1932

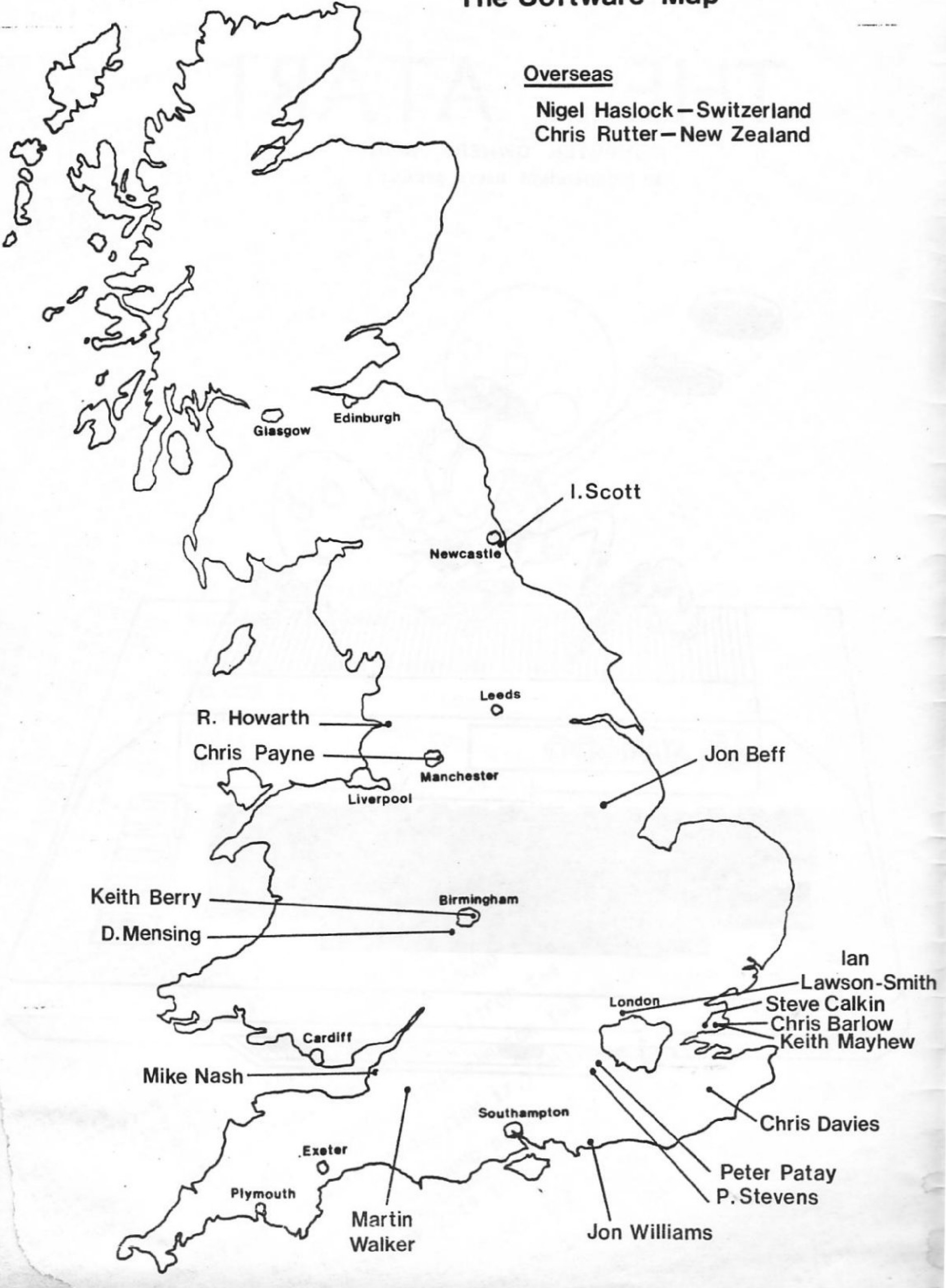
No. 2

THE U.K. ATARI

COMPUTER OWNERS CLUB

(an independent users group)

The Software Map



Overseas

Nigel Haslock—Switzerland
Chris Rutter—New Zealand

EDITORIAL.

Welcome to the second issue. Sorry about the delay in its production, but we hope that this has not discouraged you and that no delays occur again. Most of you will by now be aware that we have unfortunately had to increase the membership fee to three pounds. This is not because we wish to spend six weeks in the sunny south of France, but is necessary to cover the increased cost of producing and distributing the newsletter. There have been changes in the editorial staff. Graham Daubney who was co-editor of the user group is no longer with us, but fortunately we have three new staff, Sue Clark, Roy Smith (Bif), and Keith Mayhew.

This small, but dedicated hard working (well, most of the time!!) group of people, have devoted many late hours in producing this issue. So that their efforts do not go unrecognised, here they are.

Editor:________________________Chris Barlow, Tech.(CEI).

Technical Editor:_______________Keith Mayhew (machine code programmer.).

Software Librarian:_____________Ron Levy (G.I.B).

Technical Illustrations & cartoons:__Roy Smith (draftsman).

Text proof reader:______________Sue Clark.

We hope there are not as many errors (if any), in this issue as there were in the first. The form in which we are now producing the newsletter has changed and we hope you find it easier to handle than the earlier issue, which was printed on A4 paper and stapled at one corner. As you see, we are now printing on A3 size paper and using a different style of text. The entire text in the newsletter has been created with the ATARI word processor system, using the CENTRONIC's 739 printer, with Keith at the keyboard.

In this issue we present new regular features, interesting tutorials, and a competition, plus five new programs.

The Programs, SKYBLITZ, COLLISION COURSE, FRUIT MACHINE, FLASHING CURSOR, SNOOPY.

HARDWARE NEWS

THE ATARI 400 GROWS UP

It has often been reported in the popular computer press by authors who are not aware of the current developments in extended memory cards that the ATARI 400 is a less powerful machine than the '800 because until recently the maximum amount of RAM was 16K, and this meant that the system was not practical to use with a disk drive. The reason that a 16K machine is not suitable for use with a disk drive is that by the time you have 'booted' DOS (Disk Operating System) the amount of user RAM left is insufficient for any lengthy programs.

The answer to this problem is obvious, you need more than 16K of RAM. ATARI, however, only supply a RAM card of 16K and the '400 only has one slot for a single RAM card, thus restricting the system to 16K. The development of a RAM card containing more than 16K was started some time ago in the U.S.A. The first generation of these cards were 32K and used sixteen RAM chips on a single card, but had the undesirable effect of causing disturbing screen patterning. This was cured on later cards and the results obtained from these cards are now acceptable in their performance. The next development came with the 48K RAM card, this unfortunately had several design problems, firstly very severe picture disturbance which drove RON up the wall when he had one installed in his first '400. Second he noticed an increase in the running temperature in both his ATARI and it's power supply. Although this did not cause any noticeable damage in the short term, it obviously made RON wary of leaving his machine on for long periods of time. The reason for this increased temperature can only be put down to the high number of RAM chips used on the double decked card assembly (Twenty four). This design of double stacked card has now been superceded by a single card containing only eight RAM chips, thus making the overall power consümption lower and reducing the working temperature. The other major difference is that the new 48K card causes no adverse picture interference. However, the installation of a RAM card greater than 32K requires a small hardware modification. This is necessary because the RAM slot cannot normally provide 48K adressing capabilities. The modification is only the addition of four short wire links to the underside of the '400's main circuit, board and this is done by the retailer when he installs the 48K card. The installation of this card or upgrade offered by most retailers includes the exchange of your existing RAM card thus making the upgrade to 48K very economic.

This new 48K card was developed by MAPLIN ELECTRONIC SUPPLIES Ltd. and is distributed by MAPSOFT Ltd to other retailers. Until recently the guarantee would be invalidated on any '400 with extended memory. However with the advent of the MAPLIN/MAPSOFT 48K RAM card, should your machine develop a fault then you are still able to have it repaired under it's original guarantee period. The MAPLIN 48K card itself carries its own one year guarantee effected from date of purchase.

With this increase in memory capability, even when running with a disk drive there is now sufficient RAM to allow you to write very long programs and this means that the power of the '400 is now directly comparable to the '800 with 48K RAM. This leaves the only other major difference between the two, that of the keyboard. The '400 uses the membrane and the '800 the more conventional typewriter style keyboard. Although to our knowledge there are no quality manufactured keyboards now available to replace the membrane on the '400, we look forward to seeing future developments in this field and also pricing details.

We hope to bring you more news on '400 and '800 hardware developments. If you know of any hardware developments, rumours or otherwise, we would be grateful to hear from you.

SOFTWARE NEWS

Unfortunately we are not able to bring you news on all the recently released software for your ATARI, because there is so much of it available!!! In reviewing the latest releases, which are all to a very high quality of programming and originality, we have selected a few of the outstanding programs.

CHOPLIFTER

By: BRODERBUND SOFTWARE. Author: DAN GORLIN.
Disk: 32K. Type: MACHINE CODE HIGH-RES SIMULATION GAME.
Joysticks: 1. No. players: 1.

This is a simulation of a rescue helicopter on a mission to save as many hostages as possible. The obstacles which stand in your way include tanks firing at you and intercept missile carrying jet fighter and as the game progresses you are confronted by even more feindish persuers. Dodging these attacks you must blast open the buildings where your hostages are held, and once freed you must land your helicopter and allow as many men to board as possible before having to take evasive action from your attackers. The maximum the helicopter can carry is sixteen men, at which point you must return to your base to safely despatch the rescued men. Whilst this is happening, the men left on the ground are running about frantically, waving their arms and dodging the shells fired by the tanks. Men can also be destroyed by your own careless firing and if you land on top of a man you squash him into the ground!! The technical aspects of the game include high resolution characters, effective 3D perspective and realistic helicopter movement routines. The simulation is presented in graphics mode eight and although the sounds are realistic, they do tend to be monotonous and lacking in originality. Our conclusions are that CHOPLIFTER is an excellent simulation and makes good use of the ATARI's hi-res graphics.

S.A.M.

By: DON'T ASK COMPUTER SOFTWARE. Author: MARK BARTON.
Disk: 32K. Type: MACHINE CODE SPEECH UTILITY.
Additional hardware: NONE.

S.A.M or Software Automatic Mouth, is a speech synthesiser that is entirely software generated, by this we mean NO additional hardware is necessary. This means the cost of implementing synthetic speech from a computer has been dramatically reduced and as DON'T ASK COMPUTER SOFTWARE state in their literature "Talk is cheap". Other systems use very little software, but vast amounts of hardware, comprising of speech generating I.C.'s, additional ROM's, buffer RAM and interfacing logic. The cost of a speech system could be one hundred pounds plus for a reasonable unit. In addition most of these hardware orientated systems have a limited vocabulary of words because they do not possess the true ability of speech synthesis and rely on fixed data held in ROM's. The only way of increasing the vocabulary is to increase the amount of ROM I.C.'s in the system and obviously this leads to an increase in cost. There are now available hardware systems of more powerful design which generate speech by the use of phonemes, these release the user from the constraints of a fixed vocabulary, infinitely increasing the potential of speech construction. But this system is still highly hardware orientated, thus making this additional feature of speech to your computer an expensive undertaking.

We are fortunate that ATARI had the foresight to design into the ATARI a sound generating system with precise software control, thus making it possible to generate realistic sounding speech from software. S.A.M is booted on power-up, and is a machine code program accessed from BASIC or from your own machine code program. To use S.A.M. from BASIC is very simple. Phonemes are entered into SAM$, then executed by using the USR function, S.A.M. will speak. If you wish to enter standard english text into SAM$ you must first load another machine code program called "RECITER". Although it is easier to enter english text, this restricts your control of the inflection and pronounciation. Using phonemes from S.A.M gives you much greater control of how the words are spoken. RECITER is less accurate and may mis-pronounce certain words, but this is an acceptable compromise if there is a lot of text to be translated. To demonstrate, there is a short BASIC program called "SAYIT" which allows you to experiment with phonemes and text translation. The other programs on the disk include "DEMO", which demonstrates S.A.M.'s features during which control of pitch, speed and expression are heard. The "SPEECHES" demo includes an introduction, Shakespeare, Gettysburg Address and Pledge of Alligance. "GUESSNUM" is a vocal version of a simple number guessing game which tells you if your guess is too high or too low from the randomly chosen secret number. In conclusion, we found the speech generated by S.A.M was of excellent clarity and adds a new and exciting potential to programs that you may wish to write in the future. We leave it to your imagination. Here are just a few ideas, verbal adventure, speaking clock, story teller etc.

PREPPIE

By: ADVENTURE INTERNATIONAL. Author: RUSS WETMORE.
Cassette: 16K. Type: MACHINE CODE GAME.
Joysticks: 2. No. players: 2.

