be ORed with the graphics pen to find the pen number that the line will be actually drawn in.



Parameter	Function
0	Normal
1	EOR
2	AND
3	OR

Figure IV: Logical colour selection

Notice that the third line also changes colour as it passes over the strips. This time the colour change sequence is completely different from the previous one when we used AND logic.

Looking at the fourth strip, we are ORing pen 1 — bright yellow — with paper 3 — bright red. This results in pen 3 being used to draw the line — 01 OR 11 gives 11. This leaves the line bright red, which, of course, we cannot see as it blends in with the background.

The next strip has pen 4 — white — ORed with pen 1 — yellow — resulting in pen 5 being used to draw a black line over the white strip. Try figuring out the logic of the rest of the colours.

The third and final function that we are going to look at is the EOR function, sometimes known as the Exclusive OR or even XOR.

The rule for EOR says that "If one bit and only one bit is set to 1 then the result will be 1 otherwise the result will be zero".

Once again a look at the truth table, shown in Figure III, will show us all the different combinations. Notice that its logic is subtly different from that of the OR operator.

To demonstrate the EOR function, we need to add the final lines 250 to 290 on to Program I. The:

```
PRINT CHR$(23);CHR$(1)
```

of line 270 has the micro EORing the background and foreground colours.

Now when we run the program, we have four lines drawn across the coloured strips, one for each logic

```
10 REM PROGRAM II
20 MODE 0
30 INK 3,6
40 FOR strip=192 TO 319 STEP 4
50 MOVE strip, 100
60 DRAW strip,300,3
70 NEXT
80 FOR strip=320 TO 448 STEP 4
90 MOVE strip,100
100 DRAW strip, 300,4
110 NEXT
120 PRINT CHR$(23);CHR$(1)
130 MOVE 0,250
140 DRAW 639,250,1
150 WHILE INKEY$="":WEND
160 INK 3,26
170 WHILE INKEY$="":WEND
180 MOVE 0,150
190 DRAW 639,150,1
```

Program II

```
10 REM PROGRAM III
20 MODE 0
30 PAPER 0
40 CLS
50 FOR strip=192 TO 319 STEP 4
60 MOVE strip, 100
70 DRAW strip, 300,12
80 NEXT
90 FOR strip=320 TO 448 STEP 4
100 MOVE strip,100
110 DRAW strip,300,3
120 NEXT
130 PRINT CHR$(23);CHR$(3)
140 INK 13,24
150 FOR lines=1 TO 20
160 DRAW RND(1)*640,RND(1)*400,1
170 NEXT
180 INK 7,26
190 FOR lines=1 TO 20
200 DRAW RND(1)*640,RND(1)*400,4
210 NEXT
```

Program III

function. Again, the fourth line changes colour as it passes over the strips, but as you see, the resulting colours are different from the AND and the OR lines.

Consider the fifth strip in paper 4 — bright white — which we are EORing with our chosen graphics pen, pen 1 — bright yellow. This time the result of the EORing is 5, so pen 5 is used giving us a black line on the white background. The sixth strip in paper 5 — black — EORed with pen 1 has that part of the line drawn in pen 4 — white.

And so we've covered the logical colours as they are known. Figure IV shows the parameters used to achieve them.

An important point to note is that when we apply any of the logic functions to our drawing, the resultant numbers refer to the pen numbers to be used and not to ink numbers.

All the ANDing, ORing and EORing does is select a pen and use the ink colour that happens to be in filling pen at the time. This is demonstrated in Program II.

Here, two coloured strips are drawn using graphics pens 3 and 4 — lines 60 and 100. These colours will be bright red and bright white. At line 140 a line is drawn across the strips using graphics pen 1 which, as we haven't done anything about it, is the default colour, bright yellow.

The line is drawn using the EOR function, which results in a bright cyan line over the red strip and a black line over the white strip.

Line 150 waits for a key to be pressed before the ink colour for pen 3 is changed to 26 — bright white. After the change of ink our red strip immediately turns white and following another key-press a second line is drawn across the strips.

Notice that the second line is identical in colour to the first one. It appears to have ignored the fact that the left-hand strip is now white when before it was red.

This is correct, however, as the logic function is still being applied to the pen numbers 3 and 4 resulting in a bright cyan/black line. It doesn't matter what the ink colours filling the pens are, it's the pen numbers that affect the resulting colour.

In the last case, it didn't matter that pen 3 has been filled with white ink, the EOR logic works in just the same way.

It is important to understand this point now, as later we will be changing ink colours quite often to demonstrate some useful techniques.

Combining colours logically as we have seen affects the colour of the lines drawn. Instead of just getting the colour filling the graphics pen, we get other colours depending on the logic used.

Having learnt all this, how can we put these effects to good use? The first thing we're going to look at involves producing multiplane images.

When we draw or plot anything in colour, we know that it has to share the screen with all the other colours. At any given point or pixel on the screen, we can only display one colour at a time.

If a particular point is already plotted in, say, blue and we wish to display a red spot at the same point, then we obviously have to wipe out the blue spot to replace it with red.

However, imagine if we had a number of different screens laid on top of one another, each one to be used for its own particular colour.

We could, for example, plot a red point on the front screen, and a yellow point on the second screen. Of course we wouldn't see the yellow spot as it would be obscured by the red one. If, however, we now removed the red spot, then the yellow one would now become visible through the clear front screen.

In this example the red screen has been placed at the front and so has priority over the yellow screen. We could have another screen representing green placed behind the yellow screen. In this case, yellow and red would both take priority over green. All the colours would be there but you'd only see the front one.

If you have ever played arcade-type games on your micro, you will probably have seen this effect when applied to animation. Objects moving around the screen will appear to go behind or in front of other objects depending upon the priority given to each colour. Let's see how this works in practice.

Program III once again draws two coloured strips, this time in red and green. Lots of random bright yellow lines are drawn across the screen. Notice how the lines appear to pass in front of the green strip but behind the red strip.

In this example we have given priority to red over yellow and yellow over green. To achieve this it was necessary to change one ink colour at line 140 in order to produce the desired effect. Let's examine how Program III works.

First of all the background is drawn in the default pen 0 — blue. As we are using the OR function —

line 130 — then the background colour can be effectively forgotten about as any number ORed with 0 stays the same. We have then drawn one strip in pen 12 —bright green, ink 18—and the other one in pen 3 — bright red, ink 6. The lines are drawn using pen 1 — bright yellow, ink 24.

Now when pen 1 passes over something drawn with pen 12, the result of the OR function is 13. Pen 13 would normally produce pastel green, but in line 140 we've changed this to ink 24 — bright yellow — thus resulting in bright yellow lines over the bright green strip.

The point to grasp is that we found the result of ORing the two pens and then filled this pen with the colour we wanted to see. In this case we replaced the pastel green ink with yellow.

When a line drawn with pen 1 passes over the bright red strip — pen 3 — the result of the OR function is 3. This results in bright red lines over the bright red strip, which of course we cannot see. These lines appear to pass behind the bright red strip.

The same results could be achieved several ways by choosing an appropriate combination of inks linked to one of the logic functions.

Probably the easiest way of achieving this multiplane effect is by deciding what the end result is going to be and then working backwards selecting suitable colours.

As an exercise you may like to try changing the colour priority in Program III so that the yellow lines pass in front of red and behind green.

So that you don't have to calculate the values of every possible AND, OR and EOR combination, we can use Program IV to print out the result of every calculation.

You will find the tables produced by Program IV invaluable when selecting ink combinations for a particular application.

Copies of the tables are reproduced in Figure V.

If you're fortunate enough to have a printer connected to your Amstrad, you may like to amend Program IV to print out a copy of the tables. You'll need to change every PRINT statement to redirect the output to stream number 8 which is the printer stream.

As an extension to Program III try adding the following lines:

```
180 INK 7,26
190 FOR lines= 1 to 20
200 DRAW RND(1)*640,RND(1)*400,4
210 NEXT
```

This results in bright white lines which pass in front of red and behind green. We had considered that red was the foreground colour followed by yellow and then green. Now we have white lines which appear in front of red but behind green. A genuine optical illusion ... or is it just a bit of computer trickery? Incidentally have you observed what happens when white lines collide with yellow lines? Black holes, maybe?

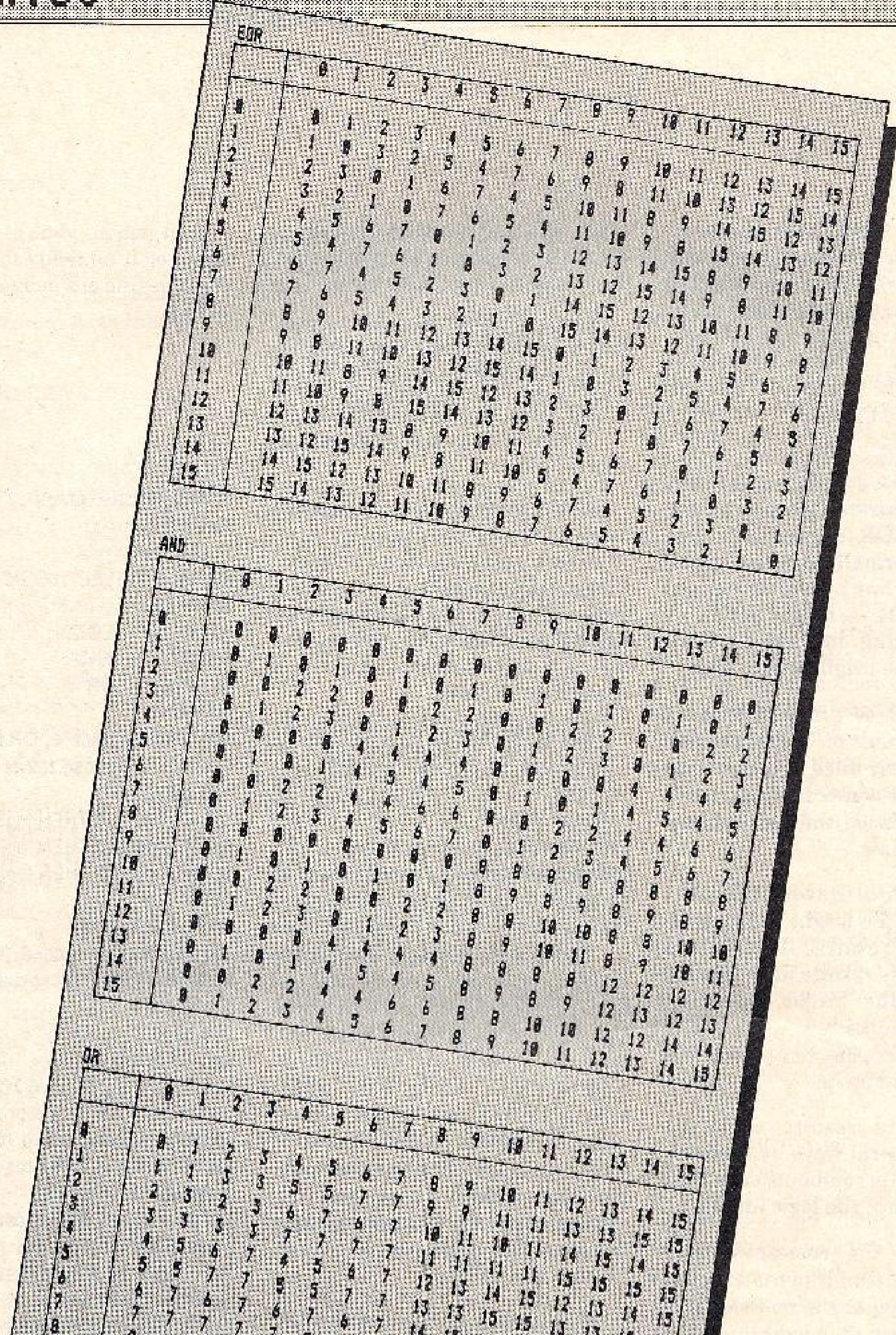
And that's it for this month. Play around with the logical functions until you feel at home with them. They're one of those things that seem complicated in theory but soon become easy with practice.

Always remember that the AND, OR, and EOR refer to the pen/paper numbers, not the ink numbers.

And when you've grasped all that, you'll be ready for next time, when we find out more about logical colours.

```
10 REM PROGRAM IV
20 MODE 1
30 choice=0
40 PRINT"SELECT LOGIC FUNCTION"
50 PRINT
60 PRINT"1. EOR"
70 PRINT"2. AND"
80 PRINT"3. OR"
90 PRINT
100 WHILE choice <1 OR choice>3
110 INPUT"Choose 1, 2 or 3 ";choice
120 WEND
130 IF choice=1 THEN logic$="EOR"
140 IF choice=2 THEN logic$="AND"
150 IF choice=3 THEN logic$="OR"
160 MODE 2
170 PRINT logic$;
180 FOR background=0 TO 15
190PRINTTAB(4*background+10);ba
ckground;
200 NEXT
210 PRINT:PRINT
220 FOR foreground=0 TO 15
230 PRINT TAB(0); foreground;
240 FOR background=0 TO 15
250 IF choice=1 THEN result=foregr
ound XOR background
260 IF choice=2 THEN result=foregr
ound AND background
270 IF choice=3 THEN result=foregr
ound OR background
280 PRINT TAB(4*background+10);
result;
290 NEXT
300 NEXT
310 WHILE INKEY$="":WEND
320 GOTO 20
```

Program IV



EOR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13	12
4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
5	5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5	4
12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2
14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

AND

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2
3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	0	0	0	0	4	4	4	4	0	0	0	0	4	4	4	4
5	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
6	0	0	2	2	4	4	6	6	0	0	2	2	4	4	6	6
7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8
9	0	1	0	1	0	1	0	1	8	9	8	9	8	9	8	9
10	0	0	2	2	0	0	2	2	8	8	10	10	8	8	10	10
11	0	1	2	3	0	1	2	3	8	9	10	11	8	9	10	11
12	0	0	0	0	4	4	4	4	8	8	8	8	12	12	12	12
13	0	1	0	1	4	5	4	5	8	9	8	9	12	13	12	13
14	0	0	2	2	4	4	6	6	8	8	10	10	12	12	14	14
15	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

OR

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	3	3	5	5	7	7	9	9	11	11	13	13	15	15
2	2	3	2	3	6	7	6	7	10	11	10	11	14	15	14	15
3	3	3	3	3	7	7	7	7	11	11	11	11	15	15	15	15
4	4	5	6	7	4	5	6	7	12	13	14	15	12	13	14	15
5	5	5	7	7	5	5	7	7	13	13	15	15	13	13	15	15
6	6	7	6	7	6	7	6	7	14	15	14	15	14	15	14	15
7	7	7	7	7	7	7	7	7	15	15	15	15	15	15	15	15
8	8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
9	9	9	11	11	13	13	15	15	9	9	11	11	13	13	15	15
10	10	11	10	11	14	15	14	15	10	11	10	11	14	15	14	15
11	11	11	11	11	15	15	15	15	11	11	11	11	15	15	15	15
12	12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
13	13	13	15	15	13	13	15	15	13	13	15	15	13	13	15	15
14	14	15	14	15	14	15	14	15	14	15	14	15	14	15	14	15
15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15

Figure V: Colour logic

A significant bit of sound programming ...

Nigel Peters explains the numbers behind the notes in Part VIII of his series on Amstrad sounds

We've come a long way in our exploration of the channel parameter of the SOUND command. We've seen how values of 1, 2 and 4 can be used to produce notes on channels A, B and C respectively.

Also we've learnt how they can be combined so that:

```
SOUND 7,100,100,7
```

produces the same note on all three channels.

We've found that adding 8, 16 or 32 causes the sounds to rendezvous with notes on channels A, B, and C. This means that when we try to produce a note on channel A with:

```
SOUND 33,100,200,6
```

we won't hear anything until we give it a rendezvous note on channel C using:

```
SOUND 12,200,200,5
```

You'll remember that both notes have to be tagged with the rendezvous parameter. Using a note such as:

```
SOUND 4,200,200,5
```

wouldn't have freed the note on the channel A queue. Try it and see.

Talking of the channel queues, each of which can hold four notes in addition to the one that's playing, leads us to the hold parameter.

By adding 64 to the channel parameter of a SOUND command we are able to ensure that when that notes reaches the head of the queue it waits until it's released.

In practice this means that when we enter something like:

```
SOUND 1,300,100,7:
SOUND 65,400,100,7
```

we only hear the first note. The second note stays in the channel A queue until it's conjured up with:

```
RELEASE 1
```

Finally we've seen how we can join all these values into one combined parameter so that:

```
SOUND 81,200,100,7
```

produces a note on channel A that is not only held but also waiting for a rendezvous note on channel B. To bring the timid thing out of hiding we have to free it with a :

```
RELEASE 1
```

and then whistle it up with a note on channel B such as:

```
SOUND 10,1000,100,3
```

I hope that you've been messing around with the channel parameters of the SOUND command. There's no better way to learn. With the Amstrad's sound chip, practice is a lot better than theory. No doubt you've managed to get your channels in a twist more than once. The small Enter key, set up with:

```
KEY 139,"SOUND  135,0,0,0"+CHR(13)
```

has probably taken a lot of hammer as you tried to escape from a cacophony or a wall of silence.

However, while you've been using it, have you noticed that there's something strange about the channel parameter of the SOUND command? It's 135, a lot higher than any we've used previously.

As you might have guessed, it's yet another number that can be added to the channel parameter. This time it's one that causes the relevant channel or channels to be flushed or cleared of notes.

Try a sound such as:

```
SOUND 1,400,4000,7
```

Monotonous isn't it? Suppose now that you wanted to have another note playing on channel A. In the middle of a program you couldn't just hit the small Enter key. How would you do it?

Using:

```
SOUND 1,200,100,7
```

has no effect, it just gets stuck on the channel A queue and the first note carries on until it's finished.

number	bit set	result
1	0	uses channel A
2	1	uses channel B
4	2	uses channel C
8	3	rendezvous with A
16	4	rendezvous with B
32	5	rendezvous with C
64	6	hold until RELEASed
128	7	flush the channel

Table I: Channel parameter values and actions.

Why not use the flush parameter, 128? We want to clear channel A, which has parameter 1, so the channel parameter we need is 129 (1 + 128). This means that:

```
SOUND 129,200,100,7
```

will do the job, stopping the first note dead in its tracks. Now there's a note of pitch 200, loudness 7 playing one second.

In other words the flush parameter gives that note priority over any other note that's playing or waiting in the queue. A channel parameter of 129 clears all the other notes on channel A out of the way while 130 (2 + 128) and 132 (4 + 128) do the same for channels B and C. This is the case even if the notes on the queue are held or rendezvoused.

Entering:

```
SOUND 66,100,100,7
```

holds a note on the channel B queue.

bit number	7	6	5	4	3	2	1	0
binary byte	0	0	0	0	0	0	0	1

Figure I. Bit positions of binary 1.

bit number	7	6	5	4	3	2	1	0
binary byte	0	0	0	0	0	0	1	0

Figure II: Bit positions of binary 2.

However:

```
SOUND 130,400,200,7
```

gets rid of the first note — and any others that might be on the channel B queue — and plays one with pitch 400, loudness 7 for two seconds.

Similarly:

```
SOUND 12,100,100,7
```

is waiting for a rendezvous with a note on channel A. Any other notes on channel C will just have to take their places behind it in the queue until its date turns up. That is, of cource, unless it is the queue-jumping:

```
SOUND 132,1000,300,6
```

which gets rid of all the other notes and plays for three seconds.

As with our other parameters we can mix and match them. A value of 131 – 1+2+128 – clearing the channel A and B queues, while 133 – 1+4+128 – does the job for A and C. As you might guess:

```
SOUND 135,0,0,0
```

sweeps the notes from all the channels. Playing, held, rendezvoused or merely waiting, they all go. HOwever since this last note has a pitch volume and a duration of zero, no other note is played. In effect:

```
SOUND 135,0,0,0
```

gets rid of all previous SOUND commands. As you've found when you've had recourse to the small Enter key, this can be a blessed relief.