Those of you who have played the popular arcade game FROGGER will already know the basic game concept, that of running or jumping across a series of obstacles to reach the goal. Preppie is a golfer who continually has to retrieve his golf balls from extremely difficult terrain. This game has a very colourful display with fine horizontal scrolling and clever use of multi-colour player graphics. The game is further enhanced by realistic musical routines, as proper attack sustain and decay envelope shaping has been implemented. Of the several 'FROGGER' type games available PREPPIE stands out as being one of the most entertaining version we have seen.

As we said earlier we are not able to bring you detailed news on all the recently released software, but we hope to bring you information in future issues. We hope to include reviews on titles such as: EMBARGO, FIREBIRD, CLOSE ENCOUNTERS, FILEMANAGER 800, DISK DETECTIVE, KRAZY KRITTERS, KSTAR PATROL, MICRO-PAINTER, CANYON CLIMBER, ZORK ADVENTURE and TUMBLE BUGS.

SKYBLITZ COMPETITION.

The most difficult aspect of writing a computer program, or indeed writing anything, is starting with nothing but a blank page in front of you and trying to think where to start. With this in mind we have already written a working program in BASIC called 'SKYBLITZ', a listing of which appears in this issue.

What we hope you will achieve is a re-written version adding any refinements or additions to the basic program, or indeed translating the concept into machine code (hybrid or pure machine code). Your modifications, and additions can take the form of improved graphics and sounds as well as increasing the game's contents overall. Although we do not stipulate a maximum memory size we would appreciate you limiting your talents to run on a minimum system of 16K RAM. If possible we would prefer your finished program on a cassette or disk, so please send a s.a.e.

We would like every member to submit a version of the program 'SKYBLITZ'. Obviously your programming skills or the amount of time you are able to devote to this task will influence the finished version of the program.

To add extra incentive we offer a prize to the most original and elegant re-write. The prize will be awarded at the end of February 1983 and the prize will be --- wait for it --- THIRTY pounds worth of software for your system chosen from the MAPLIN library, such titles as STAR RAIDERS t.m. or, for the more ambitious ATARI user, we offer the ATARI ASSEMBLER EDITOR, and in case you do not have a current MAPLIN software leaflet we enclose a copy.

The rules for entering the competition are very straight forward, they are:

1. You must be a member of the U.K. ATARI USERS GROUP.

2. You must not use any routines from any published programs, except our newsletters.

3. Any modifications must not contain offensive material.

4. You must be between the age of three years to a hundred and three years.

5. You must be male or female (not both).

6. This competition is not open to any persons directly involved with the running of the user group.

The winner of the competition will be decided by the judges, who are us!! i.e. Ron, Chris and Keith. The judgement is final and irreversible.

A selection of the best will be published in future issues and if we have an encouraging response, we hope to repeat this competition, but with a different program (of course!!!).

READERS LETTERS

These pages are devoted entirely to your comments, problems and contributions. If you have any problems or good ideas we will be glad to hear of them for future issues.

TRANSFERRING ROUTINES FROM PROGRAM TO PROGRAM.

By Keith Berry.

It often happens when creating a program that you realise that the subroutine you need exists already in another program you have on tape or disk. Here is a straightforward way to incorporate it into your new program.

Make a note on paper of the line numbers that you wish the subroutine to occupy and save the program you have just been writing onto cassette or disk using LIST "C:" or LIST "D:filename.ext". Now load the earlier program that contains the required subroutine, and LIST until the section that you need fits onto the screen. Delete unwanted lines above and below, then move the cursor to the next free line after the last line of the routine. Type NEW and press [RETURN]. Now move the cursor back, immediately below the last line and press [SHIFT] and [DELETE] to erase the words NEW and READY. Use [CTRL] and [=] to move the cursor down until it appears on the top line, then keep pressing [RETURN] until the cursor returns to the line below the subroutine.

Type LIST and you will see only the section of the program that was on the screen remains in the computer's memory. It just remains to renumber the line numbers to within the range that you had noted down (checking first for GOTO's) and then, for safety save this routine to tape, again using LIST "device:filename.ext".

Finally, take the tape containing the program that you had started to write and load it using ENTER "device:filename.ext". You should now find that both sections of program have merged into one. If they have not then load both again using ENTER"device:filename.ext"

Comment... Many thanks to Keith, I'm sure that many of our members will find this information of use when writing programs in the future.

OOP's sorry!!

Mr. M Nickels

I am experiencing problems with the character set redefinition programs in your first issue. All I seem to get is a lot of strange garbage on the screen. I think that my computer may be faulty, or is it that there is an error in the programs. As my machine recovers when I press system reset, I feel the latter may be true. Please could you help me in solving this problem.

Comment... We have recieved several letters in the same vein as Mr Nickel's letter, we can only sincerely appologise for the errors which slipt through in issue 1. The problem is that the programs were written by Graham and he neglected the fact that not every one owns a 48K machine. This means when you try and RUN the programs the machine was looking at a non-existent area of memory, this occured by POKEing 152 into location 756 (character set pointer), to correct this mistake use 56 instead of 152 where ever this number appears in the programs. This modification will then make the program suitable to RUN on a 16K or 32K system.

96K ??

Mr Simon Clark

I have an ATARI 800 with 16K and intend to expand my machine, if I buy three 32K boards will this give me 96K of memory? Also, can you tell me if this is the maximum amount of memory you can have?

Comment... We hate to disappoint you, but the fact is that any 8 bit micro such as the ATARI's 6502 can only address a maximum of 64K. This amount of memory has to be shared between ROM and RAM, and in the ATARI system 48K is available for RAM. Although several devices have been developed in the States which allow greater than 48K to be installed, these merely swap a 16K block between different RAM cards, and is only suitable for data storage, as it is not part of the permanent system RAM.

Players on graphics 0

Mr Paul Rogers

I have used players on graphics 8 successfully but when I try using players on graphics 0, I find my shapes turn into, well, all I can describe them as is a strip of random patterns. Does this mean that their is a fault in ATARI's design and that they can't be used in this mode?

Comment... The answer is quite simple, players function independantly of graphic modes being displayed, but they must be placed in an area of RAM not being used. What you have probably have done is placed your P.M. table in an area which conflicts with the display system in other graphic modes. To ensure finding a free area of memory, the solution is to use the result of PEEK(106)-16 for PMBASE.

Cassette audio

Miss Janet Kemsley

I have used the ATARI FRENCH LANGUAGE cassettes and I would like to know how to get the sound channel through to the T.V. speaker in my own programs. Have you any suggestions?

Comment... To switch on the cassette 'POKE 54018,52', this will turn on the cassette motor and you will hear whatever sound is on the tape. To turn the cassette off use 'POKE 54018,62'. This function is useful if you experience difficulty in loading a tape, to find the start of the program you can listen for the high pitch leader tone, at this point stop the cassette and load.

We must conclude this page of your letters, but we thank all those who have written to us and look forward to publishing a further selection at a later date.

An Introduction To ATARI GRAPHICS

By Martin Taylor

As a professional games programmer, one of the questions which I find myself answering with increasing regularity is how does a computer produce a picture. In this artical I hope to briefly explain how the ATARI 400/800 computer systems go about this. Although this text relates particulary to the ATARI's, the general theory of this type of computer graphics applies to many of the computer/graphics machines currently available.

Firstly let us cover a very basic computer/Video system, consisting of a CPU, (Central Processing Unit), Memory and Video interface unit. As can be seen from Fig.1 the CPU and memory are connected together by what is known as a system's bus. This bus carries control information, address and data which enable the computer to store data items to and from the memory. The Video interface is attached to the memory and uses the store information to construct the video output.

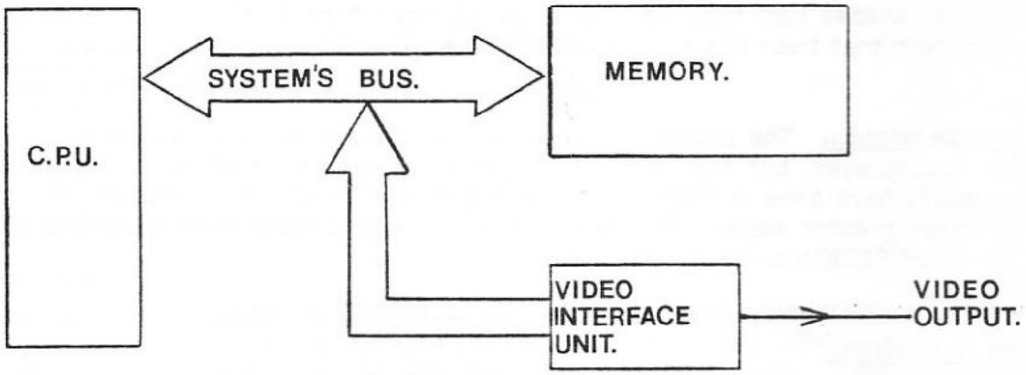


FIG 1.

A Video Interface can interpret this stored information in a variety of ways. Two methods are:

1) Bit Mapped Graphics

This is probably the most easily understood of these two graphics methods. Put simply an area of memory is mapped out in relation to the video screen, i.e. each bit of memory has a unique position on the screen associated with it. Thus a picture can be drawn, for example, by setting a particular series of bits to 1's and so illuminating that set of dots, called Pixels, on the screen. With this simple system a bit set, i.e. =1, then that pixel is illuminated, bit reset, i.e. =0, then it is dark. Therefore black and white picture, composed of a large number of pixels can be produced. How then are colour pictures produced using a bit mapped method, such as with ATARI BASIC graphics modes 3 to 8. The system is much the same , but instead of just using a single bit to indicate an illuminated pixel it uses a number of them per pixel element. Different pixel colours can therefore be represented by different bit patterns.

The ATARI carries this method one step further in that instead of using a particular 4 bit pattern to represent one colour, a 2 bit pattern is used to point to colour registers. Thus if a part of the display is formed by using a bit pattern which refers to a particular register, then all these pixels can change their colour simultaneously by altering the value in that colour register. ATARI manuals refer to this form of display as 'Playfield Graphics'.

2) Character Graphics

Usually all forms of computers which rely on Video instructions using what is known as a Memory Mapped Character Display. That is for example the screen with which you enter your BASIC text. Its implementation is slightly more involved than the bit mapped method, but is nevertheless simple to understand. A computer using this display method has an area of memory which is allocated for screen use in a similar way to that of a bit mapped display. This memory area is known as a Screen Table. The display is interpreted by taking one Byte, eight bits, at a time and using that number to reference a CHARACTER TABLE stored somewhere in the computer's memory. This table contains all the information to display a complete character. For example on the ATARI Graphics mode 0 a character is composed of eight pixels by eight pixels, as in Fig.2. Thus it can be seen that the memory needed to hold the information for a complete screen using character graphics is less than that memory requirement of a bit mapped screen for the same number of pixels.

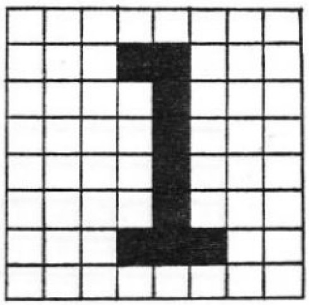


FIG 2.

This character table is usually held in ROM, (Read Only Memory), so the characters are present for use when the machine is powered on. However this charcter table does not have to be ROM, it can be held in RAM, (Random Access Memory [read/write]), which enables the programmer to define his/her own character set. These characters do not have to be single individual numbers or letters, they can be parts of a display which, when the correct arrangement of character codes are made in the screen table, form together to produce a complete picture. This method of picture production can use less memory than a bit mapped screen and produce comparable results. This technique of computer graphics is known as Redefined Character graphics or Character Slot graphics.

In the next artical I will discuss a unique feature of the ATARI-THE DISPLAY LIST.

Player/Missile Graphics

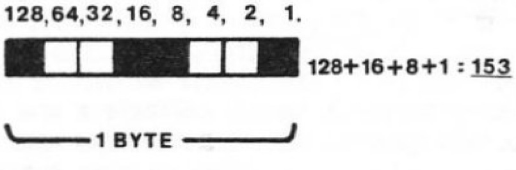
By Keith Mayhew / Ron / Chris

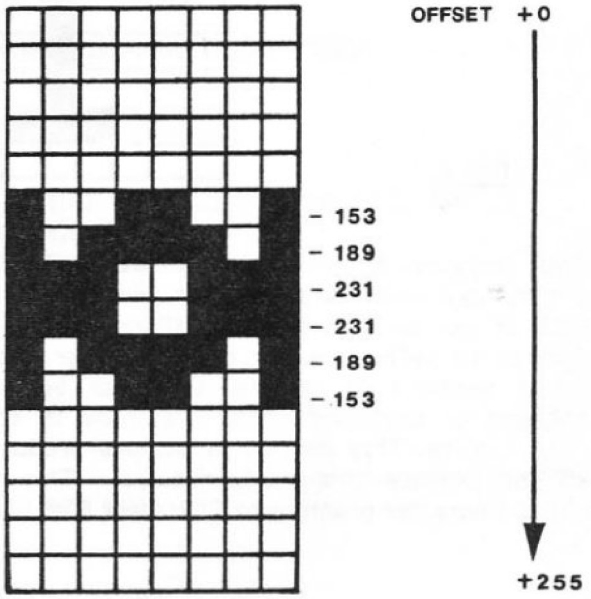
Player/missile graphics is one of the most powerful features of the ATARI 400/800, and used wisely can give the most pleasing of results.

In this artical we hope to be able to unravell the mysteries of player/missile graphics. As well as this we shall be presenting some utility routines, one in Machine Code which can be used with BASIC by those of you who grasp the principles of player/missile graphics, and would like to further enhance its speed and flexibility.

Though player/missile graphics (PMG) appears to many as a rather complex system, it is really developed from a very simple concept. The idea is that one can create a shape in an area of memory and the shape is then automatically superimposed onto the screen by the famous GTIA chip. Various factors such as horizontal position, size, colours etc. can be changed at will by the programmer by simply altering the contents of the associated memory locations.

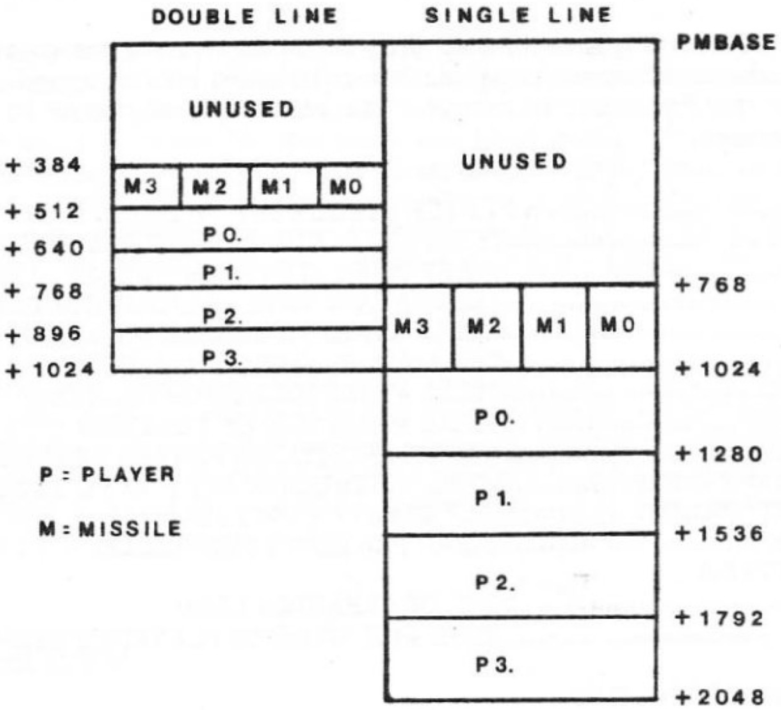
A player consists of a set of memory locations which are mapped onto the screen by GTIA as a vertical strip from the very top of the T.V. screen to the bottom. This strip is composed of eight horizontal pixels per line, thus only one byte is required per line. Usually there are 256 lines in the vertical direction, therefore each player needs a block of 256 bytes of memory. To create the shape required one must use the same principles as that of character set redefinition, i.e. various numbers are 'POKEd' into the relevant bytes of memory to illuminate the correct pixels. It must be remembered at this stage that the screen graphics mode and its contents bear no relation at all to your player/missile character and consequently each is unaffected by the presence of the other. Here is a typical mapping of a PM character.





You will note that we have placed the value of each byte on the right-hand side of the diagram. The first task is to tell the computer where in memory we intend to place our player/missile table. This is done by poking a number into the player/missile base register. This register is a memory location (54279) called PMBASE. This number is calculated by dividing the address of the first byte of the PM table by 256. You must also remember that your start location has to be on a 2K (2048) boundary, i.e. numbers in multiples of 2K, e.g. 2048, 4096, 6144.

PMBASE is now pointing to the start of your player/missile table. This table consists of 4 players and 4 missiles and 768 bytes of unused memory. We have drawn the table below showing the memory offset to each player's and missile's sub-table.



Now you have defined where in memory your table is you must then clear all the players and missiles by changing their memory locations to zero's. If however your table has been placed in an area of blank memory then this 'zeroing' process is not required. You are now able to create your player/missile characters. At this stage we strongly recommend that you read and understand fully the principles involved in bit mapping of pixels in byte form as explained in the article on character set redefinition (issue 1 of User Group). The vertical position of a player character will be determined by the offset of the first byte of this character from the start of this player's sub-table. Once you have created your shape you must now enable GTIA to display the character by means of two 'POKES', these are to SDMCTL(559) and GRACTL(53277). SDMCTL instructs GTIA to set the mode of PM display i.e. 62 for single line resolution or 46 for double line resolution (which will be explained later). The GRACTL register must be set to 3 for player/missile graphics, this value instructs GTIA to actually produce your player characters on the T.V. screen.

At this point your character will actually exist, but you will not see it because it is at position '0', which puts it just off the left hand-side of the screen. To position it in the visible screen one must change the contents of the horizontal position register for that player. The position registers are called 'HPOSP0' to 'HPOSP3' and 'HPOSM0' to 'HPOSM3' (53248 - 53251, 53252 - 53255), respectively. These registers can be altered from 0 to 255, but the extreme ranges may be off the sides of the screen. This can be a useful function for effectively switching individual characters on and off the screen. The next task is to choose a colour for your character, this is a little more trickey, but similar to the use of BASIC's 'SETCOLOR' command. To obtain the value to 'POKE' into the PM colour registers one must first look up the value of the required colour (0 - 15) and multiply this by sixteen, then add the luminance value (0 - 15). This value will be between 0 and 255 and should then be placed in the colour register. These are 'PCOLOR0' to 'PCOLOR3' (704 - 707), respectively, but we must add at this point that the missiles do not have their own colour registers, they simply use the associated player's colour.

You now have a functioning player character which you may move horizontally and change colour at will. The present horizontal width of the character is it's default value, but this may be changed to DOUBLE or QUAD size by altering the value in the appropriate SIZE register. There will still only be eight pixels across each character, but these pixels will be elongated in the horizontal direction. These registers are called SIZEP0 to SIZEP3 (53256 - 53259). The default values of the size registers are zero, double size is one, and quad size is three.

At this stage we will show a short BASIC program demonstrating the construction of a simple player/missile character construction and it's positioning on the screen. Note in this program the colour of the PM is set to orange. The REM's may of course be ommitted or re-spaced to your preference.

```
10 REM ..........................................PLAYER MISSILE SET UP DEMO.
20 PMBASE=54279:REM .............................TELLS ATARI THE BASE OF THE PM TABLE(/256).
30 PM0OFF=1024:REM ..............................START OF PLAYER0 SUB-TABLE.
40 SDMCTL=559:REM ...............................TELLS ATARI TO SET SINGLE/DOUBLE RES.
50 GRACTL=53277:REM .............................TELLS ATARI TO ENABLE PM GRAPHICS.
60 HPOSP0=53248:REM .............................TELLS ATARI HORIZONTAL POSITION OF PLAYER0.
70 PCOLOR0=704:REM ..............................TELLS ATARI COLOUR OF PLAYER0.
80 SIZEP0=53256:REM .............................TELLS ATARI SIZE OF PLAYER0.
90 YPOS=100:REM .................................OFFSET, POSITIONS PLAYER VERTICALLY ON SCREEN.
100 START=(PEEK(106)-16)*256:REM ................FIND SUITABLE START FOR PM TABLE.
110 POKE PMBASE,START/256:REM ...................SET START OF PM TABLE .
120 FOR A=0 TO 256:REM ..........................CLEAR PLAYER0'S SUB-TABLE.
130 POKE START+PM0OFF+A,0
140 NEXT A:REM ..................................END OF CLEARING LOOP
150 FOR A=0 TO 17:REM ...........................READ AND STORE IN PLAYER0'S SUB-TABLE.
160 READ DAT
170 POKE START+PM0OFF+YPOS+A,DAT
180 NEXT A:REM ..................................END OF DATA READ LOOP.
190 DATA 153,153,153,189,189,189,231,231,231
200 DATA 231,231,231,189,189,189,153,153,153
210 POKE SDMCTL,62:REM ..........................SET SINGLE LINE RESOLUTION.
220 POKE GRACTL,3:REM ...........................ENABLE PM GRAPHICS.
230 POKE HPOSP0,120:REM .........................SET PLAYER0'S HORIZONTAL POSITION.
240 POKE PCOLOR0,47:REM .........................SET PLAYER0'S COLOUR.
250 POKE SIZEP0,1:REM ...........................SET PLAYER0'S SIZE.
```

Now we have demonstrated how to generate a PM character on the screen we will now move onto it's animation. First we will deal with horizontal movement and then the more complex vertical movement. Now add the following additional lines to your program.

```
260 XPOS=120:REM SET STARTING HPOS
270 S=STICK(0):REM GET STICK VALUE
280 IF S=7 THEN IF XPOS<255 THEN XPOS=XPOS+1:REM RIGHT
290 IF S=11 THEN IF XPOS>0 THEN XPOS=XPOS-1:REM LEFT
300 POKE HPOSP0,XPOS:REM MOVE PLAYER
330 GOTO 270:REM LOOP BACK
```

The above is a simple routine which alters the players horizontal position by changing the value in it's horizontal position register. The value is calculated from the joystick in port 0, try it and see!

```
310 IF S=14 THEN IF YPOS>0 THEN GOSUB 340:REM .........GO TO 'UP' ROUTINE.
320 IF S=13 THEN IF YPOS<255 THEN GOSUB 440:REM .......GO TO 'DOWN' ROUTINE.
330 GOTO 270:REM ......................................LOOP BACK
340 FOR A=0 TO 17:REM .................................CLEAR OUT PLAYER
350 POKE START+PM0OFF+YPOS+A,0
360 NEXT A
370 RESTORE :REM ......................................RESET DATA POINTER.
380 YPOS=YPOS-1:REM ...................................CHANGE CERTICAL OFFSET.
390 FOR A=0 TO 17:REM .................................READ DATA BACK IN.
400 READ DAT
410 POKE START+PM0OFF+YPOS+A,DAT
420 NEXT A
430 RETURN :REM .......................................RETURN TO LOOP
440 FOR A=0 TO 17:REM .................................SAME AS ABOVE ROUTINE FOR OPPPOSITE DIRECTION
450 POKE START+PM0OFF+YPOS+A,0
460 NEXT A
470 RESTORE
480 YPOS=YPOS+1
490 FOR A=0 TO 17
500 READ DAT
510 POKE START+PM0OFF+YPOS+A,DAT
520 NEXT A
530 RETURN
```

Above you see a more complex routine, which, although rather crudely, provides vertical movement for your player. It does this by firstly erasing the origional player and then redrawing it at the new position, hence the RESTORing of the DATA statements used in the set-up program. You will notice however, that although the horizontal motion is of a reasonable speed, vertical motion is grindingly slow. There are various cunning methods which can be used to speed this up, such as using BASIC strings to store the data for the player, but we shall not dwell on these for we intend to provide you with a versatile block data move routine in machine code.

```
0100 FROM  =     $CB
0110 TO    =     $CD
0120 TEMP  =     $CF
0130       *=    $600
0140       PLA
0150       PLA
0160       STA   FROM+1
0170       PLA
0180       STA   FROM
0190       PLA
0200       STA   TO+1
0210       PLA
0220       STA   TO
0230       PLA
0240       PLA
0250       TAY
0260       INY
0270       LDA   FROM+1
0280       CMP   TO+1
0290       BCC   UP
0300       BNE   DN
0310       LDA   FROM
0320       CMP   TO
0330       BCC   UP
0340       BNE   DN
0350       RTS
0360 UP    DEY
0370       LDA   (FROM),Y
0380       STA   (TO),Y
0390       CPY   #$00
0400       BNE   UP
0410       SEC
0420       LDA   TO
0430       SBC   FROM
0440       TAY
0450       LDA   #$00
0460 LOOP1 DEY
0470       STA   (FROM),Y
0480       CPY   #$00
0490       BNE   LOOP1
0500       RTS
0510 DN    STY   TEMP
0520       LDY   #$00
0530 DN1   LDA   (FROM),Y
0540       STA   (TO),Y
0550       INY
0560       CPY   TEMP
0570       BNE   DN1
0580       SEC
0590       LDA   FROM
0600       SBC   TO
0610       CLC
0620       ADC   TEMP
0630       TAY
0640       LDA   #$00
0650 LOOP2 DEY
0660       STA   (TO),Y
0670       CPY   TEMP
0680       BNE   LOOP2
0690       RTS
```