And the flush parameter, you'll be glad to know, is the final one we'll meet in our treatment of the channel parameter.

Table I sums them up. Notice that in the table there are some values labelled "bit set", ranging from 0 to 7. The reason that these are shown is that the channel parameter of the SOUND command is what is known as "bit significant".

This means that when the number, which is usually in decimal, is translated into binary, the 0s and 1s of the binary number are used as flags to switch different aspects of the channel parameter on and off. You don't have to know too much about it to understand the following.

To show what I mean, let's just take the simple case where the channel parameter of a SOUND command is 1. Now the binary equivalent of 1, is not surprisingly, 1 or rather 00000001 when we pad it to eight figures. If you don't believe me, maybe you'll believe your micro when:

```
PRINT BIN$(1,8)
```

produces:

```
00000001
```

Now each of these eight figures makes up one bit of what's known as a binary byte. The bit furthest to the right — in this case a 1 — is known as bit 0, the one to its left — here a 0 — as bit 1, the next as bit 2 and so on until the final bit, bit 7. Figure I shows the bit number in a byte.

Let's take the binary byte for 2 which, as we'll find from:

```
PRINT BIN$(2,8)
```

is:

```
00000010
```

Here bit 0 is 0, while bit 1 is a 1, the remaining bets (2 to 7) all being zero. Figure II shows this.

I don't know about you, but I find talking about bit 0 as 0 and bit 1 as 1 a bit confusing! I prefer to say that a bit is "set" if it's a 1 and "clear" if it's a 0. So, in this last example, only bit 1 is set — 1 — all the others are clear — 0.

Going on to our last channel parameter, 4, a quick:

```
PRINT BIN$(4,8)
```

gives:

```
00000100
```

Bit 2 is set, the rest are clear.

Now compare these with the "bit set" column in Table I. With a channel parameter of 1, we've seen that bit 0 is set. Looking at the table it says that the note uses channel A. So if that bit is set channel A is used.

Similarly, if bit 1 is set, B is used and if bit 2 is set, C is the selected channel. As you can see of the bit is set, then the result next to it occurs. This is what is meant by saying that the channel parameter is bit significant. Different actions occur according to which bit of the binary byte is set.

Now suppose we have a note playing on all channels with a parameter of 7. In binary this is 00000111. Try:

```
PRINT BIN$(7,8)
```

if you doubt me.

As you can see, bits 0, 1 and 2 are set. Looking at the table we can see that this means that notes will be produced on channels A, B and C.

It's the same for all the other parameter values up to 255. If you translate them into eight bit binary numbers you can see which functions are switched on and off by the individual bits.

Let's look at one last example. We've seen that a channel parameter of 135 clears all the channels. Now:

```
PRINT BIN$(135,8)
```

gives us the byte:

```
10000111
```

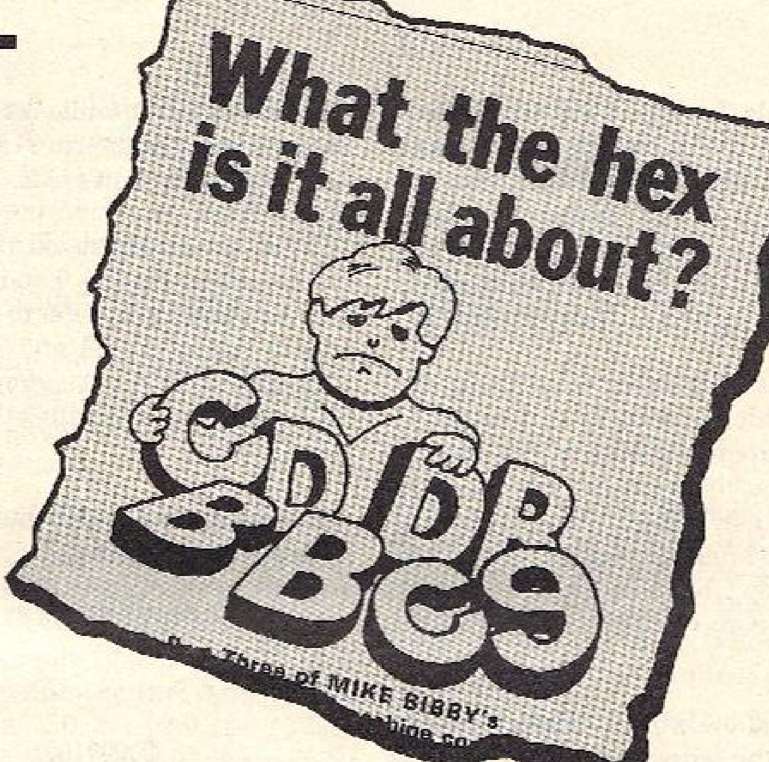
Looking at this bit by bit shows that the flush action started by bit 7 being set will affect channels A, B and C - as bits 0-2 are also set.

Try out other parameter values and examine their binary equivalents to see what will happen. It's fascinating to see how the action of a channel parameter of, say, 97 can be read from its binary equivalent — 0110001. Here bits 6, 5 and 1 are set, so it's obvious from Table I that we have a note held on channel A waiting for a rendezvous with one on channel C. Until you think about it, it seems almost uncanny that the binary version of a decimal number so closeley relates to the channel parameter.

And that's it for this month. By the time you understand the significance of bit significance you can leave channel parameters and go on to the SQ command. Your knowledge of the SOUND command will have increased a significant bit.

More About Hexer

by Mike Bibby



We're now at the stage where you might want to start saving and loading back the machine code programs you've created with Hexer. (Hexer was first published in the February 1987 issue of *Computing with the Amstrad*.)

Program I shows the lines you have to either add or alter to add these options.

Also, we've featured RAW, a full Z80 assembler, in this issue.

This new version of Hexer and RAW can quite easily be combined to produce a useful toolkit for developing machine code programs. Both can reside in the memory at the same time, provided they have different line numbers, and a simple menu can be added to allow one or the other to be selected.

First load Hexer, renumber it so that it starts at 10000 and save it as an Ascii file. Load RAW and merge Hexer, as follows:

```
LOAD "Hexer"
RENUM 10000
SAVE "Hexer",A
LOAD "RAW"
MERGE "Hexer"
```

Next, type Program II.

Hexer needs altering very slightly to enable it to run as RAW takes up a fair chunk of memory. The machine code must be located higher in the memory so change any reference of &2FF8 to &7FF8, and change any &3000's or 3000's to &8000 or 8000.

At the start of Hexer MEMORY is set to &2FF8. This would need to be changed to &7FF8 and the start address is now &8000.

When you've made these changes the program can be saved and then run.

Program I

```
150 PRINT "5. End program"
155 PRINT "6. Save code"
156 PRINT "7. Load code"
170 IF INSTR("1234567",b$)=0 GOTO
   160
180 b = VAL(b$) : ON b GOSUB 210,
400,550,350,200,700,800
700 INPUT "Start address"; start$
710 IF start$ = "" THEN start$ = "3000"
720 start = VAL( "&" + start$ )
730 INPUT "Name";name$
740 IF name$ = "" THEN GOTO 730
750 INPUT "Length"; length$
760 IF length$ = "" THEN length$ =
"400"
770 length = VAL( "&" + length$ )
780 SAVE name$,b,start,length
790 RETURN
800 INPUT "Name";name$
810 LOAD name$
820 RETURN
```

Program II

```
4000 REM PROGRAM II
4010 MODE 1:INK 0,0:BORDER 0
4020 LOCATE 10,5:PEN 3:PAPER
1:PRINT "Programmer's Toolkit "
4030 LOCATE 15,10:PAPER 0
4040 PEN 1:PRINT "1. ";:PEN 2:PRINT
   " Hexer."
4050 LOCATE 15,12
4060 PEN 1:PRINT "2. ";:PEN 2:PRINT
" Assembler."
4070 LOCATE 15,16
4080 PEN 3:PRINT "PRESS 1 or 2"
4090 IF INKEY(64)>-1 THEN
   CLS:GOTO 10000
4100 IF INKEY(65)=-1 GOTO 4090
4110 PEN 1
```

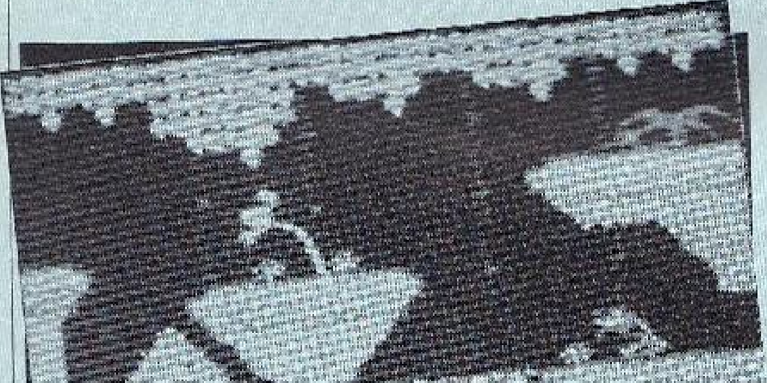
REVIEWS

Livingstone

from Alligata
cassette, disk, joystick and keys

In October 23, 1871 Henry Stanley located Dr Livingstone by the shores of Lake Tanganyika. In 1987 he's missing again, but this time it's up to you to find him.

Dressed in suitable attire you set out into the jungles of darkest Africa. Despite what David Attenborough tells you, the jungle is a horrible place. Every few yards there is some vicious creature eyeing you up as its next meal.



Fortunately you are carrying your explorer's survival kit. This comprises a boomerang, dagger, hand grenade and pole vaulting equipment.

As you play you soon learn which weapon is required for each situation. The boomerang flies horizontally before curving up and over your head — this is very cleverly animated. The dagger has a flat trajectory — useful for long distance accuracy.

The very deep pockets of your safari jacket contain an infinite supply of all weapons. Your reserves of food and drink are indicated by two meters at the bottom of the screen — it is important that you find fresh provisions at regular intervals if you are to stay alive.

Numerous plateaus and cliffs give the game a ladders and levels feel. You will explore jungle, rivers, native villages, temples and far too many damp dark pits. These can be spotted by the different texture of the ground covering them.

Even so, a mistimed leap can easily send you crashing into the darkness below. Ferocious looking eyes blink open at regular intervals - bump into these and you lose one of your eight lives.

Life is made more difficult by the flora and fauna of the jungle. Poisonous lizards, snapping piranhas, coconut-throwing monkeys and a trigger-happy hunter provide the animal attackers. Plant life is represented by a beautifully animated man-eating plant.

The varied arsenal and detailed animation stop *Livingstone* from becoming just another arcade/adventure game.

Nev Astly

Presentation 80%
Well designed, informative screen layout.

Graphics 85%
The man-eating plant is a beauty!

Sound 78%
Catchy title tune and good sound effects.

Playability 86%
The wide range of weapons is a boon.

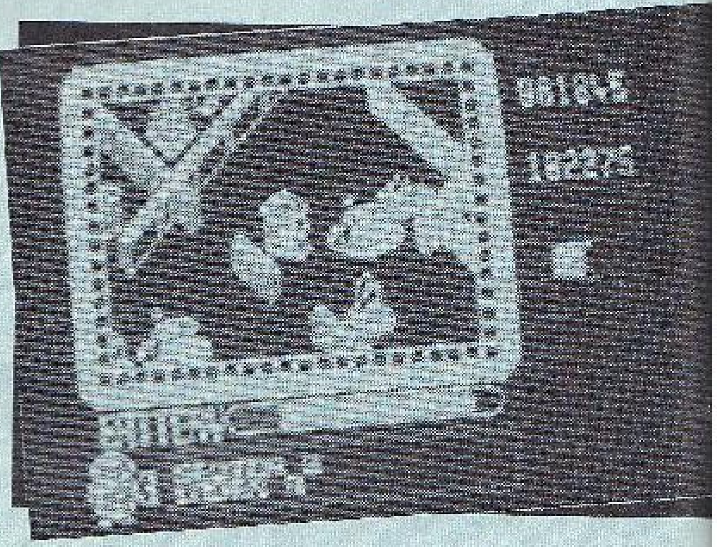
Addictive qualities 83%
Difficult at first, but it becomes a little easier with practice.

Value for money 80%
Will provide many hours of frustration.

Overall 85%
Dig out the safari jacket and sharpen the machete.

Martianoids

from Ultimate cassette, joystick and keys



If anyone was to say **Ultimate** to you, you should instantly think of 3D games such as *Knightlore* and *Gunlaw*. Once again **Ultimate** has given us a 3D game, though *Martianoids* is quite different from anything the company has given us before.

The game takes place on the starship Markon Dawn, which has been sent on a thousand year mission to gather information on new life forms and habitable planets.

The selection process must be perfect, and for this reason the Markons installed the greatest computer ever built, the Brain of Markon.

You play the part of the maintenance droid, who must destroy all the Martianoid droids that enter the Brain, and keep it in full working order.

The Brain is made up of cones, used for the transmission of programs between the data store and the central computing system. These need constant repair or replacement with spares that line the walls.

If all the aliens in a sector are destroyed and all cones are active then the sector becomes safe from further attack, leaving you free to defend elsewhere. However, if you remain too long in one sector then others may be completely destroyed.

To help you defend the Brain you have two types of weapon — a simple laser, which will destroy all Martianoid craft, and a blaster, which can destroy everything, including internal walls and the precious cones.

Your robot loses energy if it touches a Martianoid — several such collisions result in the robot being completely drained of energy and exploding.

The sound was no great surprise either — very much like a Spectrum — which is silly when you consider how much better the Amstrad sound is.

The graphics ranged from very plain to some quite detailed alien robots, including the omni-present clockwork mice. Unless you are a real fanatic of the 3D game then avoid *Martianoids*. And even if you find them addictive, take a look before buying.

Tony Clarke

Presentation 81%

Looks good on both a green screen and colour monitor. A pity that it lacks a redefine key option.

Graphics 65%

Some quite detailed graphics, but most are fairly simple.

Sound 43%

A few simple sounds and a short title tune which were taken straight from the Spectrum version.

Playability 54%

Easy enough to control robot, but the task itself is near impossible.

Addictive qualities 56%

Only for the real 3D game buffs.

Value for money 67%

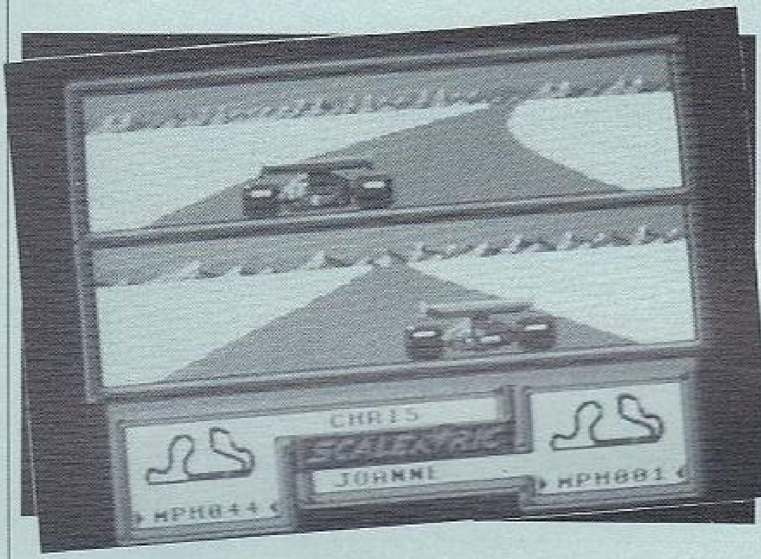
Your money would be better invested in a few budget titles.

Overall 53%

If you have bought all the decent games this month then this may be your next choice. Otherwise, give it a miss.

Scalextric

from Leisure Genius/Virgin Games, cassette, joystick



Continuing the not-too-recent trend of split-level racing games first introduced by *Pitstop 2* from **Epyx**, yet another has just been launched upon the world by **Leisure Genius** in the form of *Scalextric*, a game for one or two players.

Now I have always been of the opinion that racing games should be fast and furious, full of danger and excitement, treating you to laps and laps of fist-clenching action, with only your wits and agile wrists to preserve you and your racer.

So just how well does *Scalextric* conform to my ideal overview of racing life on the screen?

The ingredients all appear to be mixed with loving care: A plan view for each eager driver, a lap timer, cleverly simulated centrifugal force and the ability to skid. It all seems just like the real thing!

Of course you might expect that the tracks will eventually start to become over-familiar. So **Leisure Genius**, true to its name, has given you the option of designing your own track, full of banked curves, straights and chicances.

It is quite possible to idle away the best part of an hour or two constructing the track, carefully positioning the pieces until before your very eyes unfolds the Formula One track of your dreams. Saving it to tape or disc for later races is a piece of cake.

But what of the actual game-play? I can hear the question welling up in the mind of the impatient reader. Having pitted my wits against *Pitstops 1* and *2*, practiced my aggressive steering technique in *Pole Position* on the Atari, been blasted from behind in *Super Cycle* and gone over the top in *Revs* on the BBC Micro, all that remains to be said is that *Scalextric* managed to elicit absolutely no positive response from me at all.

The lack of racing competitors on the track, the poor use of sound and the lack of feel via the joystick, coupled with the unrealistic graphics presentation of the moving road, left me with the impression that no lessons have been learned by the programmers from viewing any of the above superior programs.

Victor Laszlo

Presentation 70%

The track builder is simple to use and the best part of the game. Almost as much fun as a jigsaw!

Graphics 60%

Dull, and uninspiring.

Sound 50%

Engine noise that annoys you beyond belief.

Playability 50%

Not a patch on any other driving game.

Addictive qualities 35%

None that I can accurately discern.

Value for money 50%

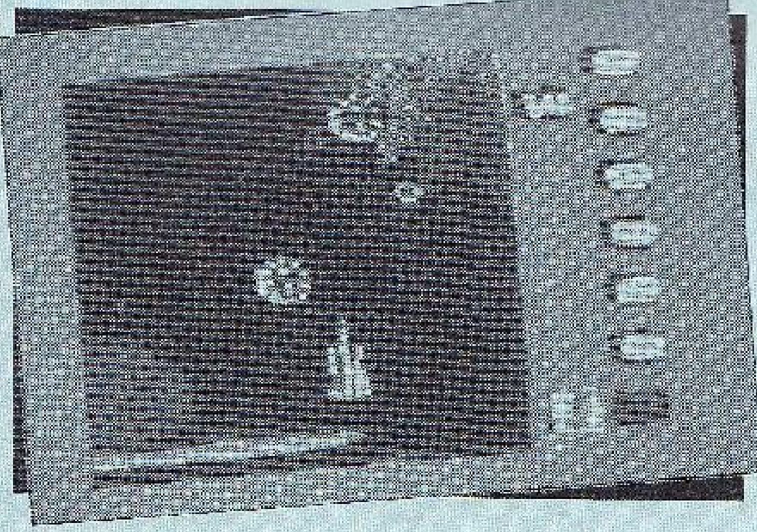
Highly overpriced, should be a budget title.

Overall 60%

Sorry, please try again.

Trap

from Alligata, cassette, joystick and keys



Alligata has had its weak months over the past year, with the release of such games as *Kettle* and the hyped *Meltdown*. *Trap*, however, has broken the run of luck and looks set to be a bestseller on the Amstrad.

The game is set in a time where the deterrents, which the leaders of each nation are so proud of, have finally become the instruments of destruction.

Your mission is to demonstrate to the enemy that the legendary fighter corps of your home world is the best fighting force ever known.

For all the scenario, *Trap* is simply a vertically-scrolling shoot-'em-up, split into three stages. Before you begin you choose one of four types of craft — each has a value in orbs, which are earned as the game progresses.

You start with no orbs, and can therefore only take ship one, which doesn't have any cargo capacity. This is an important part of the game, as dropping a cargo pod on to any human survivors found in the game will give you an extra life.

The first part of the game places you in an asteroid shower, which has to be dodged before you can progress to the next stage, though it is possible to destroy a few asteroids with lasers.