The above listing is for those of you familiar with assembly language. You may assemble this to any convenient area in memory as it is fully relocatable. For those of you without an assembler editor cartridge we have provided the following routine which can be 'tacked' onto any program which uses PM graphics.

```
10 DATA 104,104,133,204,104,133,203,104
20 DATA 133,206,104,133,205,104,104,168
30 DATA 200,165,204,197,206,144,11,208
40 DATA 34,165,203,197,205,144,3,208
50 DATA 26,96,136,177,203,145,205,192
60 DATA 0,208,247,56,165,205,229,203
70 DATA 168,169,0,136,145,203,192,0
80 DATA 208,249,96,132,207,160,0,177
90 DATA 203,145,205,200,196,207,208,247
100 DATA 56,165,203,229,205,24,101,207
110 DATA 168,169,0,136,145,205,196,207
120 DATA 208,249,96,201,1,240,6,238
130 DATA 253,6,78,255,6,96,172,254
140 DATA 6,177,203,77,255,6,145,203
150 DATA 173,132,2,201,1,208,249,96
```

As you can see the machine code program itself actually resides inside a BASIC string, consequently you need not concern yourself with it's location, as BASIC will take care of this. To use the move routine you must use the 'USR' function. You will need to carry through to this routine the information concerning the block of memory to be moved, i.e. the start of your block, the destination and the number of bytes to be moved. An example of its use is now shown.

```
X=USR(ADR(MOVE$),PMST+PM0OFF+YPOS,PMST+PM0OFF+YPOS+10,18)
```

This one line moves the player at YPOS 10 bytes down the screen. The variables used are as follows:

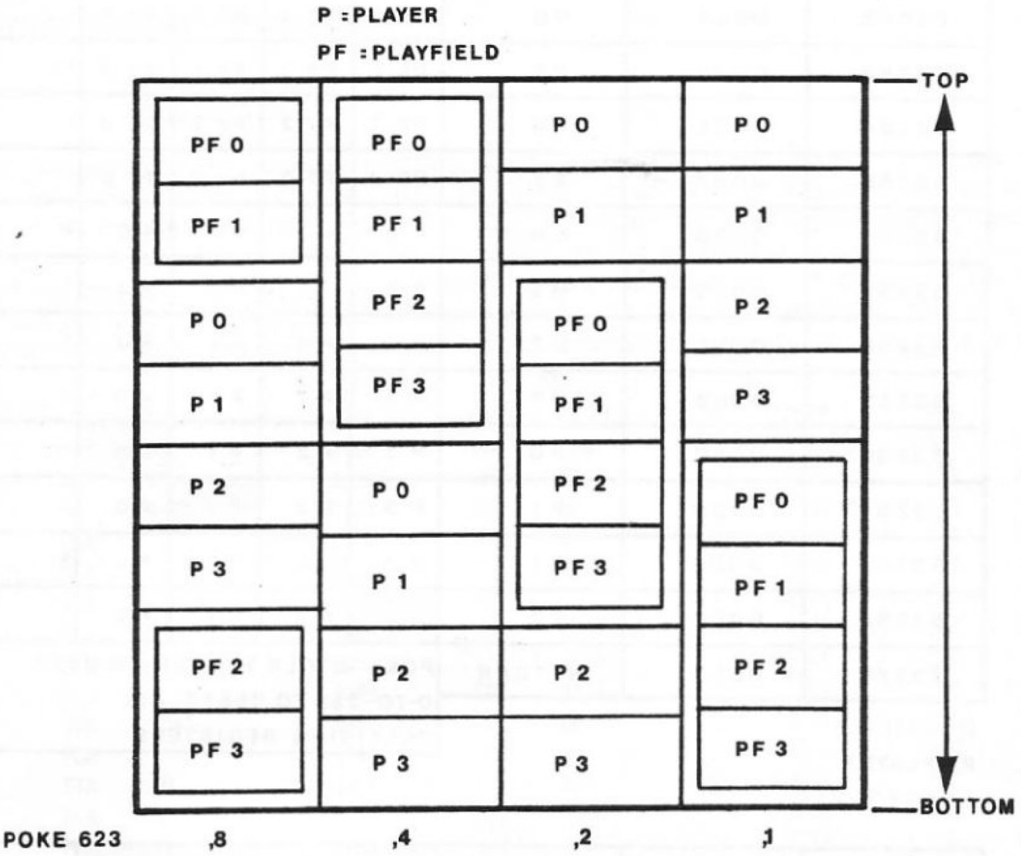
PMST = The absolute start of complete PM table.
PM0OFF = The offset to player0's sub-table from above.
YPOS = The offset inside the player's sub-table i.e. player's vertical position.

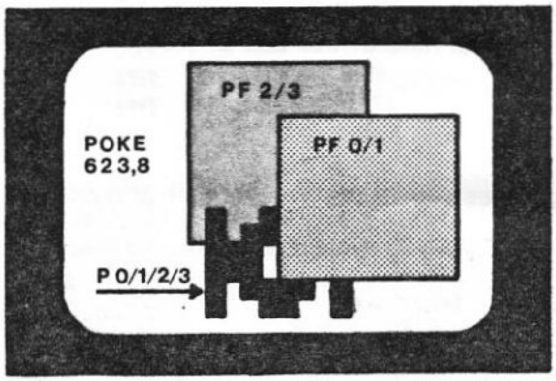
This is of course not the most elegant or fastest method of entering this machine code routine, but once it is understood you may of course experiment with different 'USR' formats. Here is an example player program which utilizes this routine in it's vertical movement. To save space we have compressed the statments with colons.

```
10 DATA 104,104,133,204,104,133,203,104
20 DATA 133,206,104,133,205,104,104,168
30 DATA 200,165,204,197,206,144,11,208
40 DATA 34,165,203,197,205,144,3,208
50 DATA 26,96,136,177,203,145,205,192
60 DATA 0,208,247,56,165,205,229,203
70 DATA 168,169,0,136,145,203,192,0
80 DATA 208,249,96,132,207,160,0,177
90 DATA 203,145,205,200,196,207,208,247
100 DATA 56,165,203,229,205,24,101,207
110 DATA 168,169,0,136,145,205,196,207
120 DATA 208,249,96,201,1,240,6,238
130 DATA 253,6,78,255,6,96,172,254
140 DATA 6,177,203,77,255,6,145,203
150 DATA 173,132,2,201,1,208,249,96
160 DIM MOVE$(120):FOR A=1 TO 120:READ D:MOVE$(A,A)=CHR$(D):NEXT A
170 PMBASE=54279:PM0OFF=1024:SDMCTL=559:GRACTL=53277:HPOSP0=53248:PCOLR0=704:SIZEP0=53256:YPOS
=100:HPOS=120
180 START=(PEEK(106)-16)*256:POKE PMBASE,START/256:FOR A=0 TO 256:POKE START+PM0OFF+A,0:NEXT A
190 FOR A=0 TO 17:READ D:POKE START+PM0OFF+YPOS+A,D:NEXT A
200 DATA 153,153,153,189,189,189,231,231,231,231,231,231,189,189,189,153,153,153
210 POKE SDMCTL,62:POKE GRACTL,3:POKE HPOSP0,HPOS:POKE PCOLR0,47:POKE SIZEP0,1
220 S=STICK(0):IF S=15 THEN 220
230 HPOS=HPOS-(S=11):IF HPOS<0 THEN HPOS=0
240 HPOS=HPOS+(S=7):IF HPOS>255 THEN HPOS=255
250 OLDYPOS=YPOS
260 YPOS=YPOS-(S=14)*2:IF YPOS<0 THEN YPOS=0
270 YPOS=YPOS+(S=13)*2:IF YPOS>255 THEN YPOS=255
280 POKE HPOSP0,HPOS
290 RESULT=USR(ADR(MOVE$),START+PM0OFF+OLDYPOS,START+PM0OFF+YPOS,18)
300 GOTO 220
```

We now briefly mention a few of the finer points involved in player/missile graphics which we intend to discuss in more detail in a future issue. To reduce the consumption of memory when using PM graphics it is possible to go to DOUBLE LINE resolution by 'POKEing' SDMCTL with 46 instead of the usual 62. This means that the players sub-tables are only 128 bytes in length, consequently for a given number of bytes a player will be twice as large in the vertical direction on the T.V. screen. The only sacrifice is of course the vertical resolution.

Another interesting feature of the ATARI's PM graphics is the ability to give players and playfields priority over one another thus enabling the user to produce a 3D effect on a two dimensional screen (just look at BASKETBALL [Atari tm.]). There are four different priority layers which can be used, these are implemented by 'POKEing' a number into GPRIOR (623). Rather than try and explain this in detail we suggest that you experiment with the priorities given below.





Finally, as we conclude this article, we will briefly cover collision detection between players, missiles, and playfields. The GTIA chip provides a set of registers dedicated to automatically provide indication of these collisions. There are sixteen registers, and are allocated as follows:

ADDRESS		FROM → TO	BIT NUMBER				
DEC	HEX	PLAYER-MISSILE	8	4	2	1	
53248	D000	M 0	PF 3	PF 2	PF 1	PF 0	=
53249	D001	M 1	PF 3	PF 2	PF 1	PF 0	=
53250	D002	M 2	PF 3	PF 2	PF 1	PF 0	=
53251	D003	M 3	PF 3	PF 2	PF 1	PF 0	=
53252	D004	P 0	PF 3	PF 2	PF 1	PF 0	=
53253	D005	P 1	PF 3	PF 2	PF 1	PF 0	=
53254	D006	P 2	PF 3	PF 2	PF 1	PF 0	=
53255	D007	P 3	PF 3	PF 2	PF 1	PF 0	=
53256	D008	M 0	P 3	P 2	P 1	P 0	=
53257	D009	M 1	P 3	P 2	P 1	P 0	=
53258	D00A	M 2	P 3	P 2	P 1	P 0	=
53259	D00B	M 3	P 3	P 2	P 1	P 0	=
53260	D00C	P 0	P 3	P 2	P 1	P 0	=
53261	D00D	P 1	P 3	P 2	P 1	P 0	=
53262	D00E	P 2	P 3	P 2	P 1	P 0	=
53263	D00F	P 3	P 3	P 2	P 1	P 0	=
53278	D01E	HITCLR	POKE HITCLR WITH ANY NUMBER 0-TO-255 TO RESET ALL COLLISION REGISTERS				

M = MISSILE
P = PLAYER
PF = PLAYFIELD

53254	D006	P2	PF 3	PF 2	PF 1	PF 0	= 4

We hope you have found this article of use and in the next issue we plan to supply you with more PM utilities.

(We also hope this encourages you to experiment with playersin your program!!!)

USER GROUP SOFTWARE

SOFTWARE LIBRARIAN: Ron Levy.

In issue one, unfortunately, we had no software in our library to publish. Since then we have recieved a selection of different types of program, some utilities, some games and some demo's. We do not have room in this issue to publish all the listings that we would like, but we hope that those we have chosen for this issue will be of interest to you. If you think that the following list of software received to date is somewhat shorter than you might have expected for a user group with over seven hundred members, it is probably because YOU have NOT contributed. Please remember that your programs do not have to be as good as commercially available software, so please send in any of your programs which you feel would be of interest to our members.

When writing your programs please bear in mind that I have to use lines 0 to 9 for identification. Also if you have more than 16K, try and ensure that any player/missile graphics or character set tables are placed in as low a place in memory as possible. It would also be helpful if you could work out how much RAM is required to run the program (16K, 32K or 48K). Finally I would like to thank all those who have contributed to this library and look forward to seeing more programs in the future.