You move on to a scrolling valley filled with rivers and walls. Each wall has only one gap through which you can fly.

Fuel is an important element of the game. A gauge on the far left of the screen indicates how much is left in your ship — run out and your ship will crash into the valley.

The final level has to be completed on foot. Your pilot is placed on the ground in the alien complex. He can then only travel forward, left and right in a search for the orbs.

The programmer has made good use of the Amstrad's colour palette, and the ships stand out well in their shades of grey. Those of you who like good shoot-'em-ups — which the CPC has been lacking for so long — will not be disappointed.

Tony Clarke

Presentation 79%

Good keyboard layout and several ships to chose from when playing. Let down by the lack of a high score table.

Graphics 91%

Well-defined graphics that give a feeling of depth.

Sound 94%

Brilliant soundtrack, if a little piercing on the high notes.

Playability 78%

A little hard to master, but well-defined control layout makes progress better.

Addictive qualities 89%

A great game that will have you coming back for more.

Value for money 89%

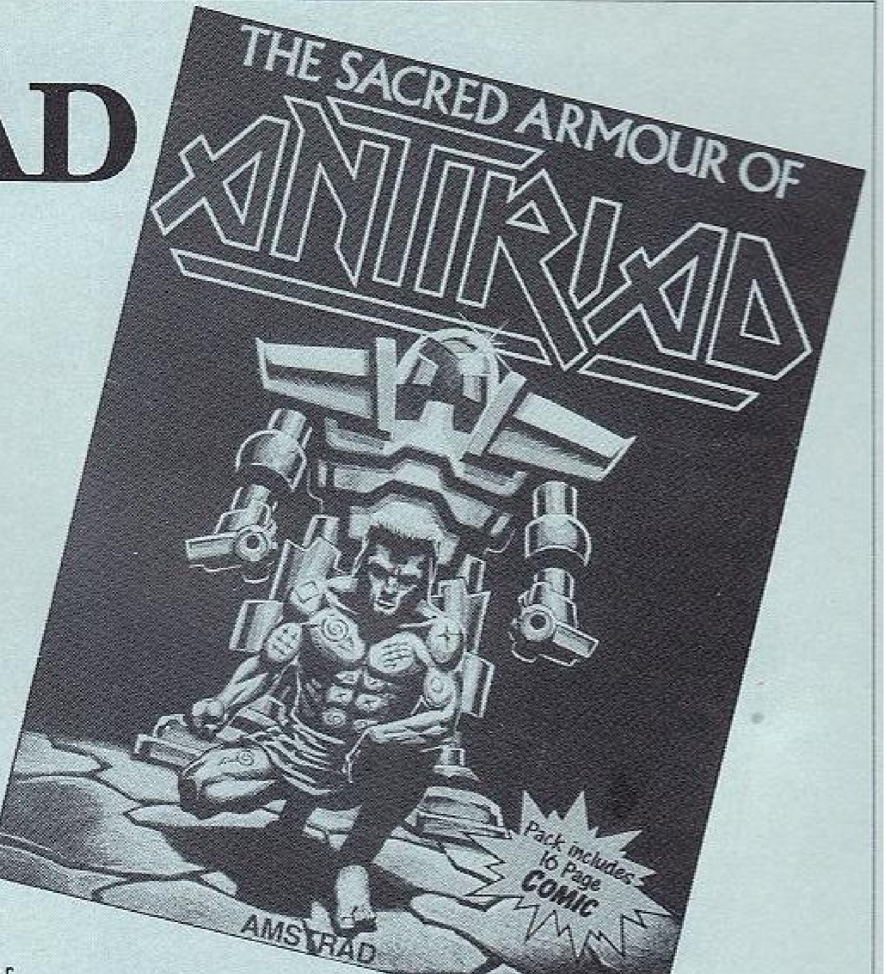
Good.

Overall 87%

A good comeback by Alligata at a perfect mid-range price.

ANTIRIAD

(Palace)



This one is just big, though not quite the largest poke we have ever printed. Just type it in and the game should be no problem to finish. Remember, if you don't want any of the features it provides, simply remove any of the data lines from 330 to 370.

```
100 ' ANTIRIAD pokes tape
110 ' by Cy Booker
120 MEMORY &1FFF
130 FOR ADDR=&5270 TO &527F
140 READ A$:BYTE=VAL("&"+A$)
150 POKE ADDR,BYTE
160 SUM=SUM+ADDR*BYTE
170 NEXT ADDR
180 READ CHECK.SUM
190 IF SUM<>CHECK.SUM THEN PRINT "Err
or in lines 300-310":STOP
200 READ A$: IF A$="-1" THEN 230
210 POKE ADDR,VAL("&"+A$)
220 ADDR=ADDR+1: GOTO 200
230 LOAD "ANTIRIAD"
240 POKE &2015,&70:POKE &2016,&52
250 INPUT "Do you want keyboard contr
ol (y/n)";A$
260 A$=UPPER$(A$)
270 IF A$="Y" THEN GOSUB 1000
280 CALL &2000
290 '
300 DATA 21,80,52,22,A1,00,3E,C3,32,A
0,00,C3,40,00,43,79
310 DATA 28542889
320 '
330 DATA 3E,B7,32,BE,5B : REM INVULNE
RABILITY
340 DATA 3E,B7,32,58,6F : REM NO GRAD
UAL DECREASE OF ENERGY
350 DATA 21,00,00,22,3A,6F : REM RADI
ATION DOESN'T DECREASE
360 DATA AF,32,F4,64     : REM DON'T N
EED TO FIND PULSE BEAM
370 DATA AF,32,49,57     : REM NO NEED
TO FIND GRAVITY BOOTS
380 DATA C3,00,54,-1
390 '
1000 RESTORE 1130
1010 FOR ADDR=&52F0 TO &534A
1020 READ A$:BYTE=VAL("&"+A$)
1030 POKE ADDR,BYTE
1040 SUM=SUM+ADDR*BYTE
1050 NEXT ADDR
1060 READ CHECK.SUM
1070 IF SUM<>CHECK.SUM THEN PRINT "Er
ror in key data!":STOP
1080 POKE &5271,&F0
1090 PRINT "KEYS ARE:- UP, DOWN, LEFT
, RIGHT, FIRE."
1100 PRINT "              S    X     O
   P     SPACE"
1110 RETURN
1120 '
1130 DATA 21,FE,52,22,3D,79,3E,C3,32,
3C,79,C3,80,52,16,00
1140 DATA 0E,43,ED,49,06,F4.ED,78,E6,
08,20,02,16,08,0E,44
1150 DATA 06,F6,ED,49,06,F4,ED,78,E6,
04,20,02,CB,D2,0E,47
1160 DATA 06,F6,ED,49,06,F4,ED,78,CB,
67,20,02,CB,C2,CB,7F
1170 DATA 20,02,CB,CA,0E,45,06,F6,ED,
49,06,F4,ED,78,E6,80
1180 DATA 20,02,CB,E2,A4,43,79,7A,C3,
45,79
1190 DATA 250367123
1200 '
1210 ' THE KEY ROUTINE WAS INCLUDED
'CAUSE MY JOYSTICK DON'T WORK.
1220 '
```

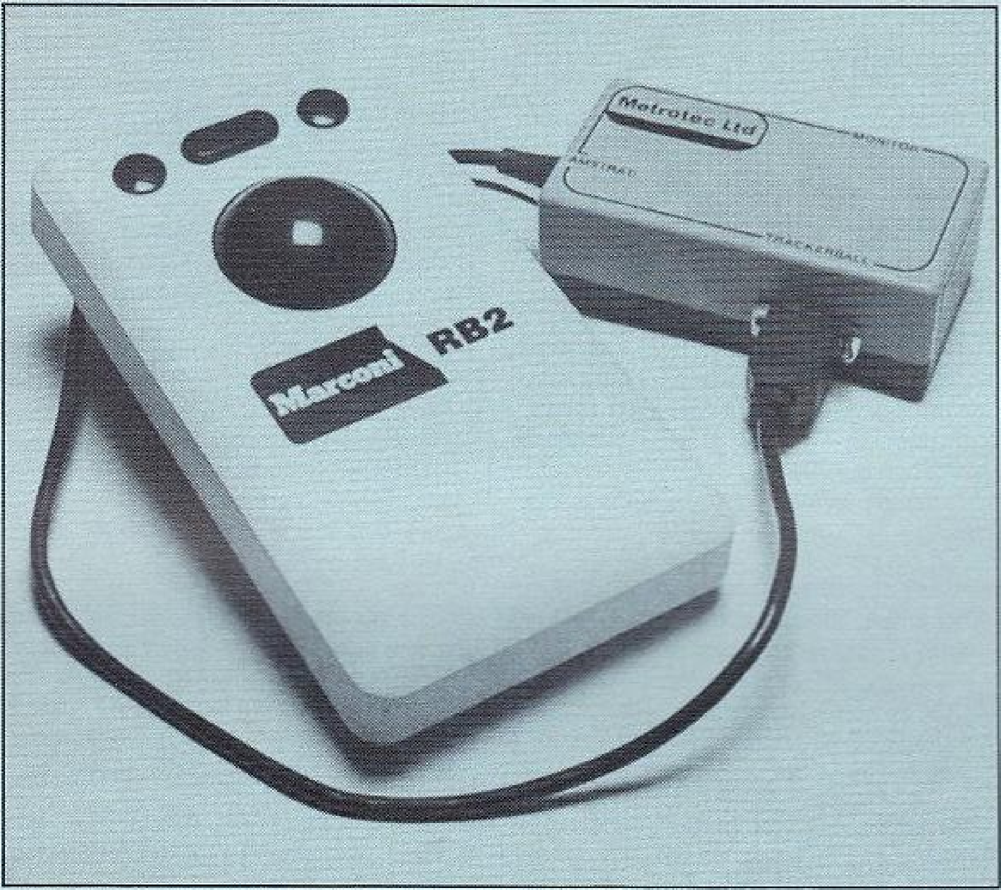
FLY SPY

Just like the Antiriad poke, you can erase the lines you don't need.

(Mastertronic)

```
100 FLY POKES by Cy Booker.
110
120 FOR ADDR=&BE8Ø TO &BEAA
13Ø READ A$: BYTE=VAL(&"-A$)
14Ø POKE ADDR,BYTE:SUM=SUM+ADDR*BYTE
15Ø NEXT ADDR
16Ø READ CHECK.SUM
17Ø IF SUM<>CHECK.SUM THEN PRINT "Err
or in lines 23Ø-26Ø!":STOP
18Ø READ A$: IF A$="-1" THEN 21Ø
19Ø POKE ADDR,VAL("&"+A$)
2ØØ ADDR=ADDR-1: GOTO 18Ø
21Ø CALL &BE8Ø
22Ø '
23Ø DATA Ø6,ØØ,31,AB,CD,11,ØØ,ØF,CD,7
7,BC,F3,21,ØØ,1Ø,11
24Ø DATA 4Ø,ØØ,Ø6,Ø3,3E,BF,D5,AE,77,E
D,AØ,EA,97,BE,21,AB
25Ø DATA BE,22,EE,ØØ,A4,43,79,32,49,Ø
0,C9
26Ø DATA -7418Ø477
27Ø DATA 21,3E,Ø7,22,62,81: REM &Ø7
LIVES.
28Ø DATA 32,3',A2,21,67,CD,22,4Ø,A2,2
1: REM INFINITE LIVES AND PRESS ANY
29Ø DATA 16,9B,22,42,A2,3E,2Ø,32,44,A
2: REM WHEN YOU DIE TO QUIT
3ØØ DATA 3E,3C,32,8Ø,84,AF,32,2Ø,A7 :
REM INFINITE SHIELDS
31Ø DATA 32,18,A6 : REM INFINITE FUEL
32Ø DATA 32,4B,AE : REM KEEP TELEKEY
33Ø DATA 32,74,A5 : REM 25Ø BULLETS
34Ø DATA 3E,18,32,3A,AE : REM NO ENCT
'S ABORTS
35Ø DATA 3E,18,32,2C,AB : REM CAN'T D
IE FROM BOMB
36Ø DATA AF,32,F8,9D,3E,1Ø,32,EØ,9D :
REM WEIGHT OF &1Ø
37Ø DATA 3E,9Ø,21,3E,42,22,1D,82,32,1
F,82 : REM CODES NOT RANDOM
38Ø DATA C3,56,77,-1
39Ø '
4ØØ ' A=ABCDEF:B=GHIJKL:C=MNOPQR:D=STU
VWX
41Ø 'E=YZBCDE:F=FGABCD:G=EFGHIJ:H=KLM
NOP
42Ø 'I=QRSTUV:J=WXYZBC:K=DEFGHI
43Ø '
44Ø 'LINE 36Ø:-&ØØ=TOO LIGHT - ALWAYS
ØOAT
45Ø ' &28 = MAX CAN'T ØY
46Ø '
47Ø ' CHEAT MODE:- A) PAUSE GAME
48Ø '              B) Type in THIS IS
TOO HARD, with spaces. You many have
to redefine the space key so that it
isn't fire
49Ø '              C) A message will
appear on the screen
5ØØ '              D) Hold down the
following keys and press Ø to quit
51Ø '        1= Walk through every
thing
52Ø '        2= Infinite fuel
53Ø '        3= When pick up loops
through 16 objects (some are undefin
ed so loop round
54Ø '        4= Infinite laser shots
55Ø '        5= Infinite lives
55Ø '        6= Infinite shields.
56Ø '        7= ?? I don't know, but
it must do something ??
6ØØ ' Due to the Amstrad's keyboard
hardware, some key combinations are
not available.
```

MARCONI TRACKERBALL



The Marconi RB2 Trackerball and interface

Mice and Wimps (windows, icons, mouse, and pull-down menus) have been all the rage lately. More programs are appearing which cut down on nearly all keyboard operations, using the icon and mouse systems for everything from databases to art packages.

The sudden leap in popularity of the mouse has obscured many devices which were designed to do the same job, and in many cases were more accurate.

The Marconi RB2 trackerball is just such a device. It is around eight inches long, the top half housing a brown ball about the size of a snooker ball, above which are three large buttons.

The lower half of the trackerball is simply a rest for your palm, making the device both easy and comfortable to use. Unlike a joystick, the trackerball cannot be plugged directly into the Amstrad's joystick port, but needs a converter box to translate the ball's analogue movements into the digital form which can be read through the joystick port.

The A/D converter takes its power from the Amstrad's 5v supply which plugs into the converter and a lead from the converter supplies the keyboard with power.

The problem with this arrangement may not be apparent if you only have the trackerball connected to the computer, but if other devices also take their power from the computer you may find that the monitor's power supply overloads.

If this happens it is a simple task to switch off the monitor, remove a few devices and then switch it on again, thus resetting the circuit breaker.

An art package is supplied with the trackerball which uses an icon-driven system to select functions within the program.

You move the ball to direct the cursor to the required function, while the three buttons are programmed to become, from left to right, Execute, Move and Cancel.

The Execute button simply selects a function from the menu, or tells the software to carry out a command that has been set up such as drawing a circle, which requires the centre point and radius to be set first.

The Move button is double the size of the other two, and allows you to move a secondary cursor. This is used to show the centre of a circle or the start of a line.

Finally, the Cancel button will cancel any operation and return you to menu mode.

Turning to Easy-Art Software, you can use the freehand draw option to create pictures directly on the screen as though you were using a pencil.

When the option is selected a secondary menu appears, from which several thicknesses of line can be selected.

Pressing the centre button will select the required thickness and also starts and stops the "ink" from appearing on the screen each time the trackerball is moved.

Computer pictures are distinguished by their harshness, but there are ways to make a picture look more natural.

The spray can icon selects an airbrush function which sprays paint as you move the trackerball. The size of the airbrush effect can also be varied. This is excellent for producing smoke or cloud.

Solid lines and shapes can be created by positioning the start and ends of the lines which make up the shape.

When two points have been chosen for the triangle function a rubber-banding system is used to help with the selection of the third point. "Elastic" lines are quickly drawn and erased from the two points selected to the current position of the cursor, so you can see how the triangle will look before it is finally drawn. With rectangles and squares you only need to define the top left and bottom right corners to draw the shape.

Again, the rubber-banding technique is used, though this time it comes into action the moment the first point is set.

When precise drawing is required, such as in building plans, it is almost impossible to work with pixel accuracy on a large screen, so a Zoom option has been included.

This allows direct editing of individual areas which are blown up to the full size of the screen. In this mode pixels can only be turned on or off, and none of the other editing functions are available.

The eraser included is more selective than those in many art packages. Usually an eraser marker wipes out everything underneath it, but this one erases only pixels of a certain colour.

If you wish to erase everything under the cursor then this system becomes a bind as the area must be wiped over several times, but this drawback is more than made up for by the abi to erase selectively.

Painting in very large or complicated areas of the scre using the freehand mode can be a long process, so a fill opti is included. When selected the cursor must be moved into t area you wish to fill.

But beware — if the area is not completely bounded ink m spill into the rest of the screen. The only way to stop this is to u the zoom function and check the area from which the ink leak and plug the gap.

A textured fill option is available which can fill an area with predefined pattern. Several patterns are already in memory b it is possible to define your own.

The printer option is configured to use any Epson-compatib machine, such as the DMP 2000/3000, but cannot be change to use any other.

Different textures are used to mimic screen colours. Tw levels of printing are available — low density for checking how picture is going to look eventually and a high density/quali option for the finished job.

The low density option is twice as fast as the high density on and also uses much less printer ribbon.

The textures used to mimic the colours can be selected fro the Change shades option, each shade representing one of th colours in the Amstrad's colour palette.

Finally an icon designer allows you to create new icons Although they are not available to Basic, they can be loaded int the art package and placed on the screen and even rotated.

The best use of icons in this manner is for technical drawings or circuit diagrams, where predefined shapes save time when creating plans or circuits.

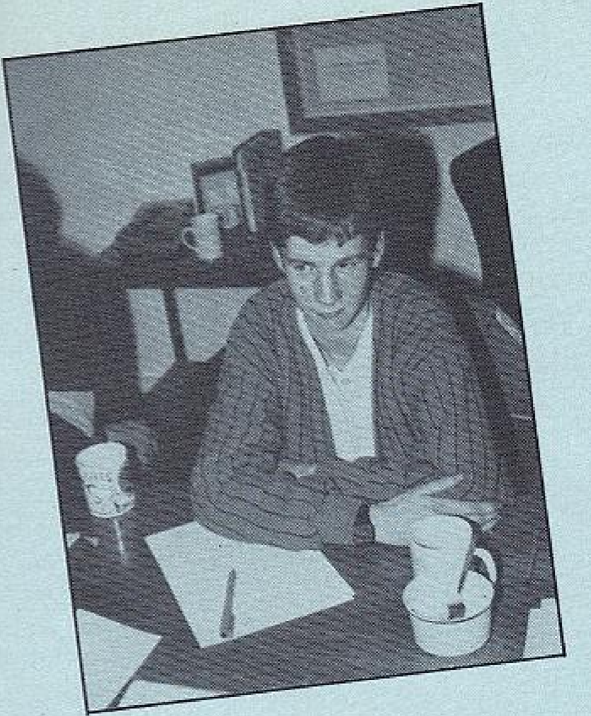
Easy-Art is one of the best art utilities available at th moment, and includes many more powerful features which make subtle changes to the basic functions and make the whole package a joy to use.

The ability to use other input devices, such as a joystick, AMX mouse or the keyboard, make the package available to all users, and not just those who have the trackerball.

The trackerball unit itself is well built and designed — it took quite a beating from some junior school children who tried it and seemed none the worse for the experience.

The large buttons are a little difficult to operate. This is not a design fault, simply the nature of control method employed when using a device of this type.

The trackerball and the Easy-Art package are top products.



Game designer Martin Lee, age17

JOHN MINSON sees it all happen!

The Birth of a Game

The only sound in the Domark boardroom was the whirring and clicking of high IQs in action. This was a think tank of the greatest depth.

Representing the coding lobby were Graham Stafford and David Llewellyn, ace programming brains behind Design Design, tangling with the knotty problems of sprite sizes and animation cycles.

Meanwhile Richard Naylor, Domark's software manager, was determined to produce a high standard game that would also be commercially successful — and that would arrive on time.

Acting as adjudicators and creative advisers were Dave Carlos and his cohort from the InterMediates public relations company, Graham Knappett.

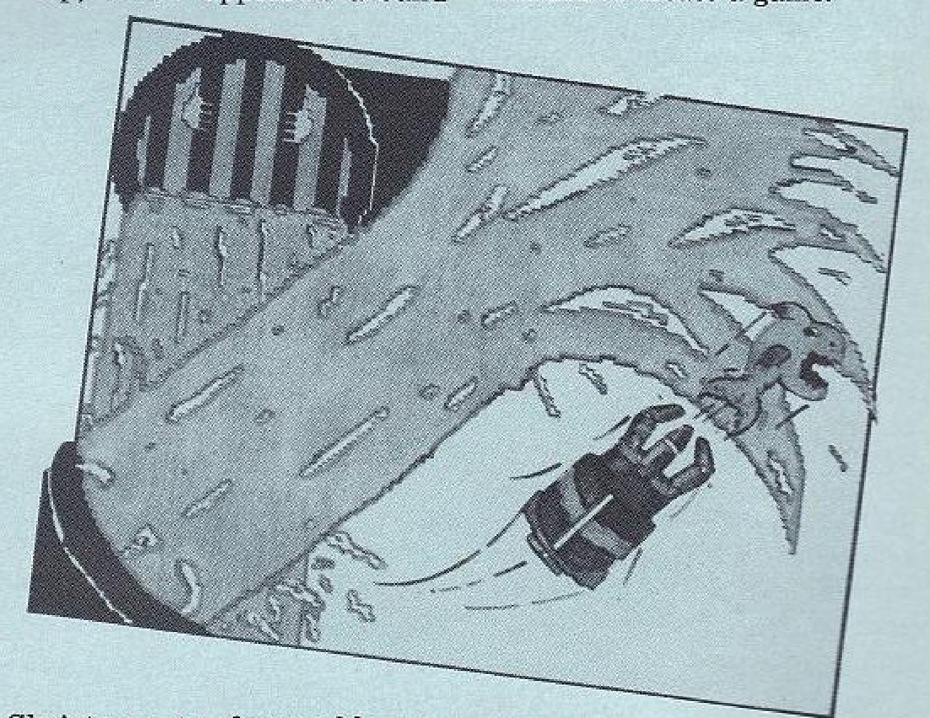
But the most important individual in the room by far was a 17-year-old called Martin Lee. We probably all daydream about sitting in on a brainstorming session for a new game. But how many of us dare to dream that one day we'll be surrounded by a high-powered team discussing our own design?

The event that had brought about this wonderful turn of events was a competition run in a computer magazine.

The search for new games, nick-named Genesis, had already produced one title, Kat Trap, which appeared around Christmas to favourable reviews. Now it was the turn of Martin's prizewinner, The Sewer.

Kat Trap had been developed as a Spectrum game but Graham and David of Design Design are programming on the Amstrad this time because it's easier to convert to other formats from a CPC. Not that it should worry Martin too much — he's a Commodore owner!

It all goes to prove that you don't need to be machine-minded to create a game.

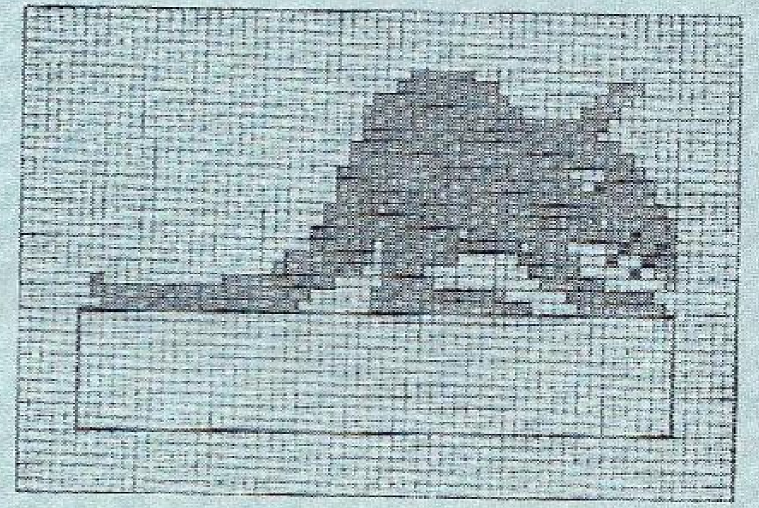


Martin had designed board games before, but never anything for the computer. In fact, he confessed that he's not a programmer at all. But with Design Design to take care of the code, all he needed to do was come up with a good idea, which he had. The Genesis judges had been most impressed with his plans, and they included that veteran of imaginative programs, Mel Croucher (ex-Automation). Some compliment!

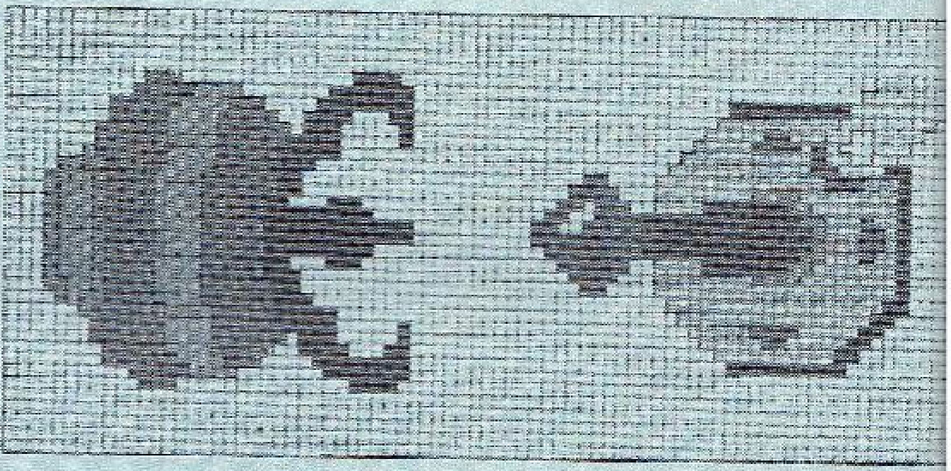
Martin's specialities are music and art, which he's studying at college, and this really showed in his presentation. It's a model of clarity, with enough detail to show that he knows what he has in mind, and yet it's also open-ended enough so that if the programmers say that a thing can't be done, it won't destroy the game's structure.

The proposal had taken a month of hard work to prepare, and contained 22 colourful pages, beautifully drawn on graph paper to indicate it could be transferred to pixels. It covers everything from the general concept to specimen problems that players could face. Martin had been afraid that it was actually too detailed and enclosed a special plea for the judges to take the time to read it. They did and were sucked into The Sewer. The action of the arcade strategy game is set in the crumbling clay pipes beneath the city, where a collection of foul creatures roam free. Unhappily for the unsuspecting workmen, sent down by a cynical mayor with an eye to the forthcoming election, the place is an invitation to industrial injury.

Soon saving the workmen had become even more important than repairing the pipes. As if wandering wildlife isn't bad enough, there are broken tubes spewing out acid, crumbling rocks and all manner of other problems. With only an old but



Martin's original design not only indicated what role the rat would play in the game, he also suggested how it could move. Using some nifty sprite manipulation, such as rotation, Graham reckoned he could keep it to a four-frame animation cycle.



When you're designing a game you have to bear in mind how much control you can have over your screen object. By indicating how the sprite of the Manipulator, on the left, relates to the joystick, on the right, Martin fully exploits the possibilities.

reliable Service Manipulator robot to save the men, the player's got his work cut out.

It's said that there are only a limited number of plots in the world and The Sewer bears an initial similarity to Boulderdash, which Martin admits is a personal favourite on his C64. But, as Dave Carlos pointed out, you could do worse than be compared with a classic like that.

When you start to design your game, it's probably as well not to worry too much about technical considerations. Naturally you wouldn't waste your time on a CPC program that required Amiga-style hardware, but there's no reason to limit yourself to what you think can be done.

The programmers will soon tell you if it can't, and there's nothing the hardened hacker likes more than trying to do the impossible.

Sorting out the technical details was a major part of that afternoon's activity. The two hours turned out to be quite a voyage of discovery, as Martin's guided tour of The Sewer kept uncovering new aspects of the design that at one stage had Graham Stafford rubbing his chin and exclaiming, "Damned devious!"

The first thing to establish was how large each level could be. Obviously the computer's memory plays a critical part in this decision, and so does the way that the plans for each level are stored. Martin had suggested at least 16 levels, four screens by four screens in size.

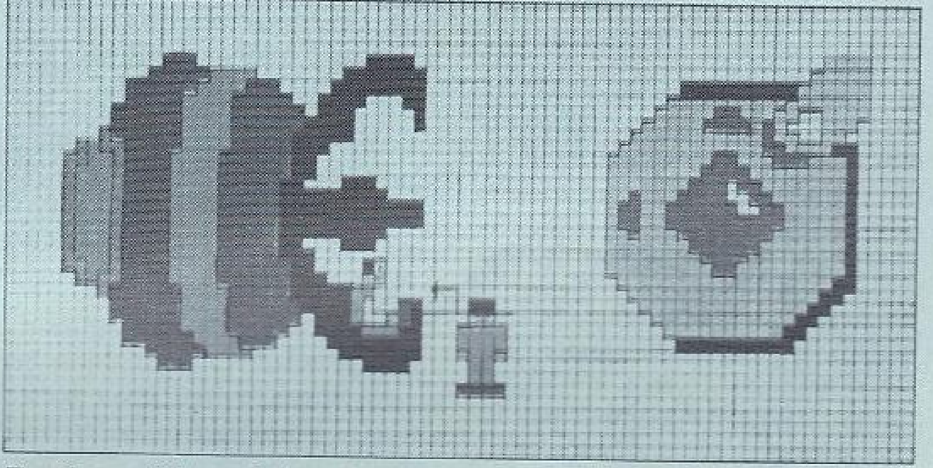
The best way to hold the map in memory would be to use a byte to indicate whether an area was water, acid or a type of rock, in terms of x-y coordinates, which would allow quick reference, even if it wasn't the most memory-efficient method. Graham wanted to combine it with a scrolling window technique, opening on to a three by three level.

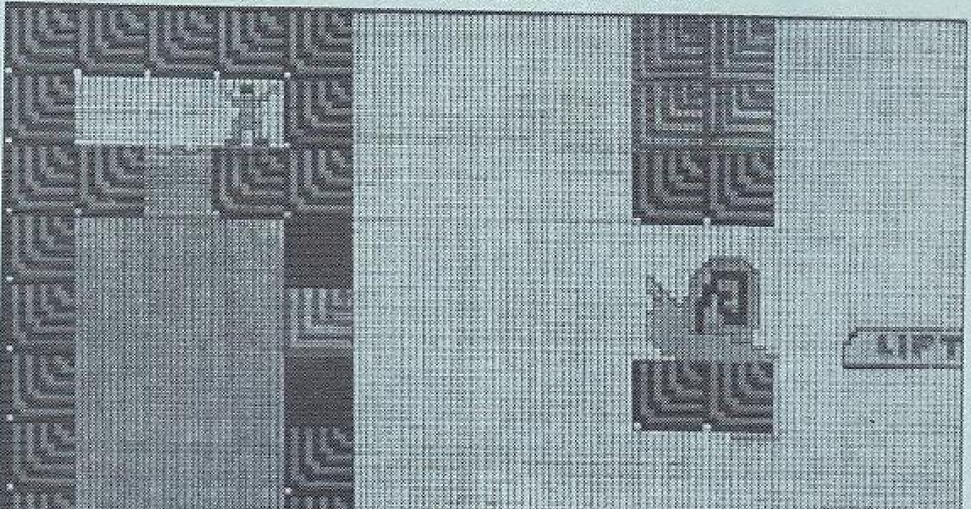
Most of the action could then be confined to the visible screen area allowing for smooth four-way movement between screens. Also, because water rises, it makes sense to have the exit at the top of a level.

The sewer system features a variety of rocks. Permanent rock stays where it is, collapsible rock can be picked up to plug pipes or to squash the nasties, and there's also soft rock, which the acid eats away. Escapes can only be made by exploiting these characteristics and changing the landscape.

Graham suggested that there should be a fourth type of mineral though, so that he could squeeze every last bit out of the bytes available. Martin immediately suggested permeable rock, which lets water through but can support a man. But that left the problem of indicating rock and water to the micro. This was one instance where programming problems outweighed a good idea.

The characters and creatures presented similar hazards.


Further stick and fire button control allows you to rescue the workmen, by taking them gently into the Manipulator's jaws. Here Martin allowed himself a little poetic license with his pixel-styled drawings ... he gave the workmen eyes and mouths.



One of Martin's sample problems with the snail that provides a sticky solution and saves the worker from a sticky end. The various types of rock are colour coded, as is the tank of acid, which will destroy the lift if you just break through the wall. Martin's explanation of a solution had us all shaking our heads that we'd not seen it before.

both the electric and conger varieties, which were rejected because they resembled the leeches. Alligators were also a short-lived suggestion when Graham's expression changed to horror. "They're big!" he screamed, "at least eight bytes long!"

While this sprite was causing problems, everyone liked the rat. It would only take four frames of animination, with a fifth when it jumped. Once again the discussion had become a trade-off of the ideal design, contained in the plan, against what's practical.

But it wasn't all Design Design saying, "Can't do that." Graham didn't see much use for the snail with its sticky trail. Very patiently Martin referred us to one of the specimen puzzles at the end of his proposal and, sure enough, the seemingly innocuous snail played an important role in freeing a man.

One of Martin's most ingenious ideas is the way you handle the Manipulator. It has to be able to pick up men and objects and either drop them or throw them at enemies. But Martin realised that players prefer complete joystick control, so he'd used the fire button with the stick to obtain different effects.

Martin also wanted a second screen, which could be called up to display control icons and details of your status. But Graham went for a more steamlined approach with consoles scattered around the sewers, which the player could access with the Manipulator.

Nobody could work out quite what they'd be doing there, but it was a much more integrated approach so we all decided to ignore that illogicality.

The main feature of the original plan to be discarded was electricity. There were going to be wires, which could be switched off, causing them to disappear, but Graham was worried that they just added an unnecessary complication.

A compromise was reached and the cables were scrapped, but if they became necessary for gameplay they can be revived easily at a later stage.

After all this I wondered if Martin would feel that his brilliant creation had been torn to bits but he was quite cheerful about all the changes. "It's much better," he said philosophically.

Which, I suppose, was the lesson of the day. Martin's original idea was already changed in some respects, and other alterations are likely to arise as the DesDes team get down to business, discovering what will and what won't work.

But those two hours round the table had turned the theory of a great idea into something that will work in practice. It was Martin's inspiration and Design Design's experience, coupled with suggestions from the other assorted brains.

I'm looking forward to my next meeting with Martin, Graham and the rest, because by now coding is well underway. But I feel like I too contributed to the polishing of those 22 pages into a potential masterpiece. I want to see how this baby grows up.

We will be coming back to the Sewer in a few months to see how the game is progressing.

Will Design Design be able to stick closely to Martins original format?

Will Martin be happy with the alterations?

Only time will tell.

Joyous sounds

Ian Sharpe tries out DK'tronics' PCW Joystick/ Sound Controller

Already well known for its CPC peripherals, Dk'tronics has branched out with the latest addition to their range — a combined joystick and sound controller for the PCW which also offers simple input/output to external circuits.

A car radio type speaker is connected to the interface by means of a jack plug and a rotary knob provides volume adjustment. There is a standard 9-pin D-type joystick socket as used on many other computers and a through connector at the back allows for the connection of further peripherals.

The unit houses an AY-3-8912 sound chip which is the same one fitted to the CPCs. However, there are important differences in the way the chip is used on the two machines.

Firstly, the CPCs were designed with the sound chip as an integral part of the hardware and give it full firmware support. This extends the chip's capabilities by implementing software tone and amplitude envelopes.

These are driven by interrupts and allow you to create envelopes that are more complec and flexible than those built into the chip — the hardware envelopes.

Secondly, the PCW and Mallard Basic were not designed with any sound in mind other than a beep. To use the unit from Basic you must write directly to the chip's registers using the OUT command rather than a SOUND statement as used in CPC Basic. The firmware on a CPC offers a user-friendly interface which makes full use of the possibilities offered by the chip.

This means that the Basic programmer will not be able to get as much out of the chip as his counterpart working on a CPC. To do this would take some difficult machine code programming.

The information contained in the 16 page manual is pitched at a fairly advanced level and leaves something to be desired for the beginner. The majority of PCW owners bought their machine as an alternative to an electric typewriter and only after using it for a while decide to investigate the potential of Basic.

Anybody used to bits and bytes and familiar with machine code will find the information complete and easy to digest, but relatively inexperienced Basic users may have problems.

On page 4, after advice on installing the interface, we are straight into bits, chip registers and register selection. There is a need for a couple of pages to provide a gentle introduction to an initially off-putting subject.

The sound quality produced is quite good, thanks mainly to the speaker which is larger and better than its CPC equivalent.

Of course commercial programs will not use the sound capability unless they have been written specifically to use the interface. This will not be the case with programs currently available, but let's hope software houses take note and incorporate this possibility in their software.

The joystick controller is easier to understand and in comparison to the joystick port of a CPC has a very useful addition — it can be written to as well as read from.

This means that you can interface your PCW with external circuitry through the joystick port, for example a relay controlling a mains device. The manual details the functions of the various pins as well as giving examples of circuits with simple control programs.

Another nice feature of the interface is that it allows the use of auto-fire if your joystick has it.

Demonstration programs of sound and joystick control are supplied on disc, as shown in the menu in Figure 1, together with a utility, DERJOY.COM.

This will make the joystick emulate any of the keyboard keys so you can use it with your favourite game as long as it is booted from within CP/M.

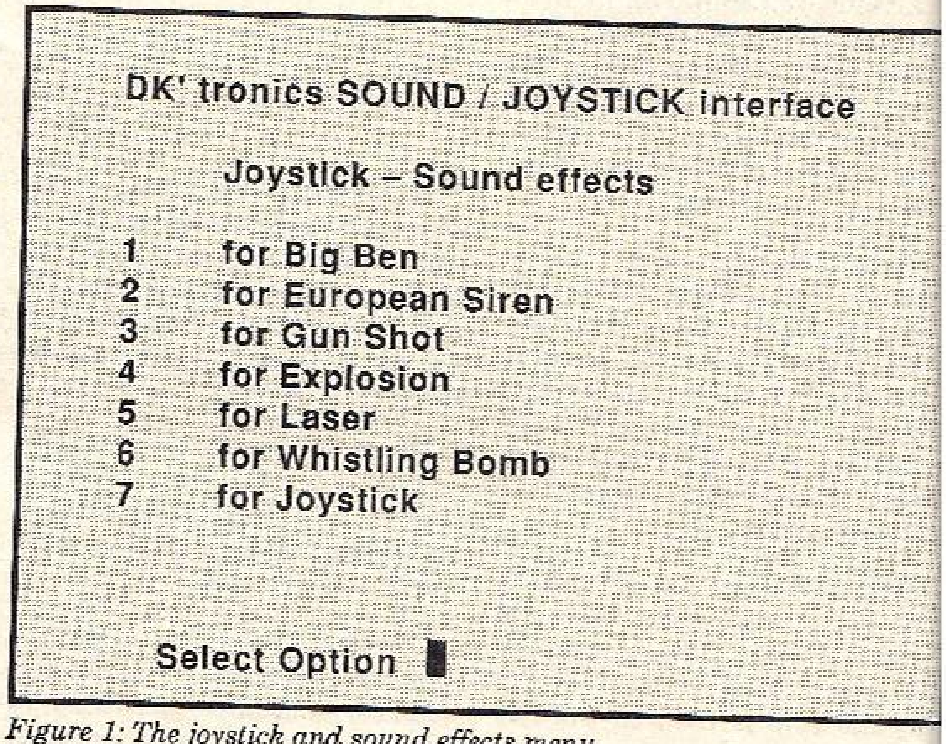
This interface is more likely to appeal to the experienced programmer and the electronics enthusiast, at least until some software appears to utilise its musical potential.