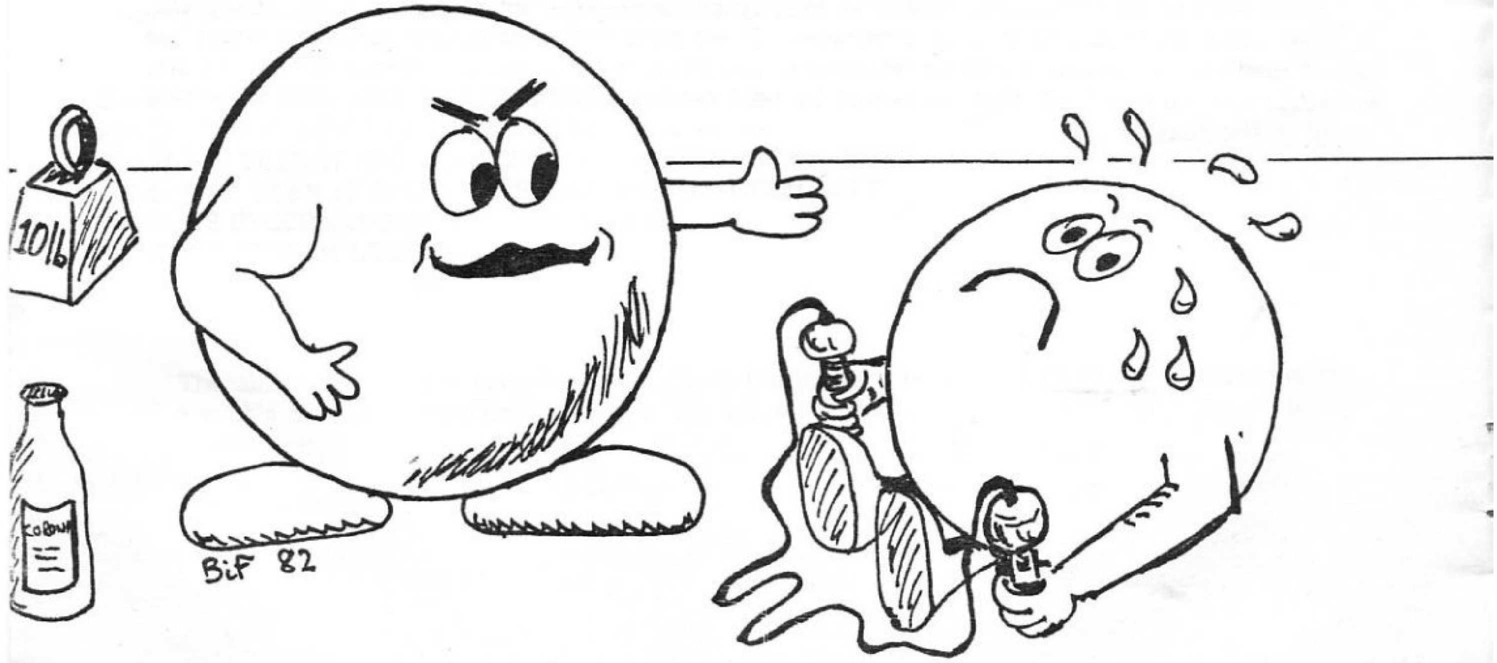
Contributions Format.

On each request received for user software any three of the libraries titles will be returned on your original cassette or disk. You must provide us with at least one of your own programs, your contribution can not be processed in the form of a print out. We recommend that with cassettes you record a second copy on the 'B' side, bearing in mind that if you use less than a C60 cassette it may be not be possible to fit all the programs you request on your tape. We recommend the use of a high quality ferro-type tape such as T.D.K. AD C60's. When submitting programs on disk, please remember to sandwich the disk between two pieces of stiff card to protect the disk from damage, if a disk can be sent please do so as this is more reliable and faster. To those of you who wish to submit software which can only be used with a disk system, it will be entered into the library and carry a note stating that fact. Do remember to include return postage, otherwise we would soon be broke !!!

When writing your programs remember that other people may not find it easy to understand, so print clear instructions in your programs. If we have the unfortunate situation where we cannot load your program, it will be returned to you to be re-recorded and re-submitted. Finally we would like to point out that we cannot be held responsible for loss or damage of materials whilst in the post.

THE PROGRAMS.

SHOOT.	Nigel Haslock.	Switzerland.	Game	Basic.
SPIRAL.	Nigel Haslock.	Switzerland.	Demo	Basic.
STONEVILLE MANOR.	Nigel Haslock.	Switzerland.	Adventure	Basic.
ASSEMBLER.	Chris Rutter.	New Zealand.	Utility	Basic.
COLLISION COURSE.	Jon Beff.	Lincoln.	Game	Basic.
SKYBLITZ.	Chris Barlow.	Leigh-on-sea.	Game	Basic.
FLANFLINGER.	Chris/Graham.	Southend.	Game	Basic.
CURSOR-FLASHER.	Jon Williams.	Littlehampton.	Utility	Basic.
CURSOR-FLASHER.	Jon Williams.	Littlehampton.	Utility	Assembler.
FAST SAVE CASSETTE.	Jon Williams.	Littlehampton.	Utility	Assembler.
FRUIT MACHINE.	Mike Nash.	Avon.	Game	Basic.
SYNTHESISER V1.3	Chris Payne.	Manchester.	Utility	Basic.
CHARACTER GEN V1.0	Martin Walker.	Wiltshire.	Utility	Basic.
MOON LANDER.	D. Mensing.	West Midlands.	Game	Basic.
DIGI-CLOCK.	Ian Lawson-Smith.	Watford.	Demo	Basic.
ATARI TRAIN.	Keith Berry.	Birmingham.	Demo	Basic.
FILER 1.	Chris Payne.	Manchester.	Utility	Basic.
CHARACTER GEN V2.0	I. Scott.	Tyne and Wear.	Utility	Basic.
SNOOPY.	Chris Davies.	Kent.	Demo	Basic.
YAHTZEE.	Steve Calkin.	Pitsea.	Game	Basic.
THE VALLEY.	Steve Calkin.	Pitsea.	Adventure	Basic.
SPHERES.	Peter Patay.	Surrey.	Demo	Basic.
QUADS.	Peter Patay.	Surrey.	Demo	Basic.
COUNTDOWN.	P. Stevens.	Surrey.	Game	Basic.
HANGMAN.	R. Howarth.	Preston.	Game	Basic.
MORSE TUTOR.	Chris Barlow.	Leigh-on-sea.	Tutorial.	Basic.
MORSE KEYBOARD.	Chris Barlow.	Leigh-on-sea.	Utility.	Basic.
GRAPHICS V1.0	Chris Barlow.	Leigh-on-sea.	Demo.	Basic.
MOLE.	Keith Mayhew.	Rayleigh.	Game.	Basic.
PMDEM.	Keith Mayhew.	Rayleigh.	Demo.	Basic.
256 COLOURS.	Keith Mayhew.	Rayleigh.	Demo.	Basic.
COLOUR CORRIDOR.	Keith Mayhew.	Rayleigh.	Demo.	Basic.
MEMSCROLL.	Keith Mayhew.	Rayleigh.	Demo.	Basic.
PIG IN MIDDLE.	Keith Berry.	Birmingham.	Game.	Basic.
PLAYER DESIGNER.	Keith Berry.	Birmingham.	Utility.	Basic.



I'M Sorry, You've failed the 'Fizzical'
You'll have to go to ATARI as a 'PAC-MAN'

THE PROGRAMS...

We present these programs to you and hope you have many hours of fun typing them in. We will briefly describe each program and any peculiarities such as control codes, here is a list of the symbols we will use to describe these codes.
Any key actually on the keyboard is shown in square brackets e.g.[esc].

When a [ctrl] is shown it is to be held down while entering the subsequent character(s).

[inv] means press the ATARI LOGO key (inverse).

Any number of spaces to be inserted will have [sp 16] for 16 spaces.

Any underlined characters on the print out are in inverse characters.

* "3" * means insert between the third set of quotes on this line any information given.

(......*6) means how many times the instructions are to be entered.

SKYBLITZ.

```
On line 325   * "1" * [esc] [ctrl =] [sp 3]
On line 570   * "1" * [esc] [tab] [esc] [ctrl -] ([esc] [ctrl *] *6)
On line 750   * "1" * [esc] [tab] [sp 4]
```

Description... BASIC game using player missile animation. To drop a bomb on the city below press any key.

COLLISION COURSE.

```
On line 30    * "1" * [esc] [ctrl <] ([esc] [tab] *2) [esc] [ctrl =]
On line 80    * "1" * [esc] [ctrl 2] ([esc] [tab] *2)
On line 1202  * "1" * [esc] [ctrl <] [esc] [tab] [sp 4] [esc] [ctrl =]
On line 1205  * "1" * [esc] [ctrl <] [esc] [tab] [sp 5] [esc] [ctrl =]
On line 1230  * "1" * [esc] [ctrl <] [esc] [ctrl =] * "2" * [esc] [tab] [sp 4]
On line 1600  * "1" * [inv] ([ctrl U] *36) [inv] [sp 2] [inv] ([ctrl U] *36) [inv]
On line 2160  * "1" * [inv] ([ctrl U] *36) [inv] [sp 2] [inv] ([ctrl U] *36) [inv]
On line 2200  * "1" * [esc] [tab] [sp 7]
```

Description... BASIC game using joysticks 1 and 2. Chase opponent and try outmanoeuvering him.

FRUIT MACHINE.

```
On line 5     * "1" * [sp 2] ([ctrl ,] [sp 1] *3) [sp 1]
On line 5     * "2" * [sp 2] ([ctrl ;] [sp 1] *3) [sp 1]
On line 5     * "3" * [sp 2] ([ctrl .] [sp 1] *3) [sp 1]
On line 10003 * "2" * [sp 2] [ctrl p] ([sp 1] - *2) [sp 2]
On line 10003 * "3" * [sp 2] ([ctrl P] [sp 1] *2) - [sp 2]
On line 10003 * "4" * [sp 2] ([ctrl p] [sp 1] *3) [sp 1]
```

Description... BASIC game, random chance with graphical representation.

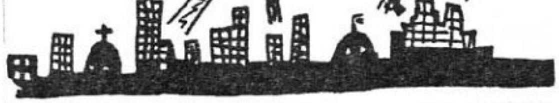
FLASHING CURSOR.

Description... Assembler utility allowing different modes for the cursor and inverse text.

SNOOPY.

Description... BASIC demo, draws 'Snoopy Dog' on graphics 8 quickly.

```
10 SCORE=0:HISCORE=500:GRAPHICS 2+16:REM **SKYBLITZ BY C.S.BARLOW **
15 POKE 77,0:EX=INT(RND(0)*16):SETCOLOR 2,EX,10:POSITION 5,5:? #6;"PRESS START"
20 IF PEEK(53279)<>6 THEN 15
25 GOSUB 300
30 X=X+0.5:POKE 704,X:POKE 53248,X:SOUND 0,X/4,8,10
40 IF PEEK(764)<255 THEN GOSUB 500
50 IF PEEK(53252)>0 THEN 1000
170 IF X>217 THEN X=27:SOUND 0,0,0,0:GOTO 190
180 GOTO 30
190 FOR MY=0 TO 3:FOR I=8 TO 0 STEP -1:POKE PMBASE+512+Y2+I,PEEK(PMBASE+511+Y2+I)
195 POKE PMBASE+384+Y2+4+I,PEEK(PMBASE+383+Y2+4+I):NEXT I:Y2=Y2+1:NEXT MY
200 IF Y2>90 THEN 750
210 GOTO 30
300 X=27:Y2=5:X1=19:GRAPHICS 3:SETCOLOR 1,0,0:SETCOLOR 2,12,8:SETCOLOR 4,7,0:SOUND 1,100,16,4:COLOR 1:RESTORE
305 FOR Y=5 TO 35
310 X2=INT(RND(0)*19):IF X2<9 THEN 310
320 PLOT Y,X2:DRAWTO Y,X1:NEXT Y
325 POKE 752,1:PRINT "    SCORE  ";SCORE," HIGH SCORE  ";HISCORE
330 A=PEEK(106)-8:POKE 54279,A:PMBASE=256*A:POKE 53248,Y2:POKE 704,58
335 POKE 53256,1:POKE 53278,0:POKE 53277,3:XM=0
340 FOR I=PMBASE+512 TO PMBASE+640:POKE I,0:NEXT I
360 FOR I=PMBASE+512+Y2 TO PMBASE+518+Y2:READ A:POKE I,A:NEXT I
370 DATA 66,90,126,219,219,126,153
380 FOR I=PMBASE+384 TO PMBASE+511:POKE I,0:NEXT I
390 POKE 53252,0:POKE 559,46:RETURN
500 POKE 53252,X+7:Y3=Y2+7:XM=INT(X/4)-10:SD=0
505 FOR DR=Y3 TO 94:YM=DR/4-4
520 POKE PMBASE+384+Y3+1,3:POKE PMBASE+384+Y3,0
530 X=X+1:POKE 704,X:SOUND 0,X/4,8,10:SOUND 2,DR,10,15
540 POKE 53248,X:IF X>217 THEN X=217
550 IF PEEK(53252)>0 THEN 700
570 IF PEEK(53248)>0 THEN COLOR 0:PLOT XM,YM:SOUND 3,25,10,15:SD=SD+1:SCORE=SCORE+1:PRINT "";SCORE
680 SOUND 3,0,0,0:IF SD>14 THEN 700
685 Y3=Y3+1:NEXT DR:POKE PMBASE+384+Y3,0
700 SOUND 2,0,0,0:POKE PMBASE+384+Y3+1,0:POKE 764,255:POKE 53252,0:POKE 53278,0:RETURN
750 SOUND 1,0,0,0:PRINT "    BONUS POINTS 100 ":SCORE=SCORE+100
760 FOR S=1 TO 25:SOUND 0,20,10,10:FOR T=0 TO 10:NEXT T:SOUND 0,0,0,0:FOR T=0 TO 10:NEXT T:NEXT S:GOTO 25
1000 POKE 53277,0:SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0:SOUND 3,0,0,0:VOL=15
1010 FOR B=10 TO 100 STEP 0.6:VOL=VOL-0.1:SETCOLOR 4,3,10:IF VOL<0 THEN VOL=0
1020 SETCOLOR 4,0,0:POKE 559,0:SOUND 0,B,8,VOL:SOUND 1,B,16,VOL:SOUND 2,B,4,VOL:POKE 559,46
1030 NEXT B
1040 GRAPHICS 2+16:SETCOLOR 4,7,0:SETCOLOR 0,3,5
1050 POSITION 2,2:PRINT #6;"HIGH SCORE ";HISCORE
1060 POSITION 7,4:PRINT #6;"SCORE ";SCORE
1070 IF SCORE>HISCORE THEN GOSUB 1100
1080 POSITION 5,6:PRINT #6;"PRESS START"
1090 POKE 77,0:EX=INT(RND(0)*16):SETCOLOR 2,EX,10:IF PEEK(53279)<>6 THEN 1090
1095 SCORE=0:GOTO 25
1100 HISCORE=SCORE:POSITION 13,2:PRINT #6;HISCORE
1110 FOR XX=0 TO 10:FOR S=100 TO 10 STEP -8:SOUND 0,S,10,10:SOUND 1,S+3,10,10:NEXT S:NEXT XX
1120 SOUND 0,0,0,0:SOUND 1,0,0,0:RETURN
```



```
0 REM COLLIDE.BAS - COLLISION COURSE
1 REM Jon Beff
2 REM 7 School Lane,                         Thorpe-On-The-Hill,                         lincoln.  LN6 9BN
3 GOSUB 10000:REM PROG. INIT. MOVED
4 POSITION 23,9:? " -by jon beff":POSITION 6,17:? "HOW MANY ARE PLAYING (1/2)? ";
5 SETCOLOR 2,2.4:SETCOLOR 4,2,4:SETCOLOR 0,2,0:SETCOLOR 1,2,10:SETCOLOR 3,4,6:POKE L+6,7:POKE L+10,6
6 FOR I=121 TO 29 STEP -8:SOUND 0.I,14,4:FOR D=1 TO 20:NEXT D:SOUND 0,0,0,0:FOR D=0 TO 10:NEXT D:NEXT I
7 ATARI=PEEK(764):IF ATARI<>31 AND ATARI<>30 THEN 6
8 ? CHR$(31-ATARI+49):POKE 764.243:POSITION 11,20:? "xxx press START xxx"
9 FOR II=14 TO 0 STEP -0.05:SOUND 0.63,12,II:NEXT II
10 IF PEEK(53279)<>6 THEN 10
14 GRAPHICS 5
15 SETCOLOR 4,12,6
16 SETCOLOR 2,9.2:SETCOLOR 0,0,0:SETCOLOR 1,0,14
19 COLOR 3
20 PLOT 0,39:DRAWTO 0,0:DRAWTO 79,0:DRAWTO 79,40:DRAWTO 0,40:PLOT 78,0:DRAWTO 78,39
30 PRINT " GET READY";
50 FOR W=1 TO 1000:NEXT W
60 XX=56:YY=20:B=11
65 V=30
66 IF ATARI=30 THEN 70
67 V=15
70 XXX=22:YYY=20:B1=7
75 COLOR 1:PLOT XX,YY:COLOR 2:PLOT XXX,YYY
77 SOUND 0,102,10,14
78 FOR I=1 TO 4:FOR W=1 TO 20:NEXT W:SOUND 0,0,0,0:FOR W=1 TO 20:NEXT W:SOUND 0,102,10,14:NEXT I
79 FOR W=1 TO 100:NEXT W:SOUND 0,0,0,0
80 POKE 752,1:PRINT ">>> GO <<<"
90 FOR W=1 TO 140:NEXT W
100 A=STICK(1)
101 D=4
102 X=XX:Y=YY
104 V=V*0.995
105 E=INT(V/2)*2
110 IF ATARI=30 THEN 140
111 H=0
112 R=INT(RND(0)*4)+1
113 IF RND(0)>RND(0)/10 THEN A=B:GOTO 140
116 A=B(R)
140 IF A=14 THEN Y1=Y-2:X1=X:GOTO 160
142 IF A=13 THEN Y1=Y+2:X1=X:GOTO 160
144 IF A=7 THEN X1=X+2:Y1=Y:GOTO 160
146 IF A=11 THEN X1=X-2:Y1=Y:GOTO 160
150 A=B:GOTO 140
160 LOCATE X1,Y1,T
170 IF T=0 THEN 196
175 IF ATARI=30 THEN 194
180 R=1+R-INT(R/4)*4
183 IF H=4 THEN 1200
185 H=H+1:GOTO 116
194 X=X1:Y=Y1:GOTO 1200
196 X=X1:Y=Y1:COLOR 1:PLOT X,Y
197 B=A
198 SOUND 0,98,10,8:FOR S=1 TO E/2:NEXT S:SOUND 0,0,0,0:FOR W=1 TO E-H*10:NEXT W
199 YY=Y:XX=X
200 X=XXX:Y=YYY
203 D=3
205 A=STICK(0)
208 V=V*0.995
209 E=INT(V/2)*2
```

```
210 IF A=14 THEN Y=Y-2:GOTO 290
220 IF A=13 THEN Y=Y+2:GOTO 290
230 IF A=7 THEN X=X+2:GOTO 290
240 IF A=11 THEN X=X-2:GOTO 290
250 A=B1:GOTO 210
290 LOCATE X,Y,T
292 IF T<>0 THEN 1200
294 COLOR 2:PLOT X,Y
295 B1=A
298 SOUND 0,102,10,8:FOR S=1 TO E/2:NEXT S:SOUND 0,0,0,0:FOR W=1 TO E:NEXT W
299 XXX=X:YYY=Y
300 GOTO 100
1200 SOUND 0,243,10,12
1202 IF X=XXX AND Y=YYY THEN PRINT "     BOTH PLAYERS LOSE":S(1)=S(1)+1:S(2)=S(2)+1:GOTO 1208
1205 PRINT "      PLAYER ";D-2;" LOSES."
1206 S(5-D)=S(5-D)+1
1208 FOR I=1 TO 6:SETCOLOR 0,0,14:SETCOLOR 1,0,0:FOR W=1 TO 20:NEXT W:SETCOLOR 0,0,0:SETCOLOR 1,0,14:FOR W=1 TO 20:NEXT W:NEXT I
1210 SOUND 0,0,0,0
1220 FOR W=1 TO 400:NEXT W
1230 PRINT " SCORE:   PLAYER 1-";S(1);? "    PLAYER 2-";S(2)
1250 PRINT " AGAIN (Y/N)?";
1260 POKE 764,0
1270 A=PEEK(764):IF A<>43 AND A<>35 THEN 1270
1280 IF A=43 THEN 14
1400 GRAPHICS 0:SETCOLOR 4,2,4:SETCOLOR 2,2,4
1500 PRINT :PRINT
1600 PRINT "  ":PRINT
2000 PRINT "FINAL  SCORE: PLAYER-1 PLAYER-2/COMP":PRINT
2100 PRINT "                 ";S(1);"           ";S(2)
2150 PRINT
2160 PRINT "  ":PRINT :PRINT
2200 PRINT :PRINT :PRINT "      BYE FOR NOW!":PRINT :PRINT :PRINT
2955 FOR II=14 TO 0 STEP -0.05:SOUND 0,63,12,II:NEXT II
3000 POKE 764,255
5000 REM COLLISION COURSE-BY JON BEFF
5030 REM THIS IS A GAME FOR ONE PLAYER       AGAINST THE COMPUTER, OR FOR       TWO PLAYERS. EACH PLAYER
5040 REM CONTROLS THE DIRECTION OF A         MOVING BLOB THAT LEAVES A          TRAIL TO SHOW WHERE IT HAS
5050 REM BEEN. THE AIM IS TO AVOID           HITTING ONES OWN TRAIL,THAT        OF YOUR OPPONENT, OR ANY OF
5060 REM THE SURROUNDING WALLS. THE          FIRST PLAYER TO CRASH GIVES        A POINT TO THE OTHER PLAYER.
5065 REM IF BOTH PLAYERS CRASH THEN          IT'S A DRAW AND EACH GETS A        POINT.
5070 REM THE GAME SPEEDS UP, AND SO          BECOMES MORE DIFFICULT AS IT       GOES ON.
5075 REM LEFT  JOYSTICK=WHITE BLOB           RIGHT JOYSTICK=BLACK BLOB.
5080 REM 2-PLAYERS:USE JOYSTICKS 1+2
5090 REM 1-PLAYER :USE JOYSTICK  1.
6000 END
10000 REM THIS SECTION MOVED TO ALLOW         CLASSIFICATION  - Ron      *
10001 DIM B(4),S(2)
10002 B(1)=11:B(2)=7:B(3)=14:B(4)=13:S(1)=0:S(2)=(0)
10003 GRAPHICS 0:POKE 752,1:L=6+PEEK(741)+256*PEEK(742):POSITION 2,6:? "CoLlIsIoN cOuRsE"
10010 RETURN
```

```
0 REM FRUIT1.BAS - FRUIT MACHINE        *
1 REM Mike Nash
2 REM 6, The Leaze,                         Radstock,                          Nr. Bath,
3 REM *    Avon.                        *
4 GOSUB 10000:REM First part shifted       to give me room !!  - Ron        *
5 DIM A(3,20):? "      8":? "          10":? "          14":? "3 ARROWS 16":? "3  PLUMS 20"
9 POKE 710,98:POKE 712,98:POKE 752,1:? "  7 7 7  30":FOR I=1 TO 6000:NEXT I
10 GRAPHICS 7:C170=170:C85=85:C40=40:U=7000:C0=0
15 SETCOLOR 1,C0,15:SETCOLOR 2,C0,C0:SETCOLOR 0,3,6:SETCOLOR 4,7,4
20 L=PEEK(88)+PEEK(89)*256
25 Z=L+2013:POKE Z,C170:POKE Z+1,C170:POKE Z+2,C170:POKE Z+3,C170:POKE Z+10,C170:POKE Z+11,170:POKE Z+12,170
26 POKE Z+13,C170:POKE 752,1
110 FOR I=1 TO 3:FOR J=1 TO 20
120 READ X:A(I,J)=X:NEXT J:NEXT I
150 DATA 7,1,5,2,1,3,6,2,4,1
160 DATA 2,5,1,2,6,1,2,1,4,1
170 DATA 7,6,4,2,3,6,4,1,5,4
180 DATA 6,1,3,5,2,4,3,2,4,3
190 DATA 3,4,6,2,4,3,2,4,6,3
200 DATA 7,5,3,1,4,3,1,4,3,2
210 CR=0:H1=0:H2=0:H3=0
400 CHAR=255
450 GOSUB 7100
451 GOSUB 7200
455 IF CHAR<>33 THEN 451
460 HFLAG=0:GOSUB 4900
500 CR=CR-1
505 IF H1=1 THEN 540
510 B1=INT(RND(0)*20)+1:GOSUB 7500
540 IF H2=1 THEN 570
545 B2=INT(RND(0)*20)+1:GOSUB 7600
570 IF H3=1 THEN 615
580 B3=INT(RND(0)*20)+1:GOSUB 7700
615 H1=0:H2=0:H3=0:W=0:NFLAG=0
620 HFLAG=0:GOSUB 4900
630 REM CHECK IF WINNER
635 IF A(1,B1)=3 THEN W=1
640 IF A(1,B1)=A(2,B2) AND A(2,B2)=A(3,B3) THEN W=1
645 IF W=0 THEN 750
650 REM WINNER
655 IF A(1,B1)=3 AND A(2,B2)=3 AND A(3,B3)=3 THEN W=6:GOTO 700
660 IF A(1,B1)=3 AND A(2,B2)=3 THEN W=4:GOTO 700
665 IF A(1,B1)=3 THEN W=2:GOTO 700
670 IF A(1,B1)=2 THEN W=8
675 IF A(1,B1)=4 THEN W=10
680 IF A(1,B1)=1 THEN W=14
685 IF A(1,B1)=6 THEN W=16
690 IF A(1,B1)=5 THEN W=20
695 IF A(1,B1)=7 THEN W=30
700 CR=CR+W:FOR I=1 TO 3:? "WINNER PAYS ";W:NEXT I:GOSUB 8000:GOSUB 7100
710 GOTO 450
750 REM CHECK NFLAG
755 IF NFLAG=0 THEN 765
756 FOR I=1 TO 3:? "YOU BLEW IT !!!!":NEXT I:SOUND 0,200,10,10:SOUND 1,220,12,6
757 FOR I=1 TO 250:NEXT I:SOUND 1,0,0,0:SOUND 0,0,0,0
760 NFLAG=C0:GOSUB 7100:GOTO 450
765 REM CHECK HOLD
770 H=INT(RND(0)*4)
775 IF H<>1 THEN 900
780 FOR I=1 TO 6:SOUND 1,C40,10,6
785 ? "  *** HOLD AVAILABLE ***":SOUND 1,0,0,0:NEXT I
```

```
790 GOSUB 7200:IF CHAR=255 THEN 790
795 IF CHAR=33 THEN 500
799 HFLAG=1
800 IF CHAR<>31 THEN 815
805 H1=1:X=L+2490:GOSUB 4900
810 GOTO 790
815 IF CHAR<>30 THEN 830
820 H2=1:X=L+2500:GOSUB 4900
825 GOTO 790
830 IF CHAR<>26 THEN 845
835 H3=1:X=L+2510:GOSUB 4900
840 GOTO 790
845 IF CHAR<>24 THEN 790
850 H1=C0:H2=C0:H3=C0:HFLAG=C0:X=L+2490:GOSUB 4900
855 GOTO 790
900 REM NUDGE SECTION
905 N=INT(RND(0)*8):NFLAG=1:NN=INT(RND(0)*10)+5
910 IF N<>1 THEN 450
915 FOR I=1 TO 3
920 ? "xxx  NUDGE NOW  xxx"
925 NEXT I
927 SOUND 1,50,4,10
930 GOSUB 7200
935 NN=NN-1
940 IF CHAR=255 THEN FOR I=1 TO 150:NEXT I
945 IF NN=0 THEN ? :? :? :? :? :? :SOUND 1,C0,C0,C0:GOTO 630
950 IF CHAR<>31 THEN 955
951 B1=B1-1:IF B1<1 THEN B1=B1+20
952 GOSUB 7500:GOTO 965
955 IF CHAR<>30 THEN 960
956 B2=B2-1:IF B2<1 THEN B2=B2+20
957 GOSUB 7600:GOTO 965
960 IF CHAR<>26 THEN 965
961 B3=B3-1:IF B3<1 THEN B3=B3+20
962 GOSUB 7700
965 NFLAG=1:GOTO 930
4900 REM PRINT "H"
4905 IF HFLAG=0 THEN X=L+2490:GOTO 4950
4910 POKE X,130:POKE X+40,130:POKE X+80,170:POKE X+120,130:POKE X+160,130
4915 RETURN
4950 FOR Z=0 TO 20 STEP 10
4955 Y=X+Z
4960 FOR I=0 TO 320 STEP 40:POKE Y+I,0:NEXT I
4965 NEXT Z:RETURN
5000 REM PRINT FRUIT 1
5005 Z=X+80:POKE Z+1,169
5010 Z=Z+C40:POKE Z+1,165:POKE Z+2,106
5015 Z=Z+C40:POKE Z+1,149:POKE Z+2,90
5020 Z=Z+C40:POKE Z+1,C85:POKE Z+2,86
5025 Z=Z+C40:POKE Z,169:POKE Z+1,C85:POKE Z+2,C85
5030 Z=Z+C40:POKE Z,165:POKE Z+1,C85:POKE Z+2,C85:POKE Z+3,106
5035 Z=Z+C40:POKE Z,169:POKE Z+1,C85:POKE Z+2,C85
5040 Z=Z+C40:POKE Z+1,C85:POKE Z+2,86
5045 Z=Z+C40:POKE Z+1,149:POKE Z+2,90
5050 Z=Z+C40:POKE Z+1,165:POKE Z+2,106
5055 Z=Z+C40:POKE Z+1,169
5060 RETURN
5100 REM PRINT FRUIT 2
5105 Z=X+160:POKE Z+1,90:POKE Z+2,150
5110 Z=Z+40:POKE Z,169:POKE Z+1,86:POKE Z+2,C85
```