If you aren't interested in the sound or external circuitry aspects Dk'tronics is selling a cheaper unit with only the joystick control facility.

A word of caution to owner the Amstrad serial interface. W I plugged ours into the back of sound unit I blew a couple of com nents in the PCW's power supp

We think that this may be du the conductive tracks on through connector being rat broad and the serial interface d not have a locating key. It is p sible to insert it very slightly off thus possibly causing a short cuit.

Anybody buying this or a other peripheral would be w advised to fit a small plactic loc ing key into the through connect of the serial interface. It's easy to — you simply push it in. Perha Dk'tronics would consider inclu ing one with any future versions the interface.



```
DK' tronics SOUND / JOYSTICK interface

        Joystick - Sound effects

   1      for Big Ben
   2      for European Siren
   3      for Gun Shot
   4      for Explosion
   5      for Laser
   6      for Whistling Bomb
   7      for Joystick

     Select Option █
```

Figure 1: The joystick and sound effects menu

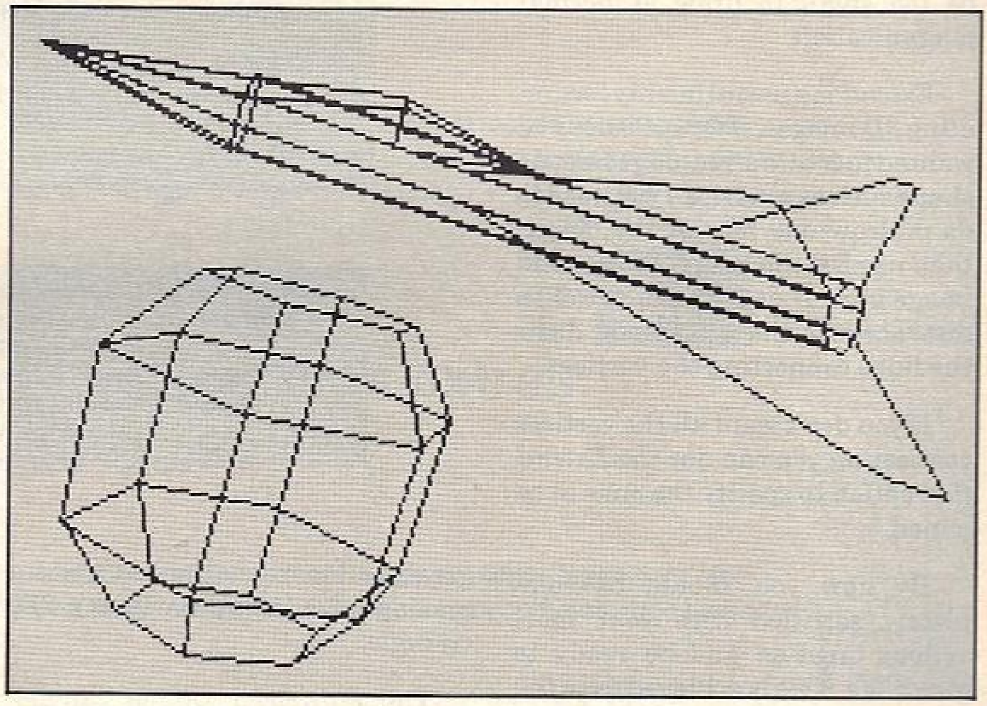
Easy entry into the world of 3D graphics

Jon Revis reviews Model Universe

Model Universe is a three dimensional drawing system which enables you to create wire frame models, rotate them, and either dump them to a printer or save them for later use in your own programs.

Similar packages on the market rely on entering reams of X,Y and Z coordinates to achieve the finished model. Thankfully this is not the case with Model Universe as all your designs are created on-screen using either joystick, mouse or the keyboard. For the purposes of this review I opted for joystick, but using the keyboard is just as easy.

Model Universe is menu driven, very easy to operate and I was using all the package's facilities with complete confidence in under an hour — the figures give you a complete breakdown of the menu system with its various options.


Samples from the demonstration program

The program comes complete with 10 demonstration files that provide ideal raw material for your future experiments.

When using the program your first port of call will probably be the *Design menu*. Having selected *Design mode* from this, you are presented with a three-line information window at the bottom of the screen and a pointer near the centre.

The pointer's X and Y coordinates change as it is moved around the screen, while holding down the fire button and moving the joystick forwards or backwards changes its Z coordinate.

The only other controls you need to remember are three function keys which draw and erase the lines. So long as you keep an eye on your Z coordinate you should have no difficulty at all in creating some quite complex models.

The remainder of the options on the Design menu enable you to modify and manipulate your masterpiece. *Viewpoint* allows you to view it from above, below, left, or right and is useful for spotting incorrect Z coordinates.

Once you have created an object you may move it off to one side of the screen using the *Position* facility, generating space for a second model on the same screen.

Enlarge will increase or reduce the size of your model. A typical use for this function would be to mag-

Save file	Save data file to tape/disc.
Load file	Load data file from tape/disc.
Catalog	Catalog tape/disc.
Background	Change background colour.
Foreground	Change foreground colour.
Border	Change border colour.
Destroy file	Erase data file currently in memory.

Figure I: Main menu

nify an object, add some small detail to the drawing then reduce it back to normal size. Doing this you can create fine detail which would be too small to draw at normal magnification.

Extend is a useful facility for creating 3D images. If for instance you wanted to create a simple geometric shape such as a cube, all you need to do is draw one face of the object, then use Extend to create a second image of your design at a specifies distance into the screen and draw the lines connecting the vertices.

If you don't want all the vertices connecting you can use the *Extend flag* option to specify the ones to be joined.

Design mode	Create a new design or modify an existing one.
Viewpoint	View object from left, right, above or below.
Position	Move finished object to a new position on screen.
Enlarge	Expand or reduce size of object.
Circle	Draw a circle.
Extend	Extend a 2D object into the screen.
Extend flag	Extend selected lines only.

Figure II: Design menu

Display mode	Enter display mode.
Parameters	Vary size of steps through which object is rotated.
Reset	Reset all parameters to default values.
Output	Send screen display to tape/disc, printer, or a Basic variable file.

Figure III: Display menu

By now you should be completely satisfied with your marvellous creation and be ready to give it life. This is achieved from the *Display menu*.

Here, selecting *Display mode* allows you to rotate your model about various axes, and one very clever facility allows you to alter your viewing position along the Z axis.

Pressing Shift and the down cursor moves you into the screen, and when experimenting with the "House" demo design you can actually move inside the building and rotate it around you.

The speed at which the object rotates is determined by the complexity of the design — there's a great deal more maths involved in moving a 3D jet fighter than a 2D triangle.

The *Parameter* option can be used to help compensate for any lack of speed by increasing the angle through which the object is rotates each time it moves. The default step size is 6 degrees but this can be increased to a maximum of 90.

By the time you've finished playing with your design it will probably be sitting on its end in the top corner of the screen. It's at moments like this that the *Reset* facility will be invaluable, returning all parameters to their default values and placing your design at its original position on screen.

Finally the program will look a
all lines currently on the screen an
produce a file of the X and Y coord
nates necessary to recreate tha
screen using a Basic program.
routine is also supplied that wi
use the contents of that file to creat
a dump on a printer/plotter such a
the Tandy CGP–115.

The one major flaw in this oth
erwise very impressive package i
the absence of a routine to manipu
late the objects from within you
own Basic programs.

Model Universe is an easy-to-use
yet very professional piece o
software — an ideal introduction to
the fascinating world of 3D
graphics.

BUSINESS CONTENTS

Authors have a good word for the PCW

by Mike Gerrard

THE AMSTRAD PCW was advertised as the machine that makes you throw your typewriter out of the window, and with a few hundred thousand of them sold a great many people must be doing that, figuratively if not literally.

What difference is it making, though, to the people who must process more words than any others, professional writers?

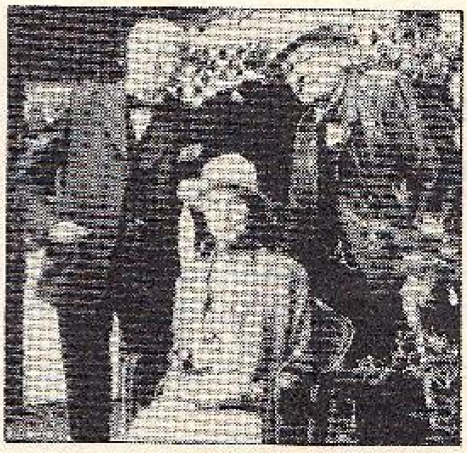
After a recent survey, the writers' union, the Writers' Guild of Great Britain, produced a list of members making up their Word Processor User Group. Although this was limited to those members who were prepared to share their skills with others, and obviously excluded members who felt they didn't have the time or knowledge to deal with panic phone calls, it nevertheless gives an indication of the inroads the PCW had made into the world of the professional author.

Of the 78 members listed, exactly one third of them, 26, used the Amstrad PCW, the nearest rivals being the BBC Micro, used by 11 members of the Guild, and the Apricot, by 10 people. The PCW obviously leads the field quite easily.

One writer not on the Guild's list, but glad of its existence, is Peter Buckman, who was responsible for adapting the highly-acclaimed BBC TV series *All Passion Spent* from the novel by Vita Sackville-West, and who has also adapted several other books for radio: *Pendennis, The Napoleon of Notting Hill* and *Tristram Shandy*, among others.

As a writer of film scripts, books and stage plays as well, he obviously needed a machine that could bope with all types of writing, though his first experiences with the Amstrad were none too encouraging.

"I bought one about last September", he says, "and far from wanting to throw my typewriter out of the window, after a few weeks with that wretched manual I was ready to throw my PCW out of the window instead.


"All Passion Spent" . . . adapted on a PCW by Peter Buckman

"I was getting absolutely n
where with it, but fortunately th
Writers' Guild was able to put me i
touch with a member who kne
something about the machines an
he was able to explain lots of thing
to me so that with a day's tutoria
from him and a lot of help from th
very good and patient compute
dealer here in Chipping Norton,
began to come to terms with it.

"I suppose it took me two t
three months to be confiden
enough to be able to smile when
sat down to work in the mornings.
did find it very difficult to adjust, i
many ways. I found it both chal
lenging and worrying to have ye
another excuse for not writing.

"But I now find that I can worl
directly on to it much more easily.
recently finished the first episode o
a radio adaptation of *Parade's Enc*
by Ford Madox Ford, and I workec
direct from the book on to th
screen.

"The slowness of the printer anc
of LocoScript are the two majo
drawbacks. I recently printed ou
an 80 page script, and in near lette
quality using continuous station
ery it took me eight hours.

"It's also quite slow doing littl
amendments, getting to the middle
of a particular paragraph to remove
a sentence, that kind of thing
where the cursor seems to disap
pear for hours at a time. Mind you

it's still much quicker than using a pen and retyping."

Peter Buckman says that having the word processor doesn't seem yet to have increased his output, a feeling shared by other writers working on lengthy material, though for journalists and others producing shorter pieces that perhaps don't need quite as much polishing it certainly does have a great effect on the amount of work that can be done.

"I'm very pleased with it, though", Peter Buckman says, "and go round recommending it to lost of people who were unsure about getting one. My final verdict would be that it's wonderful, but don't use it for the first time if you're up against a deadline or you'll end up in the loony bin.

"I do like the mechanics of it, too. There's a little high you get from the machine looking so efficient, and making you feel efficient too. I like the silence. I hated the humming sound of my electric typewriter impatiently waiting for me to write something. I can cope with the ever-blinking green eye looking at you, I don't mind that at all. I mean, that's just like being married."

That ever-blinking cursor has proved a problem for another writer who has had an Amstrad for about the same length of time as Peter Buckman, but is less enthusiastic about the machine.

Richard O'Keefe has written many episodes of *Crown Court* and *The Practice*, as well as being the writer responsible for *Rockcliffe's Babies*, just coming to the end of its first run on BBC1 and already booked for a second series.

"When I'm writing well", he says, "I tend to write slowly but steadily and take a lot ot time and thought over things. But the cursor sits there blinking away and I find I have to call up a menu or something like that on the screen just to get rid of the cursor for a while, and then I cancel it later when I want to carry on writing. If you're neurotic that kind of thing could drive you mad.

"I think I've sorted out the initial problems I had with the machine, and I've mastered it, in as much as I can do the things that I want to do, though there are obviously lots of other things that I haven't discovered yet.

"I still feel ambivalent about it though, and keep wandering back to the typewriter. One thing about the typewriter is that you can ingore how it works, it's quite simple and most people can grasp it, but there's so much ingenuity gone into the Amstrad that you feel it's got a life of its own and I don't want that.

Terry James. . . "I'm glad I got it"

"It's as if there's a conversation going on with the machine the whole time, and the PCW tends to answer back quite a lot, whereas I'd prefer to feel that I was in charge.

"But I would still have many positive things to say about it. It's extremely good for letters, juggling things around, it's just that for me with anything longer I like everything to be quiet and placid.

"In my first initial ecstasy over the machine I thought 'That's it, the typewriter's going to disappear', but I don't think I'll ever switch completely. Part of that is definitely the combination of me and the machine, though, rather than the machine in itself.

"I've had terrible trouble with it, which no one can really understand. I'm on my third machine now, and that doesn't help at all.

"The dealers have been very helpful in replacing machines and trying to trace what's been happening, but no one really knows, and I definitely think there's something odd going on between me and the machine."

Another Writers' Guild member who is still finding the place that the PCW has in his working life is Terry James, who has written several stage plays as well as having been a regular scriptwriter on radio series such as *The Archers* and *Waggoners Walk*.

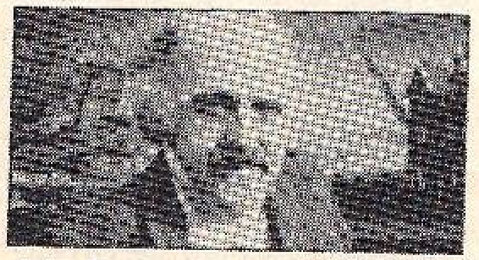
More recently he has written a lot of drama for BBC Schools Radio, and a feature film he wrote, *Zina*, was due to be broadcast on Channel 4 in February after a cinema release last year.

"I suppose I've had the machine for about a year or so now, and I'm glad I got it. It took me about a month to get to grips with it, and everthing was fine but then I went and lost a whole afternoon's work the day before a delivery deadline and I wasn't exactly delighted about that.

"I had to work through the night to write everything again and have it ready for the next day. What had happened was that I'd filled a disc with about four scripts, and ran out of space. The disc full message came up, and whereas I now know how to deal with that, this was the first time I'd encountered it and I pressed the wrong button and lost the lot.

"I hadn't realised that you actually need twice as much space as you think you do in order to save a file, but now I make sure I have masses of space on every disc I'm working on anyway.

"I went through a very heavy alienation period with the machine, when I was working on a stage play and I wasn't really happy about it, so I switched back to the typewriter for a while and that helped.


John Finch . . . "I really got interested in computers"

"Now I find I'm progressively writing direct on to the screen, and I've now used it to write five radio plays for schools.

"It's absolutely fantastic for letters, for small thing, but on longer things it's very slow. I'm still using LocoScript because I haven't really got time to learn something new.

"But then, I've also got this wonderful thing called a pen, and the type of pen I like writes really fast over paper, and in a quiet moment I feel I can write faster than I can type on the screen. It's no problem switching between pen and paper, typewriter and computer.

"I don't think the Amstrad's the greatest thing since sliced bread, but it has its place."

For some writers then, the Amstrad has to take its place as another tool of the trade, while for others it becomes the sole means of writing.

Writer John Finch, author of those two popular TV series *Sam* and *A Family at War*, explained that he recently tried to use a typewriter again for the first time since buying his PCW almost a year ago, and found that he couldn't.

Buying the Amstrad in the first place also helped him through a bad case of writers block.

"I've just finished a 10-part series for the BBC called *Joss*, which is a family in the 1980s and how their lives were shaped by the 1960s.

"But part way through writing the series I dried. My normal practice when that happens is to go out and buy a new typewriter, and that often works. This time, though , I'd seen the ads for the Amstrad and my wife suggested I get one of those. A firm locally hired them out, so I thought I'd try one for a week and see what I made of it.

"After about four to five days I was getting nowhere and was ready to give up on it. I thought I must be too old for this kind of technology.

"Then in the last couple of days everything seemed to fall into place and at the end of the week I went out and bought one. I finished the series on it, and honestly think I wouldn't have done it if it wasn't for the Amstrad. If I'd had that machine while I was doing *Sam* and *A Family at War*, life would have been so much easier.

"I've also really got interested in computers, it's become quite a hobby for me as well. I bought the John Hughes book and found that very helpful, but I bought all the books that came out about the machine, and I also get *Computing with the Amstrad* every month.

"I've also got three different word processors now, though only use two of them, Tasword for writing letters on and LocoScript for everything else. I like Tasword because you can use different typefaces with it, and that's good fun for doing correspondence, but prefer LocoScript for scripts, even though it is so slow.

"I've found databases incredibly useful as well. I use Sagesofts Magic Filer for outlining a TV series. The kind of lengthy thing that I do is really like writing a novel on television, and there are lots of characters and events to keep track of.

"You sometimes find that halfway through you forget the name of a character from an earlier episode, or perhaps the name of someone's father or how old he was, and so keeping everything in a database enables me to check things and cross-reference everything that I need.

"I find using the Amstrad makes me think harder about what I write for some reason — it forces me to think. I honestly feel that I'm writing better because of it".

PlanIt, family book-keeping plus extras

Reviewed by Jo Stork

No sooner had I finished writing my reviews of two personal organisers, and pointed out all the many deficiencies which the whole concept contains, than I was sent another: PlanIt.

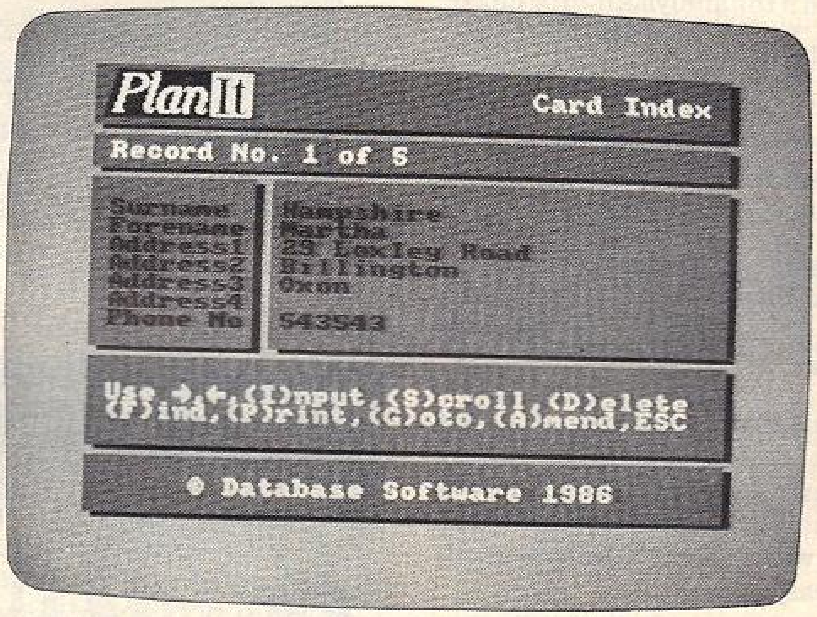
It is produced by Database Software, sister company to the publishers of this magazine. Reviewing it here is an ideal opportunity for me to underline my independence.

PlanIt for the CPC or PCW machines consists of three main modules. In a sense it can be regarded as a family book-keeping system or financial planner with additional features.

The first and arguably most attractive module is the personal accounts system which allows you to keep very precise track of where all your money is going to or coming from.

If this could have been used in conjunction with the third module — the financial diary — it would have been far better.

It keeps track of all your appointments. And since it has a field for recording money it can also be used as a forward log of all the standing orders and regular payments which need to be made on particular days — for instance your mortgage, car or washing machine repayments.



```
PlanIt                              Card Index
Record No. 1 of 5
Surname    Hampshire
Forename   Martha
Address1   29 Loxley Road
Address2   Billington
Address3   Oxon
Address4
Phone No   543543

Use →,←,(I)nput,(S)croll,(D)elete
(F)ind,(P)rint,(G)oto,(A)mend,ESC

© Database Software 1986
```

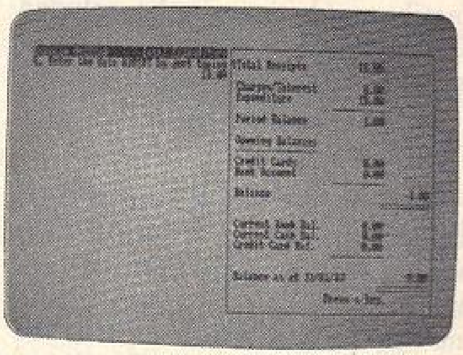
The card index system

This does make it rather different from, and potentially better than, the other diaries I have met. As time progresses the financial diary is also used to archive all the historical information.

The second module is a card index which may be used to create a very simple database or to produce labels.

These modules work very well indeed and within an hour or so even novice users should be familiar with their operations.

Careful thought as to how much use PlanIt can be put to in your household is necessary.



The personal accounts system

The more time I spent working with PlanIt, the more I came to appreciate its many merits. However, I also wondered how many households have the discipline to enter, on a very regular basis and for year after year, the considerable quantities of data that the system requires.

And if the analysis is to be meaningful, I can see people wandering round with their pockets stuffed with receipts until they input the value — unless the heading with the most entries is to be Miscellaneous.

If you have this discipline then look no further: If not, give PlanIt a miss since it will be a nine day wonder.

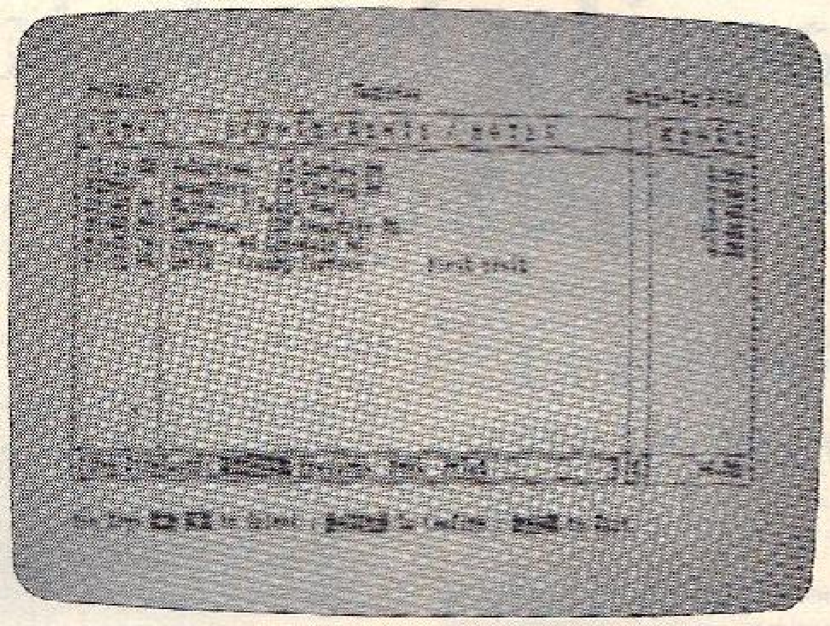
This is not a criticism of PlanIt — but I do doubt if most families are prepared to spend the time running the family finances like a business, even when this is relatively quick and easy.

Curiously this drawback is also PlanIt's greatest potential strength.

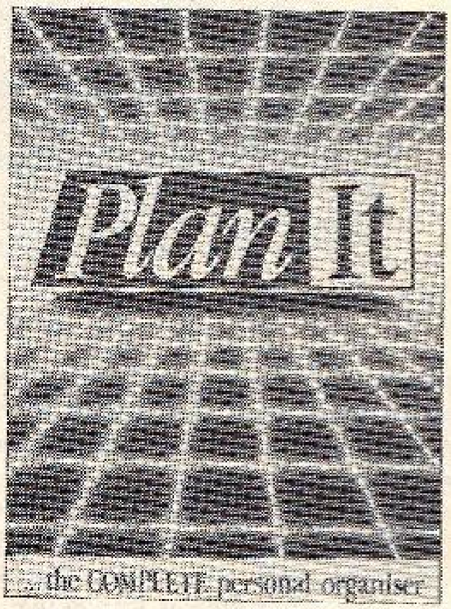
Because it provides so much information and can handle many more transactions than the average family could generate, I concluded that it would be of most benefit to someone needing to keep the books for a very small, certainly non-VAT rated, cash-based business.

Examples which spring to mind in my own town are the postman whose wife is a children's swimming instructor or the Toyota manager whose wife is a part-time chiropodist.

With this concept of PlanIt's scope now clear, I can turn to the accounting features in more detail.



The financial planner



The first stage is to set up the 20 headings under which you can analyse expenditure. Many people will find that only a handful will need changing from those preset by the software.

The next step is to enter the bank account details and finally you input the credit card start balances.

With these three simple actions completed you can start recording where your money is going. Entry is very quick and easy, using menus and minimal input of text or figures.

Anyone familiar with Mini Office II will immediately recognise the house style, although I was rather annoyed to discover that all six figures on a cheque needed to be entered.

In the record shown above I have purposely left the numbers incorrect in order to demonstrate the effect of entering too few — please, Database, put in a couple of extra lines of code to ensure this quirk is removed.

Another addition could be a facility to amend an incorrect entry, although I can see the merit of leaving it on the file and placing a contra-entry by it.

Finally, a facility to cater for standing orders automatically would be nice, to avoid having to make the same entries every month.

But despite these gripes the system works well and should pay for itself in a short while just by enabling you to avoid unnecessary interest payments.

Turning to the reports that are produced, I give these an unqualified "first class". Any of the individual headings can be analysed or a cumulative summary may be produced. The latter is particularly useful since it takes into account all your credit cards.

The card index can be used in one of two ways. PlanIt comes set up initially as a name, address and telephone number index. However, you can set it up for any other purpose which can be held in seven short lines and keep as many files as you wish.

The chiropodist for instance could hold the names and addresses of all her clients in one file, while her husband holds brief details of his Great Western Railway memorabilia in another.

Sorting or searching is simple and quick, and for those familiar with Mini Office II it again uses a similar style of marking the selected records and laying out the reports.

A nice feature is the ability to jump to any record number in the file without needing to scroll through what could, on a PCW8512, be a file with 2000 records on it.

It does only print one label across, but those of you whose stationery is two or more across merely need to set the first character offset to a value which will cause the print to be in the second column.

In short, while the Card Index does not offer much, what it does it does very well indeed — so much so that in many respects I regard it as the strongest and potentially the most useable of the three units.

It is the financial diary which I have the most difficulty in justifying. I do not withdraw a single word I wrote last month about placing diaries on the computer.

Two hundred live entries is probably more than the average family needs just to keep track of their daughter's ballet concerts, angling club dinners and the like.

But anyone who makes 10 appointments in any day is going to run out of space in only a few weeks before archiving is needed, particularly if the family has a heavy social calendar and several standing order commitments.

Add to this the fact that while the money column definitely has its uses, you can record money due and set the value to zero when it is paid, its lack of integration with the book-keeping module reduces its practicality very significantly.

On the positive side, you can manipulate the data in the diary in many different ways. Repeated events, such as an evening class, can be entered once and copied as many times and as far into the future as required.

You can mark and list selected items from the diary or dump the whole to screen or printer. And naturally entries can be changed or deleted.

Granted the whole looks far neater than my own pocket book with its scrawls, doodles and plentiful crossings out, but I couldn't use PlanIt's diary to replace it.

To complete PlanIt there is a separate Loan Calculator, which will certainly tell you how much you will be paying for a product and how much to the money lenders.

In conclusion, my opinion of PlanIt is very mixed. Fundamentally it is sound, inexpensive, comes with a comprehensive and well-written manual and despite a certain delay in updating large files, appears to be efficiently coded.

Nevertheless I believe it needs a little extra development to make it a genuinely excellent product with a fairly broad appeal. At the price it would then be every bit as good as Mini Office II.

Sadly, as it stands now, while offering no glaring weaknesses and many positive virtues, I suspect that the average user will only require one or at most two of its elements.

This being the case, I suggest that anyone needing to log finances would do better to purchase a specialist cashbook program, while those needing a first database use the excellent one in Mini Office II, which allows more fields as well as arithmetic on them, plus a blizzard of other facilities.

I will certainly recommend PlanIt, but only in those circumstances where you can use all its modules, maintain the necessary discipline and not be handicapped by its simplicity.

Not hostile, but do treat with respect...

Gabriel Jacobs
reviews the Sage Database

Data management systems generally involve a trade-off — the greater the range of facilities, the more rigid and unfriendly the rules for structuring the data.

From this point of view the Sage Database is a middle of the order package, not as hostile as full-blown relational databases, but more powerful and therefore less forgiving that simple electronic card index systems.

With the Sage package the most demanding operation is that of setting up a new file, since this involves a degree of planning at odds with the way most people's minds work.

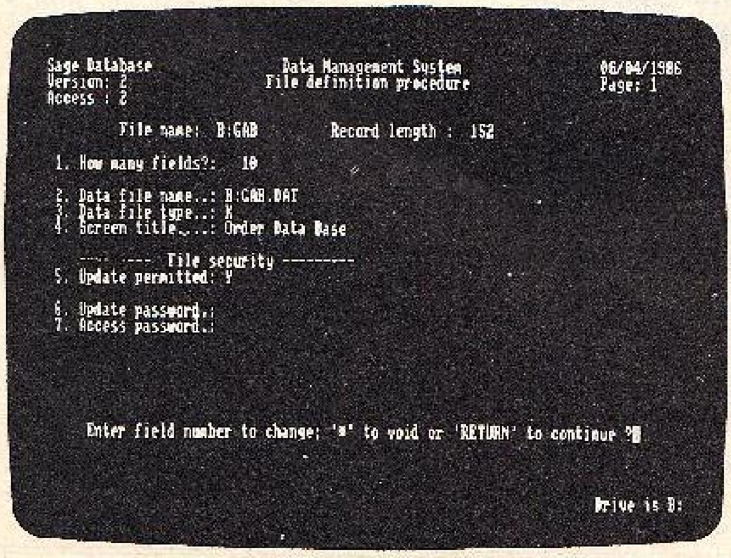
The user has to declare in advance the number of fields per record their lengths and types, output width, left or right justification, decimal precision, conversion to upper case and so on. Apart from the purely cosmetic operation of altering screen display coordinates, changing a field paramenter in an existing file involves restructuring the entire database.

On the other hand, with a measure of forethought and almost certainly some frustrating trial and error it is possible to set up precisely tailored data-entry and report-generation procedures.

Files and individual fields can be permanently or temporarily protected with a mixture of locks, passwords and un-echoed keystrokes, and there is total flexibility in field selection for report purposes. There is also a wide range of field types, including compacted numeric and date fields, time fields and fields on which various kinds of calculation can be carried out, with column totals and basic statistical analyses thrown in for good measure.

Data entry in selected fields can be restricted to numeric to textual ranges, such as allowing only numbers between 1 and 50, or only YES or NO. Forced entry can be specified, obliging the operator to enter information before moving to the next field.

The built-in text editor and mailshot utility are surprisingly powerful. In fact the set of page layout commands doesn't fall far short of that of many dedicated packages, and the range of options includes such things as run time keyboard input and viewing a merged letter on screen before printing. Labels can be printed using the special utility provided,



```
Sage Database                Data Management System              06/04/1986
Version: 2                   File definition procedure           Page: 1
Access : 2

        File name:  B:GAB          Record length :  152

 1. How many fields?:   10

 2. Data file name..: B:GAB.DAT
 3. Data file type..: K
 4. Screen title....: Order Data Base

    ---- ---- File security ---------
 5. Update permitted: Y

 6. Update password.:
 7. Access password.:



    Enter field number to change; '*' to void or 'RETURN' to continue ?█


                                                                 Drive is B:
```

File definition screen

though unfortunately it limits you to a three-across format.

Of course the provision of a wide range of merge-print features inevitably adds to the complexity of using the program. So agian, despite Sage's praiseworthy attempt to sugar the pill, inexperienced users will probably have to learn through bitter experience.

However for those who find creating a new file or producing personalised letters a daunting prospect, interrogating a database and manipulating the information it contains is about as gentle an exercise as it could be.

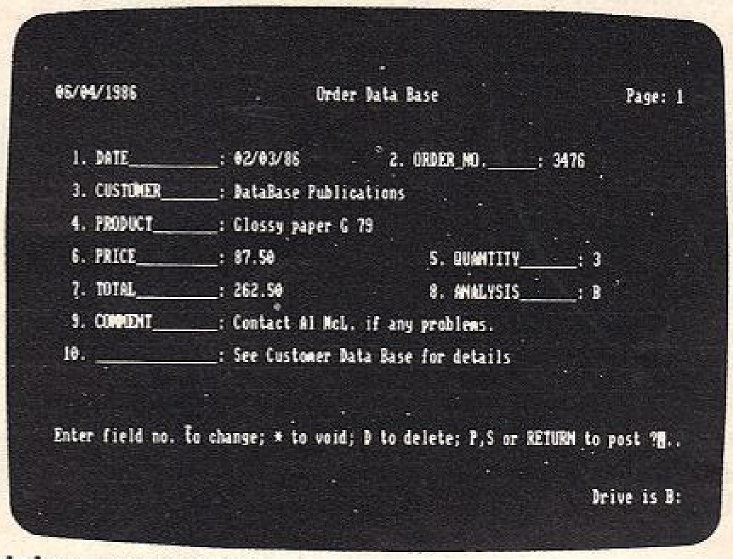
In what Sage calls the Enquiry Processor, the normal computer-style database query language has been replaced by a welcome English-like syntax. This has been achieved by a judicious choice of system words and by adding a sprinkling of disposables — these are words which may be included for the sake of intelligibility but which the program will ignore. The following command, for example, would be syntactically valid:

Please list all employees with a surname of Jones and a salary greater than 6,000 *showing me the age and the department, Thank you.*

Using field numbers instead of names, symbols instead of words and discarding disposables, this could be reduced to:

list employees with 1="*Jones*" *and* 5>6000 *show* 8 9

The Enquiry Processor is also used for sorting records, either relatively slowly if in-string searches or searches with relational operators have to be carried out, or rapidly if a key field consisting of only a compacted record number has been included in the initial file definition.



```
06/04/1986                 Order Data Base                    Page: 1

  1. DATE_________: 02/03/86          2. ORDER_NO._____: 3476
  3. CUSTOMER______: DataBase Publications
  4. PRODUCT_______: Glossy paper G 79
  6. PRICE_________: 87.50                   5. QUANTITY______: 3
  7. TOTAL_________: 262.50                  8. ANALYSIS______: B
  9. COMMENT_______: Contact Al McL. if any problems.
 10. ______________: See Customer Data Base for details


Enter field no. to change; * to void; D to delete; P,S or RETURN to post ?█..

                                                          Drive is B:
```

Typical data entry screen

Such indexed fields are a common feature of databases, and even some of the simpler ones allow a number of fields to be indexed. Surprisingly the Sage Dagabase allows only one — the key field — which could prove something of a nuisance if your records are mostly textual, such as in a bibliography, or if you constantly require numeric sorts. Furthermore search criteria are restricted to two fields at a time and the program will only sort into ascending order.

Again however these limitations have to be set against the plain English implementations of the search and sort commands. In many circumstances speed and range may be secondary considerations when it comes to being able to give a clear instruction such as:

Sort products by category with price greater than 20 *or weight less than* 8 *showing total number of ptr*

Much of Sage's reputation for quality Amstrad applications software has been built on the fact that their products have been well implemented on the PCW and designed as far as possible to run on the entry-level system. I was therefore surprised to find that the database has not been configured with the care we have come to expect.

For instance, the Delete key does not work properly in the Enquiry Processor — Alt + H has to be used instead — and the program is not completely bomb-proof. On one occasion I managed to crash back to CP/M, having permanently corrupted some data by trying to write to a disk previously used for PCW Locoscript files, on a single-drive PCW or CPC6128, the user must make alarmingly frequent changes between system and data disks. Sage admittedly recommends that the program should be run on a twin drive machine, but surely better use could have been made of, for example, the PCW's RAM disk.

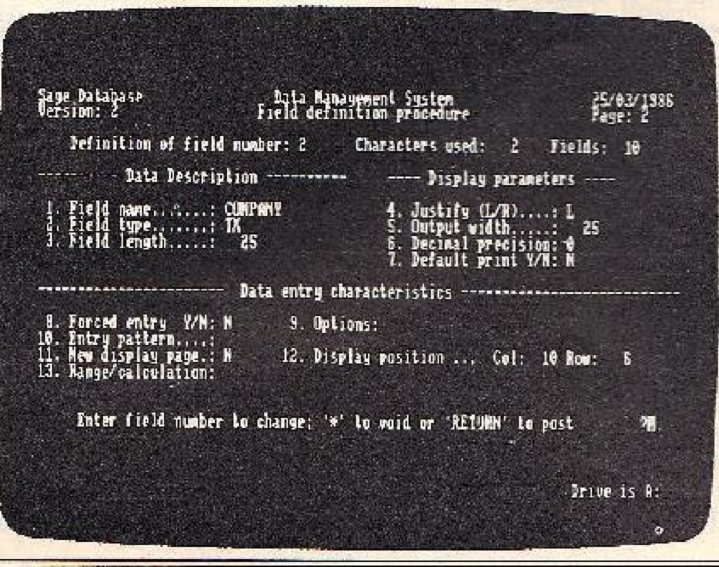
SAGE DATABASE CHECKLIST

Basic features	1
No. of files open	Disc capacity
File size limit	Unknown
Max. recordsper file	1,024 characters
Max. record length	255
Max. fields per record	Text, Interger, Single-precision,
Field types	Extended-precision, Date, Key, Date-stamp, Time-stamp, Blank, Heading.
	255 characters
Max. field length	Yes. Can be locked
Record addition/ deletion	Yes. Can be disabled
Record update/edit	No
Record duplication	Yes
calculations	Comprehensive
File security	
Searching and sorting	
No. of search/sort	2
fields	20
No. of criteria	All 6
Logical operators	
AND/OR/WITH/BY	All 4
modifiers	Yes
Wild cards	Yes, with=and<>
In-string search	Yes
Ascending sort	No
Descending sort	No
Case-independence	
Hard Copy	Yes
Select fields	No
Change field order	No
Re-position fields	Full facilites
Merge-print	Yes
Text editor	Three-across only

The documentation is of a high standard, with an excellent tutorial to get you started.

However errors in the instructions for making a working program disk are hardly likely to inspire first-time users with confidence as some of the descriptions of the more advanced functions, contained only in appendices, are equally likely to baffle them.

Yet despite these criticisms — most of which are ultimately relatively minor — the Sage Database is worthy of serious consideration if you have a twin drive machine, and have not yet invested in a data storage and retrieval system. It is not an outstanding bargain, but it does offer significantly more than many of its rivals in the same price bracket.



```
Sage Database              Data Management System              25/03/1986
Version: 2                 Field definition procedure          Page: 2

   Definition of field number: 2     Characters used:   2   Fields:  10

------ Data Description ----------    ---- Display parameters ----

 1. Field name........: COMPANY        4. Justify (L/R)....: L
 2. Field type........: TX             5. Output width.....:   25
 3. Field length.....:   25            6. Decimal precision: 0
                                       7. Default print Y/N: N

--------------------- Data entry characteristics -------------------------

 8. Forced entry  Y/N: N        9. Options:
10. Entry pattern....:
11. New display page.: N      12. Display position ...  Col:  10 Row:   6
13. Range/calculation:

    Enter field number to change; '*' to void or 'RETURN' to post       ?█.

                                                          Drive is A:
```

Field definition screen

BCPL: Designed for humans, not computers

Roland Waddilove reviews an easy to use systems programming language

The latest addition to Arnor's superb range of software is BCPL, a well established high level systems programming language. It is available for all Amstrad models except the CP1512 and is supplied in a variety of formats.