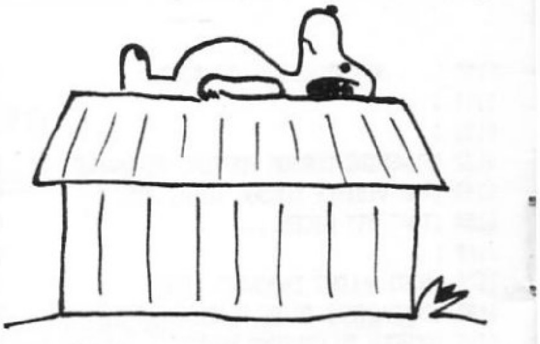
```
5115 Z=Z+40:POKE Z,169:POKE Z+1,C85:POKE Z+2,C85
5120 Z=Z+40:POKE Z,169:POKE Z+1,C85:POKE Z+2,C85
5125 Z=Z+40:POKE Z+1,C85:POKE Z+2,86
5130 Z=Z+40:POKE Z+1,149:POKE Z+2,90
5135 Z=Z+40:POKE Z+1,165:POKE Z+2,106
5140 Z=Z+40:POKE Z+1,169
5145 RETURN
5200 REM PRINT FRUIT 3
5205 Z=X+80:POKE Z+2,234
5210 Z=Z+40:POKE Z+1,171:POKE Z+2,250
5215 Z=Z+40:POKE Z+1,175:POKE Z+2,254
5220 Z=Z+40:POKE Z+1,171:POKE Z+2,250
5225 Z=Z+40:POKE Z+1,186:POKE Z+2,235
5230 Z=Z+40:POKE Z+1,254:POKE Z+2,239:POKE Z+3,234
5235 Z=Z+40:POKE Z,171:POKE Z+1,255:POKE Z+2,255:POKE Z+3,250
5240 Z=Z+40:POKE Z,171:POKE Z+1,255:POKE Z+2,255:POKE Z+3,250
5245 Z=Z+40:POKE Z+1,254:POKE Z+2,239:POKE Z+3,234
5250 Z=Z+40:POKE Z+1,186:POKE Z+2,235
5255 Z=Z+40:POKE Z+1,171:POKE Z+2,250
5260 Z=Z+40:POKE Z+1,175:POKE Z+2,254
5265 RETURN
5300 REM PRINT FRUIT 4
5305 Z=X+120:POKE Z+2,234
5310 Z=Z+C40:POKE Z+1,171:POKE Z+2,250
5315 Z=Z+C40:POKE Z+1,175:POKE Z+2,254
5320 Z=Z+C40:POKE Z+1,191:POKE Z+2,255
5322 FOR I=1 TO 3
5325 Z=Z+C40:POKE Z+1,255:POKE Z+2,255:POKE Z+3,234:NEXT I
5335 Z=Z+C40:POKE Z+1,190:POKE Z+2,239
5340 Z=Z+40:POKE Z+1,171:POKE Z+2,250
5345 Z=Z+C40:POKE Z+1,175:POKE Z+2,254
5350 RETURN
5400 REM PRINT FRUIT 5
5405 Z=X+200:POKE Z,170:POKE Z+1,0:POKE Z+2,42
5410 Z=Z+C40:POKE Z,168:POKE Z+1,0:POKE Z+2,10
5415 Z=Z+C40:POKE Z,160:POKE Z+1,0:POKE Z+2,2
5420 Z=Z+C40:POKE Z,160:POKE Z+1,0:POKE Z+2,63:POKE Z+3,234
5425 Z=Z+C40:POKE Z,160:POKE Z+1,51:POKE Z+2,2:POKE Z+3,234
5430 Z=Z+C40:POKE Z,160:POKE Z+1,12:POKE Z+2,10
5435 Z=Z+C40:POKE Z,C170:POKE Z+1,0:POKE Z+2,42
5440 RETURN
5500 REM PRINT FRUIT 6
5505 Z=X+120:POKE Z+1,168:POKE Z+2,42
5510 Z=Z+C40:POKE Z+1,160:POKE Z+2,10
5515 Z=Z+C40:POKE Z+1,128:POKE Z+2,2
5520 Z=Z+C40:POKE Z+1,8:POKE Z+2,32
5525 Z=Z+C40:POKE Z,168:POKE Z+1,C40:POKE Z+2,C40:POKE Z+3,42
5530 FOR I=1 TO 5
5535 Z=Z+C40:POKE Z+1,168:POKE Z+2,42
5540 NEXT I
5545 RETURN
5600 REM PRINT FRUIT 7
5605 Z=X+80:POKE Z+1,C85:POKE Z+2,C85
5610 Z=Z+C40:POKE Z+1,C85:POKE Z+2,C85
5615 Z=Z+C40:POKE Z+1,191:POKE Z+2,245
5620 Z=Z+C40:POKE Z+2,165
5625 Z=Z+C40:POKE Z+2,151
5630 Z=Z+C40:POKE Z+2,151
5635 Z=Z+C40:POKE Z+2,94
5640 Z=Z+C40:POKE Z+2,94
5645 Z=Z+C40:POKE Z+1,169:POKE Z+2,122
5650 Z=Z+C40:POKE Z+1,169:POKE Z+2,122
5655 Z=Z+C40:POKE Z+1,165:POKE Z+2,234
5660 Z=Z+C40:POKE Z+1,165:POKE Z+2,234
5665 RETURN
5700 REM PRINT WHITE SQUARE
```

```
5705 FOR I=0 TO 640 STEP 40
5710 Z=X+I:POKE Z,C170:POKE Z+1,C170:POKE Z+2,C170:POKE Z+3,C170
5715 NEXT I
5720 RETURN
7000 REM PRINT FRUIT
7005 GOSUB 5700
7010 I=5000+(A(R,B)*100)-100
7015 GOSUB I
7020 RETURN
7100 REM DISPLAY CREDITS
7105 IF CR<0 THEN 7130
7110 IF CR>0 THEN 7145
7115 FOR I=1 TO 6
7120 ? "***  ALL SQUARE  ***":NEXT I
7125 RETURN
7130 FOR I=1 TO 6
7135 ? "You are losing ";-CR;" units":NEXT I
7140 RETURN
7145 FOR I=1 TO 6
7150 ? "YOU ARE WINNING ";CR;" UNITS":NEXT I
7155 RETURN
7200 REM GET KEY
7201 CHAR=255:POKE 764,255:FOR I=1 TO 50:NEXT I
7205 IF PEEK(764)=255 THEN RETURN
7210 POKE 53279,0
7215 CHAR=PEEK(764)
7217 FOR I=1 TO 10:NEXT I
7220 POKE 53279,255
7225 RETURN
7500 REM PRINT REEL 1
7505 R=1:X=L+80+8:B=B1-2
7510 IF B<1 THEN B=B+20
7515 GOSUB U:SOUND 2,120,10,5
7520 X=X+800:B=B1-1
7525 IF B<1 THEN B=B+20
7527 SOUND 2,0,0,0
7530 GOSUB U
7535 X=X+800:B=B1:GOSUB U:RETURN
7600 REM PRINT REEL 2
7605 R=2:X=L+80+18:B=B2-2
7610 IF B<1 THEN B=B+20
7615 GOSUB U:SOUND 2,80,10,5
7620 B=B2-1:X=X+800
7625 IF B<1 THEN B=B+20
7627 SOUND 2,0,0,0
7630 GOSUB U
7635 B=B2:X=X+800:GOSUB U:RETURN
7700 REM PRINT REEL 3
7705 R=3:X=L+80+28:B=B3-2
7710 IF B<1 THEN B=B+20
7715 GOSUB U:SOUND 2,40,10,5
7720 B=B3-1:X=X+800:SOUND 2,0,0,0
7725 IF B<1 THEN B=B+20
7730 GOSUB U
7735 B=B3:X=X+800:GOSUB U:RETURN
8000 REM NOTES WHEN WINNER
8001 FOR I=1 TO 255:SOUND 0,255-I,10,6
8002 SOUND 1,I,10,6:NEXT I
8003 SOUND 0,0,0,0:SOUND 1,0,0,0:RETURN
10000 REM This Section Has Been Moved *
10001 GRAPHICS 18:POKE 712,198:POSITION 7,3:? #6;"FrUiT":POSITION 6,5:? #6;"mAcHiNe":POSITION 0,8:? #6;"BY MIKE NASH"
10002 FOR I=1 TO 300:NEXT I
10003 GRAPHICS 0:? :? "  --- PAYOUTS ---":? :? "   - -  2":? "    -  4":? "      6"
10010 RETURN
32767 END
```