The package consists of a rom, disc and manual. CPC owners with some form of rom expansion board will probably use the rom version. This is quick, convenient, easy to use and is always instantly available.

For those without rom boards there is an Amsdos version on the 3in disc . There isn't a tape version, which bars those CPC464 owners who haven't upgraded to disc from using it.

On the reverse side of the disc is a version of BCPL which runs under CP/M—both 2.2 and CP/M Plus and this is equally happy running on either a CPC or PCW.

Although not as well known as Basic, BCPL is quite a popular language and was a precursor to C, the language currently in vogue in professional programming circles.

It has been used on a variety of computers and its flexibility makes it well suited to a wide range of applications. These include word processors, databases, languages and operating systems. In fact Arnor says that its Protext word processor is partly written in BCPL.

It is a compiled language, producing fairly compact code which runs at quite a rate of knots. This is perhaps the main reason for its suitability for these sort of applications.

Arnor's BCPL compiler produces pure Z80 stand alone code that may be run on any micro regardless of whether BCPL is present or not. The object code contains all the runtime routines required

Of course the object code produced under CP/M Plus on the PCW can only be run on the PCW. Similarly the Amsdos version only runs under Amsdos, though the source code is quite portable.

BCPL is well structured and many of its features can be seen in some of the popular languages currently in use, especially C. There are procedures and functions with both local and global variables, indirection operators, FOR, IF, WHILE, REPEAT, UNTIL and CASE to name but a few.

BCPL source code is quite readable but may look rather stange at first sight. Newcomers should find the language fairly easy to learn since it is deliberately kept simple.

BCPL is adapted to different environments and applications by the provision of a variety of procedures and functions.

Arnor's BCPL is a full implementation and libraries of standard input/output functions are provided on the disc. Additionally there are machine specific procedures and functions to access the CPC Amstrads' advanced graphics and sound capabilities.

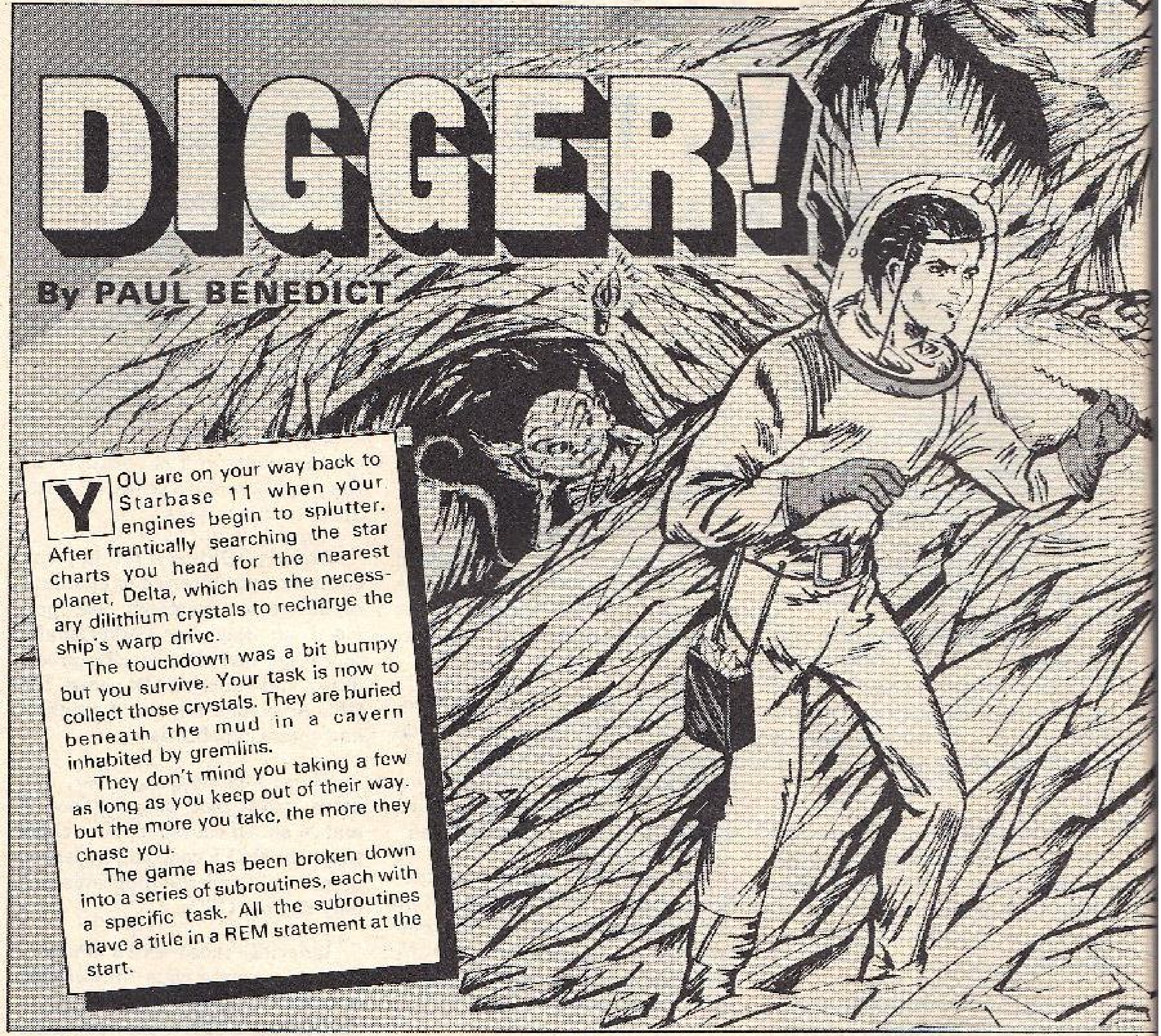
These include pen, paper, ink, move, draw, plot, ent, env, sound and so on. Of course if you include these the BCPL code will no longer be portable and the programs will only run on CPCs.

Ignoring these extra features produces source code that can be compiled and run on any micro supporting BCPL with very little change. So it's quite feasible to develop an application for a PCW or BBC Micro, PC or whatever on a CPC and vice versa.

In addition to the libraries, several example BCPL programs are to be found on the disc, some being quite long and complex. They are provided as source code enabling you to load, list and modify them to your own requirements.

Although the package is not aimed at the beginner, these files,

Turn to page 65



DIGGER!

By PAUL BENEDICT

YOU are on your way back to Starbase 11 when your engines begin to splutter. After frantically searching the star charts you head for the nearest planet, Delta, which has the necessary dilithium crystals to recharge the ship's warp drive.

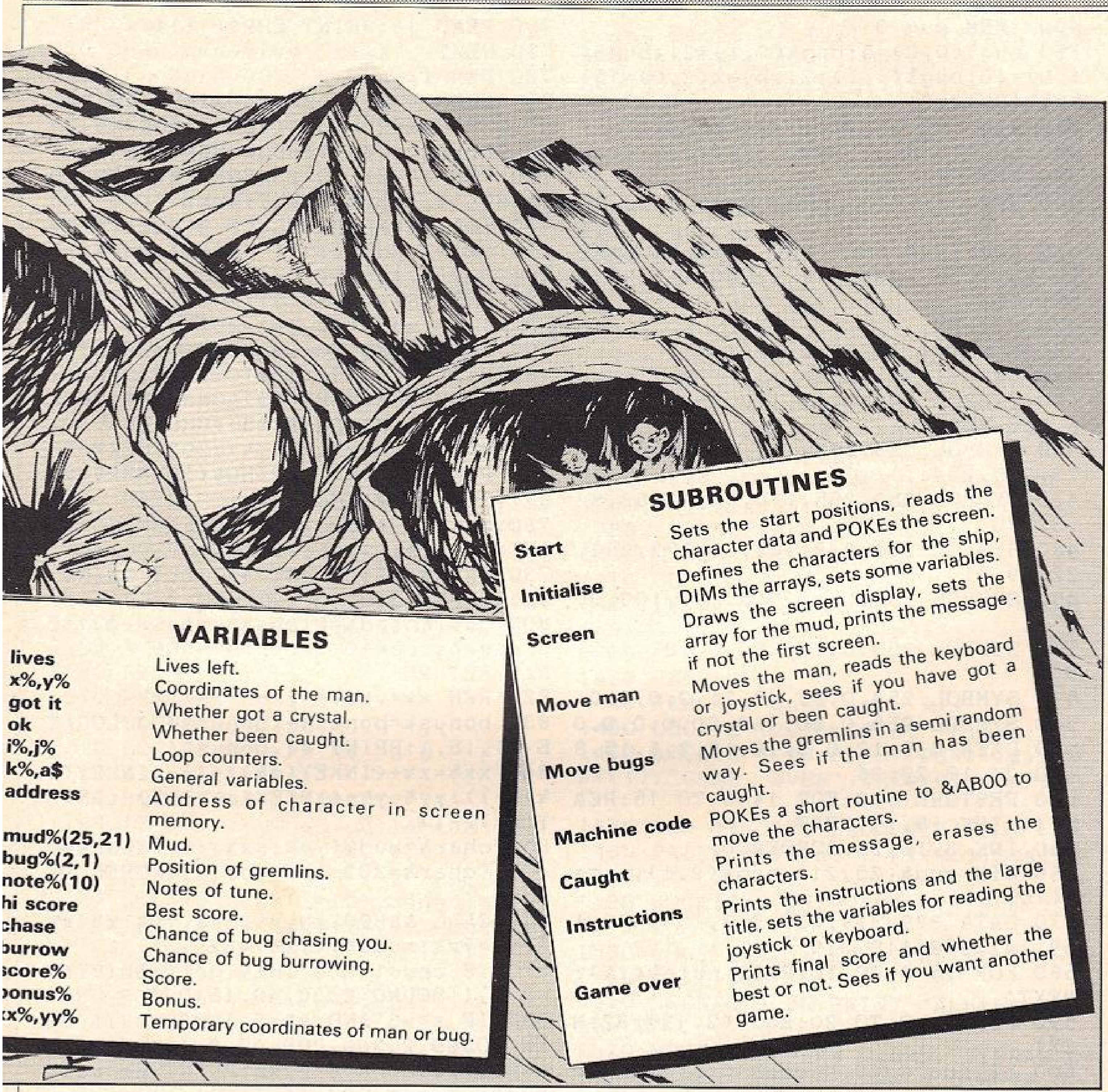
The touchdown was a bit bumpy but you survive. Your task is now to collect those crystals. They are buried beneath the mud in a cavern inhabited by gremlins.

They don't mind you taking a few as long as you keep out of their way, but the more you take, the more they chase you.

The game has been broken down into a series of subroutines, each with a specific task. All the subroutines have a title in a REM statement at the start.

When keying in this program it will probably help to set the screen width to the same as that of the listing. This is achieved very simply by using the command: CLS: WINDOW 1,34,1,25
The command should be typed in command mode (i.e. no line number) before beginning to key in the program.

```
10 REM ***** Digger *****
20 REM *By R.A.Waddilove*
30 MEMORY &A000
40 MODE 1
50 GOSUB 1200 :REM instructions
60 GOSUB 1020 :REM machine code
70 GOSUB 430 :REM initialise
80 MODE 0
90 WHILE a$="Y"
100 WHILE lives
110 GOSUB 630 :REM screen
120 WHILE collected%<8 AND lives
130 GOSUB 250 :REM start
140 WHILE ok AND collected%<8
150 GOSUB 820 :REM move man
160 GOSUB 910 :REM move bugs
170 WEND
180 IF NOT ok THEN GOSUB 1090 :REM
 caught
190 WEND
200 WEND
210 GOSUB 1440 :REM game over
```



VARIABLES

lives	Lives left.
x%,y%	Coordinates of the man.
got it	Whether got a crystal.
ok	Whether been caught.
i%,j%	Loop counters.
k%,a$	General variables.
address	Address of character in screen memory.
mud%(25,21)	Mud.
bug%(2,1)	Position of gremlins.
note%(10)	Notes of tune.
hi score	Best score.
chase	Chance of bug chasing you.
urrow	Chance of bug burrowing.
score%	Score.
onus%	Bonus.
x%,yy%	Temporary coordinates of man or bug.

SUBROUTINES

Start	Sets the start positions, reads the character data and POKEs the screen.
Initialise	Defines the characters for the ship, DIMs the arrays, sets some variables.
Screen	Draws the screen display, sets the array for the mud, prints the message if not the first screen.
Move man	Moves the man, reads the keyboard or joystick, sees if you have got a crystal or been caught.
Move bugs	Moves the gremlins in a semi random way. Sees if the man has been caught.
Machine code	POKEs a short routine to &AB00 to move the characters.
Caught	Prints the message, erases the characters.
Instructions	Prints the instructions and the large title, sets the variables for reading the joystick or keyboard.
Game over	Prints final score and whether the best or not. Sees if you want another game.

```
220 WEND
230 MODE 1:PEN 1
240 END
250 REM *** start ***
260 LOCATE 12,25:PRINT "Lives:";US
ING "#";lives
270 RESTORE 280
280 DATA 0,20,40,0,0,60,60,0,0,20,
40,0,84,252,252,168,168,252,252,84
,40,84,168,20,0,168,84,0,84,168,84
,168
290 DATA 3,2,1,3,2,2,1,1,0,3,3,0,1
,0,0,2,2,130,65,1,2,0,0,1,2,65,130
,1,1,3,3,2
300 DATA 5,15,15,10,79,143,79,143,
79,5,10,143,15,15,15,15,5,79,143,1
0,5,143,79,10,15,10,5,15,10,0,0,5
310 DATA 179,51,51,115,115,162,81,
179,115,0,0,179,34,81,162,17,17,11
5,179,34,17,17,34,34,17,0,0,34,179
,0,0,115
320 address=&C000+&50+8:GOSUB 400
:REM man
330 address=&C000+20*&50+16:GOSUB
400 :REM bug 1
340 address=&C000+20*&50+36:GOSUB
400 :REM bug 2
350 address=&C000+20*&50+56:GOSUB
```

```
400 :REM bug 3
360 bug%(0,0)=5:bug%(0,1)=21:bug%(
1,0)=10:bug%(1,1)=21:bug%(2,0)=15:
bug%(2,1)=21
370 x%=3:y%=2:got.it=0:ok=-1
380 LOCATE 8,2:PRINT SPC(10)
390 RETURN
400 REM *** poke character data **
*
410 FOR i%=0 TO 7:FOR j%=0 TO 3:RE
AD k%:POKE address+j%+&800*i%,k%:N
EXT:NEXT
420 RETURN
430 REM *** initialise ***
440 ENT -1,50,10,5
450 SYMBOL 248,3,14,117,255,146,14
6,255,255
460 SYMBOL 249,192,112,174,255,73,
73,255,254
470 SYMBOL 250,255,170,85,42,63,32
,32,80
480 SYMBOL 251,248,168,88,172,252,
22,19,41
490 SYMBOL 252,0,1,10,0,109,109,0,
0
500 SYMBOL 253,0,128,80,0,182,182,
0,0
510 SYMBOL 254,0,85,42,21,0,0,0,0
520 SYMBOL 255,0,80,160,80,0,0,0,0
530 DATA 0,9,12,0,3,10,16,2,6,15,8
,20,24,13,23,26
540 RESTORE 530:FOR i%=0 TO 15:REA
D j%:INK i%,j%:NEXT
550 INK 3,13,26:BORDER 0
560 DIM mud%(25,21),bug%(2,1),note
%(10)
570 DATA 379,319,239,213,190,478,1
59,142,379,119,319
580 FOR i%=0 TO 10:READ note%(i%):
NEXT
590 FOR j%=3 TO 20:mud%(2,j%)=32:N
EXT
600 hi.score=10:score%=0:lives=3:c
hase=1:burrow=0
610 a$="Y"
620 RETURN
630 REM *** screen ***
640 IF score%>0 THEN LOCATE 4,10:P
RINT CHR$(22);CHR$(1);"CONGRATULAT
IONS";CHR$(22);CHR$(0):FOR i%=0 TO
 3000:NEXT:score%=score%+bonus
%
650 CLS:RESTORE 660
660 DATA 248,249,8,8,10,250,251,8,
8,11,252,253,8,8,10,254,255
670 LOCATE 1,1:PEN 14:PRINT CHR$(2
2);CHR$(1);
680 FOR i%=1 TO 17
690 IF i%=10 THEN PEN 2
700 READ j%:PRINT CHR$(j%);
710 NEXT
720 PEN 1
730 FOR i%=3 TO 20:LOCATE 1,i%:PRI
NT STRING$(20,CHR$(207)):FOR j%=1
TO 20:mud%(i%,j%)=207:NEXT:NEXT
740 FOR i%=21 TO 23:FOR j%=1 TO 20
:mud%(i%,j%)=32:NEXT:NEXT
750 FOR j%=2 TO 8 STEP 2:LOCATE j%
,23:PRINT CHR$(15);CHR$(3);CHR$(23
3);CHR$(8);CHR$(15);CHR$(4);CHR$(2
02);CHR$(8);CHR$(15);CHR$(5);C
HR$(148):mud%(23,j%)=202:NEXT
760 FOR j%=13 TO 19 STEP 2:LOCATE
j%,23:PRINT CHR$(15);CHR$(3);CHR$(
233);CHR$(8);CHR$(15);CHR$(4);CHR$
(202);CHR$(8);CHR$(15);CHR$(5)
;CHR$(148):mud%(23,j%)=202:NEXT
770 PEN 1:PRINT STRING$(20,CHR$(20
8));CHR$(22);CHR$(0);
780 score%=score%+bonus%:PEN 15:PR
INT "Score:";score%
790 PEN #1,2:LOCATE #1,10,1:PRINT
#1,"Bonus 500"
800 collected%=0:chase=chase-0.1:b
urrow=burrow+0.1:bonus%=500
810 RETURN
820 REM *** move man ***
830 bonus%=bonus%+(bonus%>0):LOCAT
E #1,15,1:PRINT #1,bonus%
840 xx%=x%+(INKEY(c%)>-1)-(INKEY(d
%)>-1):yy%=y%+(INKEY(a%)>-1)-(INKE
Y(b%)>-1)
850 char%=mud%(yy%,xx%):IF char%=0
 OR (char%=202 AND got.it) THEN RE
TURN
860 CALL &AB00,x%,y%,xx%,yy%:x%=xx
%:y%=yy%:mud%(y%,x%)=32
870 IF char%=202 THEN got.it=1:INK
 7,2,1:SOUND 2,30,40,15,0,1
880 IF x%=3 AND y%=2 AND got.it TH
EN SOUND 2,200,200,15,0,1:score%=s
core%+10:LOCATE 7,25:PRINT score%:
mud%(y%,x%)=32:got.it=0:collec
ted%=collected%+1:INK 7,2
890 IF (x%=bug%(0,0) AND y%=bug%(0
,1)) OR (x%=bug%(1,0) AND y%=bug%(
1,1)) OR (x%=bug%(2,0) AND y%=bug%
(2,1)) THEN ok=0
900 RETURN
910 REM *** move bugs ***
920 SOUND 132,note%(INT(RND*11)),2
0,14:SOUND 129,478,200,12
930 FOR i%=0 TO 2
940 IF RND>chase THEN xx%=bug%(i%,
0)+(x%<bug%(i%,0))-(x%>bug%(i%,0))
:yy%=bug%(i%,1)+(y%<bug%(i%,1))-(y
%>bug%(i%,1)) ELSE xx%=bug%(i%
,0)+INT(RND*3)-1:yy%=bug%(i%,1)+IN
```

```
T(RND*3)-1
950 char%=mud%(yy%,xx%)
960 IF char%=202 OR char%=0 OR cha
r%=255 OR (char%=207 AND RND>burro
w) THEN RETURN
970 CALL &AB00,bug%(i%,0),bug%(i%,
1),xx%,yy%
980 mud%(bug%(i%,1),bug%(i%,0))=32
:mud%(yy%,xx%)=255:bug%(i%,0)=xx%:
bug%(i%,1)=yy%
990 IF xx%=x% AND yy%=y% THEN ok=0
1000 NEXT
1010 RETURN
1020 REM *** machine code ***
1030 RESTORE 1070
1040 FOR i%=0 TO 64
1050 READ a$:POKE &AB00+i%,VAL("&"
+a$)
1060 NEXT
1070 DATA DD,46,00,DD,5E,02,CD,2D,
AB,E5,DD,46,04,DD,5E,06,CD,2D,AB,D
1,01,00,08,C5,06,04,7E,36,00,12,23
,13,10,F8,01,FC,07,09,EB,09,EB
,C1,10,EB,C9,1D,21,00,C0,55,A7,CB,
13,CB,13,19,05,C8,11,50,00,19,10,F
D,C9
1080 RETURN
1090 REM *** caught ***
1100 SOUND 2,90,50,15,0,1,1
1110 LOCATE 8,2:PRINT "G U L P"
1120 lives=lives-1
1130 LOCATE x%,y%:PRINT " "
1140 IF got.it THEN collected%=col
lected%+1:got.it=0:INK 7,2
1150 FOR i%=0 TO 2
1160 LOCATE bug%(i%,0),bug%(i%,1):
PRINT " ":mud%(bug%(i%,1),bug%(i%,
0))=32
1170 NEXT
1180 FOR i%=0 TO 2000:NEXT
1190 RETURN
1200 REM *** instructions ***
1210 INK 0,0:INK 1,0:INK 2,15:INK
3,11
1220 BORDER 0:PLOT -5,-5,3:PEN 1
1230 LOCATE 1,25:PRINT "DIGGER"
1240 FOR i%=0 TO 96 STEP 2
1250 FOR j%=0 TO 16 STEP 2
1260 IF TEST(i%,j%) THEN PLOT 200+
i%*2,365+j%*2:PLOT 200+i%*2,367+j%
*2:PLOT 202+i%*2,365+j%*2:PLOT 202
+i%*2,367+j%*2
1270 NEXT
1280 NEXT
1290 LOCATE 1,25:PRINT SPC(10):INK
 1,6
1300 LOCATE 1,5:PEN 2
1310 PRINT "On your way back to st
ar base 11 you     find yourself ru
nning short of fuel.    Fortunatel
y a nearby planet, Delta 2,
has the necessary dilithium crysta
ls."
1320 PRINT:PRINT "Collect the crys
tals and take them back to your sh
ip, but watch out for the     grem
lins who inhabit the planet,th
ey    are deadly."
1330 PRINT:PRINT "At first they ig
nore you, but on later  screens th
ey will start to chase you."
1340 PEN 3:PRINT:PRINT:PRINT:PRINT
 "Use joystick or keyboard."
1350 PEN 2:PRINT:PRINT "A...up    Z
...down    <...left    >...right"
1360 PEN 1:LOCATE 1,25:PRINT "Pres
s SPACE BAR or FIRE to start..."
1370 a$="":WHILE INKEY$<>"":WEND
1380 WHILE a$<>" " AND JOY(0)<>16
1390 a$=INKEY$
1400 WEND
1410 CLS
1420 IF a$=" " THEN a%=69:b%=71:c%
=39:d%=31 ELSE a%=72:b%=73:c%=74:d
%=75
1430 RETURN
1440 REM *** game over ***
1450 LOCATE 12,25:PRINT "Lives:0"
1460 LOCATE 6,10:PRINT CHR$(22);CH
R$(1);"GAME  OVER";CHR$(22);CHR$(0
)
1470 FOR i%=0 TO 5000:NEXT:CLS
1480 PAPER 8:PRINT CHR$(30);STRING
$(100," ")
1490 LOCATE 1,25:PRINT STRING$(20,
" ");
1500 LOCATE 3,3:PEN 12:PRINT "Fina
l score:";score%
1510 LOCATE 3,25:PRINT "Another  G
ame ?"
1520 MOVE 0,0:DRAW 0,399,15:DRAW 6
39,399:DRAW 639,0:DRAW 0,0:MOVE 0,
320:DRAW 639,320
1530 MOVE 0,16:DRAW 639,16
1540 PAPER 0:PEN 11
1550 IF score%>hi.score THEN hi.sc
ore=score%:LOCATE 3,10:PRINT "This
 is the best":LOCATE 7,15:PRINT "s
o far !" ELSE LOCATE 5,10:PRIN
T "The best is":LOCATE 8,15:PRINT
hi.score
1560 a$=""
1570 WHILE INSTR(" YN",a$)<2
1580 a$=UPPER$(INKEY$)
1590 WEND
1600 score%=0:lives=3:chase=1:burr
ow=0
1610 RETURN
```

The procedure to follow is ...

Robin Nixon shows how to take a more structured approach to your CPC programming

Having previously been a BBC Micro user I was pleasantly surprised on my first encounter with the Amstrad, which has 40k available for Basic programs, a true 16 colour mode and a choice of 27 colours.

The 16 colour mode on the BBC Micro has eight normal and eight flashing colours selectable from eight and only leaves you 10k free for Basic.

Other Amstrad features I was impressed with were the comprehensive jump block, the ease with which you can add resident system extensions (RSXs), and the logical way the screen is laid out, facilitating simple development of machine code sprites.

But, and there's always a but when you compare two micros, there were one or two features I was used to on the BBC Micro which I missed when I started programming the CPC.

The first was the in-built assembler, but this has been overcome by a number of assemblers that are now available commercially.

Second came the use of procedures. The closest the Amstrad gets to procedures is the GOSUB statement which is fine for working on small programs but can make development trickier the longer a program gets. As you add more and more subroutines, remembering where a particular one is located becomes quite a headache.

The problem is avoided on the BBC Micro because procedures can replace GOSUBS and are not line dependent. If you look at Program I, which is written in BBC Basic, you'll see that line 40 is actually the equivalent to GOSUB 1000. However, the subroutine at line 1000 is identified by "name" and when working on a program you never need know where PROCname is in order to call it.

```
10    REM Program I
20    REM An example of using
        procedures
30    CLS
40    PROCname
50    REM Rest of program....
      ...
      ...
900   END
1000  DEFPROCname
1010  PRINT"Amstrad"
1060  ENDPROC
```

Program I

Without a GOSUB in sight I think you'll agree that this lends itself to structured programming as well as making your listings more legible to other people.

Program II is a routine that will give you procedures on the CPC. Save it before you run it. It sets up the RSXs |PROC, |DEFPROC, |ENDPROC and |START. |PROC works in exactly the same way as in Program I except that the syntax is slightly changed. In place of typing:

```
PROCname
```

you enter:

```
|PROC,name
```

The same goes for DEFPROC.

You also have to enter |START as one of the first lines before you use any |PROCS or |DEFPROCS. This is because you are allowed to nest procedures up to 10 deep, which is achieved by storing the return address each time a procedure is called.

|START restores the pointer to the return addresses to 0 so you start on the first level of nesting each time you run the program.

If you leave |STARTS out and press Escape, or the program stops in the middle of a few layers of nesting due to an error, the pointer won't be restored when you re-run the program. The next |ENDPROC encountered will take you back to the procedure that was executing before you pressed Escape or the error occurred.

If you try to nest procedures more than 10 deep "Too many PROCs" will be reported and you will be returned to Basic with a syntax error. A true Basic error occurs and the program doesn't attempt to continue execution.

Another way this error could occur is if you forget to enter the correct |ENDPROC for each procedure, or put it in the wrong place.

The procedure to follow is ...

Robin Nixon shows how to take a more structured approach to your CPC programming

Having previously been a BBC Micro user I was pleasantly surprised on my first encounter with the Amstrad, which has 40k available for Basic programs, a true 16 colour mode and a choice of 27 colours.

The 16 colour mode on the BBC Micro has eight normal and eight flashing colours selectable from eight and only leaves you 10k free for Basic.

Other Amstrad features I was impressed with were the comprehensive jump block, the ease with which you can add resident system extensions (RSXs), and the logical way the screen is laid out, facilitating simple development of machine code sprites.

But, and there's always a but when you compare two micros, there were one or two features I was used to on the BBC Micro which I missed when I started programming the CPC.

The first was the in-built assembler, but this has been overcome by a number of assemblers that are now available commercially.

Second came the use of procedures. The closest the Amstrad gets to procedures is the GOSUB statement which is fine for working on small programs but can make development trickier the longer a program gets. As you add more and more subroutines, remembering where a particular one is located becomes quite a headache.

The problem is avoided on the BBC Micro because procedures can replace GOSUBS and are not line dependent. If you look at Program I, which is written in BBC Basic, you'll see that line 40 is actually the equivalent to GOSUB 1000. However, the subroutine at line 1000 is identified by "name" and when working on a program you never need know where PROCname is in order to call it.

```
10    REM Program I
20    REM An example of using
        procedures
30    CLS
40    PROCname
50    REM Rest of program....
      ...
      ...
900   END
1000  DEFPROCname
1010  PRINT"Amstrad"
1080  ENDPROC
```

Program I

Without a GOSUB in sight I think you'll agree that this lends itself to structured programming as well as making your listings more legible to other people.

Program II is a routine that will give you procedures on the CPC. Save it before you run it. It sets up the RSXs |PROC, |DEFPROC, |ENDPROC and |START. |PROC works in exactly the same way as in Program I except that the syntax is slightly changed. In place of typing:

```
PROCname
```

you enter:

```
|PROC,name
```

The same goes for DEFPROC.

You also have to enter |START as one of the first lines before you use any |PROCS or |DEFPROCS. This is because you are allowed to nest procedures up to 10 deep, which is achieved by storing the return address each time a procedure is called.

|START restores the pointer to the return addresses to 0 so you start on the first level of nesting each time you run the program.

If you leave |STARTS out and press Escape, or the program stops in the middle of a few layers of nesting due to an error, the pointer won't be restored when you re-run the program. The next |ENDPROC encountered will take you back to the procedure that was executing before you pressed Escape or the error occurred.

If you try to nest procedures more than 10 deep "Too many PROCs" will be reported and you will be returned to Basic with a syntax error. A true Basic error occurs and the program doesn't attempt to continue execution.

Another way this error could occur is if you forget to enter the correct |ENDPROC for each procedure, or put it in the wrong place.

Program III

```
10 REM              Mastermind
20 REM          by Robin Nixon
30 REM (c) Computing with the Amstrad
40 REM--------- CPC Only ------------
50 |START
60 |PROC,initialise
70 WHILE 1:g=0
80 FOR x=1 TO 4:a(x)=-1:NEXT:b=0
90 |PROC,title
100 |PROC,randomnum
110 WHILE b<4 AND g<12
120 |PROC,guess
130 |PROC,result
140 |PROC,show
150 WEND
160 IF b=4 THEN |PROC,youwin
170 IF b<4 THEN |PROC,youlose
180 |PROC,playagain
190 WEND
200 |DEFPROC,initialise
210 MODE 1:BORDER 10
220 DEFINT a-z
230 DIM a(4),b(4),c(4)
240 |ENDPROC
250 |DEFPROC,randomnum
260 FOR x=1 TO 4
270 r=RND*9:|PROC,check
280 IF r=-1 THEN GOTO 270
290 a(x)=r
300 NEXT
310 |ENDPROC
320 |DEFPROC,check
330 FOR y=1 TO 4
340 IF a(y)=r THEN Y=4:r=-1
350 NEXT
360 |ENDPROC
370 |DEFPROC,title
380 MODE 1
390 DRAW 638,0:DRAW 638,398:DRAW 0,398
:DRAW 0,0
400 PEN 3:PAPER 2:LOCATE 16,2:PRINT"Ma
stermind"
410 |PROC,wind
420 |ENDPROC
430 |DEFPROC,guess
440 WINDOW 1,40,1,25
450 PEN 1:PAPER 0:LOCATE 10,24
460 a$="":PRINT "Enter your guess ?";S
TRING$(5,32)
470 PEN 2:|PROC,getnum
480 LOCATE 28,24:PRINT a$;" "
490 FOR x=1 TO 4:b(x)=VAL(MID$(a$,x,1)
):NEXT
500 |ENDPROC
510 |DEFPROC,getnum
520 LOCATE 28,24:PRINT a$;CHR$(143);
530 ik$=INKEY$:IF ik$="" THEN GOTO 530
540 IF (ik$<"0" OR ik$>"9") AND ik$<>C
HR$(13) AND ik$=CHR$(127) THEN GOTO 53
0
```

Mastermind procedures

initialise	Sets up mode, border and arrays.
randomnum	Creates a four digit random number with no repeating digits.
check	Used by randomnum to prevent repeating digits.
title	Draws the screen and title. Prompts for your guess.
getnum	Used to fetch a four digit input.
wind	Sets up parameters for the main window.
shadow	Creates a window and puts a shadow behind it.
result	Works out the result of a guess.
show	Displays the result. Red means correct number in the correct place and blue means correct number in the wrong place.
youlose	Tells you you've lost and shows the answer.
youwin	Tells you you've won.
gameover	Plays the game over sound effect and sets up the game over window.
playagain	Asks you to press space to play again.

```
550 IF ik$=CHR$(127) AND LEN(a$)<4 THE
N GOTO 530
560 IF ik$=CHR$(127) AND LEN(a$)>0 THE
N a$=LEFT$(a$,LEN(a$)-1):GOTO 520
570 IF ik$=CHR$(127) AND LEN(a$)<4 THE
N GOTO 520
580 IF ik$=CHR$(13) THEN |ENDPROC
590 IF LEN(a$)<4 THEN a$=a$+ik$
600 GOTO 520
610 |DEFPROC,wind
620 w1=13:w2=27:w3=5:w4=20:p1=2:p2=3::
|PROC,shadow
630 PEN 1:PAPER 0:LOCATE 2,2
640 PRINT "Guess     Score"
650 |ENDPROC
660 |DEFPROC,shadow
670 WINDOW w1+1,w2+1,w3+1,w4+1:PAPER p
2:CLS
680 WINDOW w1,w2,w3,w4:PAPER p1:CLS
690 |ENDPROC
700 |DEFPROC,result
710 b=0:c=0:FOR x=1 TO 4:c(x)=0
720 IF a(x)=b(x) THEN c(x)=1:b=b+1
```

```
730 NEXT
740 FOR x=1 TO 4
750 FOR y=1 TO 4
760 IF c(x)=1 THEN GOTO 780
770 IF a(x)=b(y) THEN c(x)=1:c=c+1
780 NEXT
790 NEXT
800 |ENDPROC
810 |DEFPROC,show
820 g=g+1:LOCATE 14,g+7
830 PEN 3:PAPER 2:PRINT a$;STRING$(5,3
2);
840 PEN 3:PRINT STRING$(b,233);
850 PEN 0:PRINT STRING$(c,233);
860 |ENDPROC
870 |DEFPROC,youlose
880 |PROC,gameover
890 PEN 0:PAPER 1:LOCATE 11,2:PRINT"Yo
u lost this game"
900 PEN 3:LOCATE 10,3:PRINT"The soluti
on is   ";
910 a$="":FOR x=1 TO 4:a$=a$+CHR$(a(x)
+48):NEXT
920 PRINT a$
930 |ENDPROC
940 |DEFPROC,youwin
950 |PROC,gameover
960 PEN 0:PAPER 1:LOCATE 11,2:PRINT"Yo
u won this game"
970 |ENDPROC
980 |DEFPROC,gameover
990 FOR x=0 TO 500 STEP 17
1000 SOUND 2,x,2:NEXT:FOR x=1 TO 1000:
NEXT
1010 w1=2:w2=38:w3=18:w4=23:p1=1:p2=3:
|PROC,shadow
1020 |ENDPROC
1030 |DEFPROC,playagain
1040 PEN 2:PAPER 0:LOCATE 7,5
1050 PRINT "Press SPACE to play again"
1060 WHILE INKEY$<>"":WEND
1070 WHILE INKEY$<>" ":WEND
1080 |ENDPROC
```

BCPL Designed for humans not Computers

...continued

combined with the quite thick manual, will prove sufficient for a fairly intelligent novice to learn BCPL. However a good book will ease the process.

There are both serious and more light hearted applications to study and I must admit to having headed straight for the space invaders game first!

On a more serious note there is a 6502 cross assembler, a debugger, text editor, file copy program and Z80 disassembler.

The text editor is for entering BCPL source code and produces pure Ascii files. It is fairly rudimentary, with the bare minimum of functions. There are better editors around and it is provided merely as a stop gap to get you going straight away.

Alternatively you can use Protext or Maxam, and this is particularly useful on the CPCs if you have the rom versions. Source code can be compiled directly from memory speeding up the process enormously, not that the compiler is slow anyway.

The cross assembler enables you to enter 6502 assembly mnemonics which it compiles to 6502 source code. The BCPL source file is interesting and there is much to learn form studying it but I'm not convinced that the program is actually useful.

The code produced would need to be transferred to a 6502-based micro before it could be run. This would be very time consuming and I'd rather use a 6502 assembler on the target micro.

The debugger is useful for debugging BCPL object code. The language is so flexible and powerful there is little restriction on what you can do. This makes it very easy to crash the system if you're not careful. You can simply step through the program and identify the problem area quite easily with Debug.

Arnor have built up an enviable reputation for producing quality products and BCPL is a worthy addition expanding an already excellent range.

The thing I like about Arnor products is that they are designed for humans, not computers. This makes them easy to use, yet both flexible and powerful. I shall certainly be using BCPL for some time to come.

"Messy-dos" ?? Forget it...

Our congratulations to Tony Blakemore on his appointment as Computer manager of the Billy Guyatt chain of stores in Victoria. Congrats are also in order to senior management of 'King Billy's' for having the foresight to make the appointment. Other chains would do well to follow suit as this can only lead to an overall improvement in product knowledge of those on the selling side and a consequent increase in sales of Amstrad computers.

Historically, Mitsubishi-AWA have dealt with the likes of Billy Buyatts, Retravision, Grace Bros. and others in the white goods areas supplying TVs, Video Recorders and Hi-fi. When Alan Sugar introduced the Amstrad range it was only natural that M-AWA would wish to use their existing dealer network to aid in the launch. In the early days the rush caused by the popularity of the products themselves created a whole new area of sales for both M-AWA and their dealers. Unfortunately, the area of product training was sadly neglected leading to numerous complaints about lack of service from the chain stores. Happily, over the fullness of time and in large part due to the creation of user groups by dedicated individuals the problems have been gradually resolved.

Alan Sugar's apparent determination to become the world's number 1 computer supplier in record time now sees us at the beginning of a new rush. Unfortunately for those on the sales side this plunge into PC compatibility will make learning about CPCs and PCWs seem like throwing a brick compared with building a house when they have to know about such things as Messy-DOS, DOS Plus, GEM and Basic 2 (to paraphrase a well known American TV ad — "Where's the memory?").

In Australia and probably other parts of the world Amstrad dominance will only be achieved when a prospective buyer an confidently approach a salesperson in any Amstrad dealership and be told the truth about the product and its support. We'd like to see a 'Buyers Charter' established along the lines of a code of ethics which would help prospective Amstrad buyers to weed out the good from the bad in dealers — let's have your ideas.

None of the above takes into account the situation in the Ma and Pa computer store which is a whole different kettle of fish and I look forward to discussing their side on another occasion.

Computing With The Amstrad

'Computing With The Amstrad' welcomes program listings and articles for publication. Material should be typed or computer printed, and must be double spaced. Program listings should be accompanied by cassette tape or disk. Please enclose a stamped addressed envelope or the return of material cannot be guaranteed. Contributions accepted for publication by Database Publication or its licensee will be on an all-rights basis.