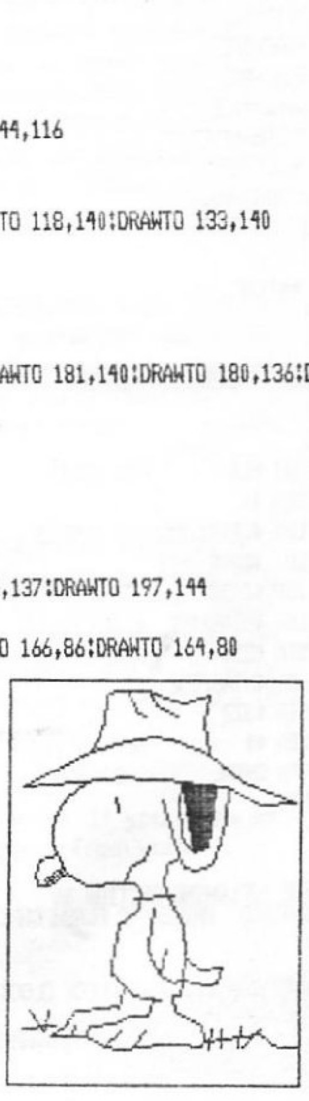
```
0100 ;    JON WILLIAMS. 23/03/82.
0110 ;
0120 ;
0130 ;FLASHING CURSOR ROUTINE TO ENABLE
0140 ;THE VISIBLE CURSOR TO BE JUST
0150 ;THAT BIT NICER....
0160 ;
0170 ;ALSO ALLOWS INVERTED VIDEO
0180 ;CHARACTERS TO BE FLASHED ON THE
0190 ;SCREEN IN VARIOUS MODES.
0200 ;
0210 ;CNTRL ($CE) CONTROLS THE MODE.
0220 ;
0230 ;  0=NEW FLASHING CURSOR.
0240 ;  1=BLINKING CHARACTERS
0250 ;  2=INVERSE TO NORMAL FLASHING.
0260 ;  3=NORMAL TO 'SOLID WHITE'.
0270 ;  4=UPSIDE DOWN TO NORMAL!!
0280 ;$40=BLINKING INVERSE CHARS.
0290 ;$80=ALL OFF. I.E.NORMAL CURSOR.
0300 ;
0310 ;
0320 ;
0330 RTCLOK=$12
0340 DSTAT=$4C
0350 ROWCRS=$54
0360 COLCRS=$55
0370 OLDADR=$5E
0380 LOGCOL=$63
0390 BUFCNT=$6B
0400 BUFSTR=$6C
0410 TEST=$CB
0420 FLAG=$CC
0430 TEMP=$CD
0440 CNTRL=$CE
0450 CRSINH=$2F0
0460 CHACT=$2F3
0470 ATACHR=$2FB
0480 ;
0490 ;
0500  *=$600
0510 ;
0520 ;
0530 ;INITIALISATION ROUTINE...
0540 ;
0550 ;
0560 INIT PLA          POP STACK
0570  LDA #6
0580  LDX #ENTRY/256 SET VVBLKI
0590  LDY #ENTRY&$FF
0600  JSR $E45C
0610  LDA #HTAB&$FF  ALTER EDITOR
0620  STA $321       HANDLER TABLE
0630  LDA #HTAB/256
0640  STA $322
0650  LDA #0         SET FOR FLASHING
0660  STA CNTRL        CURSOR
0670  RTS
0680 ;
0690 ;
0700 ;THE FOLLOWING SECTION IS
0710 ;INTERUPT DRIVEN TO FLASH CURSOR
0720 ;
0730 ;
0740 ENTRY LDA RTCLOK+2 CHECK CLOCK
0750 ;
```

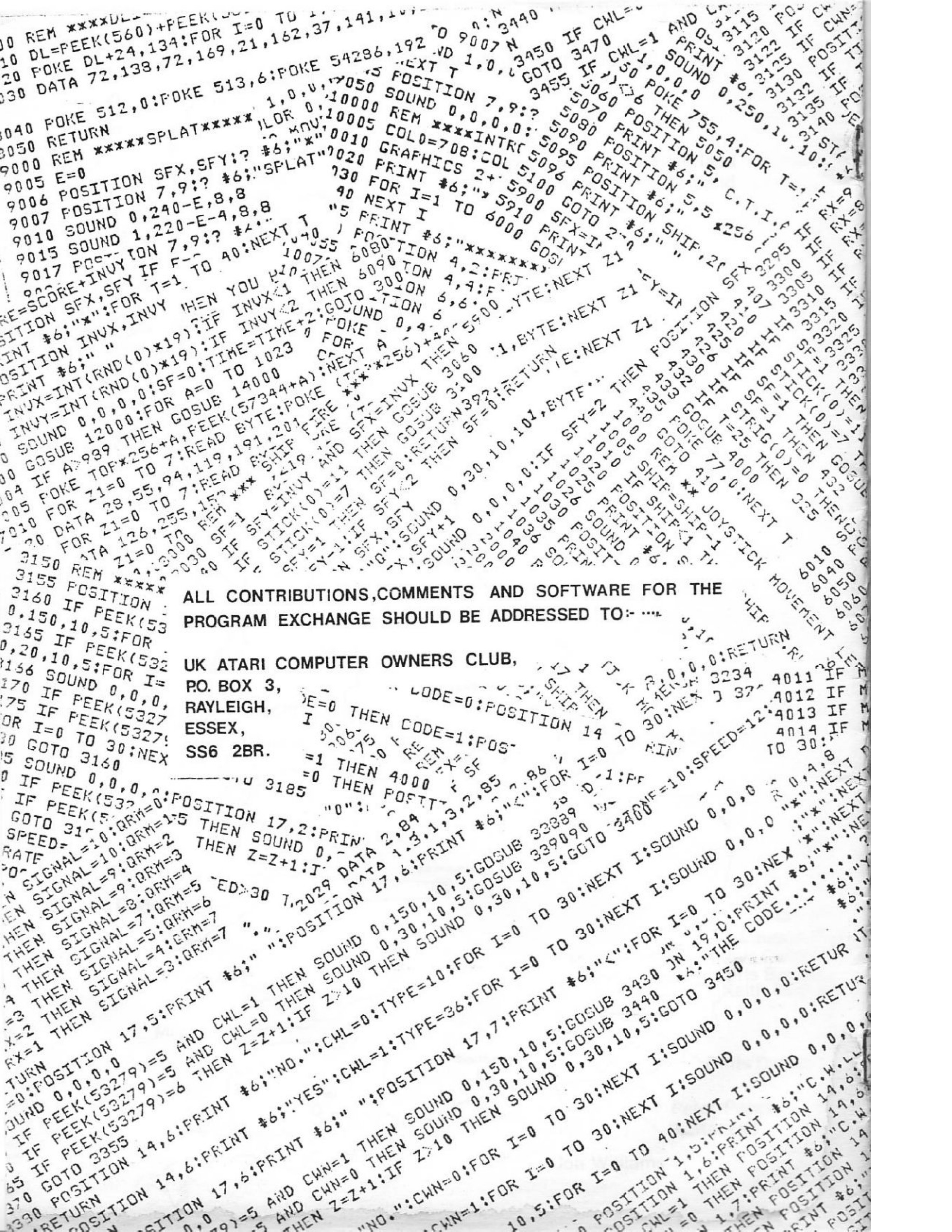
```
0760  AND #$10     HAS BIT 5
0770  CMP TEST     CHANGED STATE
0780  BEQ SKIP     NO
0790  STA TEST     YES RESET TEST
0800  LDA CNTRL    LOAD CONTROL BYTE
0810  TAX          AND SAVE IT
0820  BMI OFF      MSB SET SO SKIP IT
0830  BEQ CURSOR   ZERO SO DO CURSOR
0840  AND #$40     IS IT SPECIAL
0850  BNE BLINK    YES SO BRANCH
0860  TXA          RESTORE CONTROL
0870  LDX TEST     ON OR OFF?
0880  BNE STORE    ON
0890  TXA          OFF.SET TO ZERO
0900 STORE STA CHACT  STORE IT
0910  JMP SKIP        AND LEAVE
0920 BLINK LDA #2 LOAD INVERSE VAL.
0930  LDX TEST     ON OR OFF?
0940  BEQ STORE    ON
0950  LDA #3       OFF.SO LOAD OFF VAL
0960  BNE STORE    AND STORE IT.
0970 ;
0980 CURSOR LDA CRSINH  IS CURSOR OFF
0990 ;
1000  ORA FLAG     OR NOT INPUT
1010  BNE OFF      ONE IS NON-ZERO
1020  LDA OLDADR   MAKE SURE THAT
1030  CMP TEMP     THE CURSOR IS
1040  BEQ OK       SETTLED.
1050  STA TEMP
1060  BNE OFF      NO SO SKIP IT
1070 OK LDY #0     SET INDEX
1080  LDA #$80
1090  EOR (OLDADR),Y TOGGLE MSB OF
1100  STA (OLDADR),Y CURSOR ADDRESS
1110 OFF LDA #2       MAKE SURE CHACT
1120  STA CHACT       IN NORMAL MODE.
1130 SKIP JMP $E7D1  ALL DONE.......
1140 ;
1150 ;
1160 ;THE FOLLOWING ROUTINE ALTERS
1170 ;THE EDITOR GET SECTION OF O.S.
1180 ;TO ALLOW CURSOR CONTROL.
1190 ;
1200 ;
1210 MYGET JSR $FCB3 !AS PER O.S.LIST
1220  JSR $FA88
1230  LDA BUFCNT
1240  BEQ X1
1250  JMP $F67C
1260 X1 LDA ROWCRS
1270  STA BUFSTR
1280  LDA COLCRS
1290  STA BUFSTR+1
1300 E1 LDA #0       ..ENABLE CURSOR..
1310  STA FLAG
1320  JSR $F6E2
1330  STY DSTAT
1340  LDA #1         ..DISABLE CURSOR..
1350  STA FLAG
1360  LDA ATACHR
1370  CMP #$9B
1380  BEQ E2
1390  JSR $F6AD
1400  JSR $FCB3
1410  LDA LOGCOL
```

```
1420  CMP #113
1430  BNE E6
1440  JSR $F90A
1450 E6 JMP E1
1460 E2 JMP $F66E   BACK TO NORMAL
1470 ;
1480 ;
1490 HTAB=*
1500 ;
1510 ;
1520  .WORD$F3FB
1530  .WORD$F633
1540  .WORDMYGET-1
1550  .WORD$F6A3
1560  .WORD$F633
1570  .WORD$F633
1580  JMP $F3E4
```



```
0 REM .    SNOOPY.BAS            .
1 REM .   Chris Davies,                          71 Kingswood,
2 REM .      Bromley,                              Kent  BR2 0NR.
5 GRAPHICS 8:POKE 709,0:POKE 710,222:POKE 712,222
10 COLOR 1
20 PLOT 100,0:DRAWTO 220,0
30 DRAWTO 220,159:DRAWTO 100,159:DRAWTO 100,0
40 PLOT 106,42:DRAWTO 134,28
50 DRAWTO 139,16:DRAWTO 140,6
60 DRAWTO 144,4:DRAWTO 157,4:DRAWTO 164,7:DRAWTO 172,9:DRAWTO 182,6:DRAWTO 180,16:DRAWTO 182,29
70 REM END OF RIGHT SIDE OF HEAD OF HAT AT STATEMENT 60 ABOVE
80 DRAWTO 166,23:DRAWTO 159,22:DRAWTO 148,24:DRAWTO 134,28
90 PLOT 106,42:DRAWTO 128,40:DRAWTO 140,39:DRAWTO 148,38:DRAWTO 156,39:DRAWTO 164,39:DRAWTO 172,40:DRAWTO 180,41:DRAWTO 190,42
100 DRAWTO 204,49:DRAWTO 218.48:DRAWTO 196,40:DRAWTO 182,29
110 REM END OF HAT AT END OF STATEMENT 100 ABOVE
120 PLOT 190,42:DRAWTO 189,56:DRAWTO 187,64:DRAWTO 184,72:DRAWTO 180,74:DRAWTO 176,75:DRAWTO 172,74:DRAWTO 168,64:DRAWTO 163,47
130 REM INNER EAR
140 PLOT 185,42:DRAWTO 184,52:DRAWTO 182,66:DRAWTO 176,68:DRAWTO 172,64:DRAWTO 171,60:DRAWTO 170,48:DRAWTO 171,40
150 REM  FILL INNER EAR
160 POKE 765,1:POSITION 176,68:XIO 18,#6,0,0,"S:"
170 REM  EYE
180 PLOT 144,46:DRAWTO 145,52
190 REM  FRONT OF FACE
200 PLOT 132,40:DRAWTO 124,46:DRAWTO 120,52:DRAWTO 118,58:DRAWTO 118,64:DRAWTO 120,69:DRAWTO 124,76:DRAWTO 132,81:DRAWTO 148,80:DRAW
TO 152,80
210 REM  MOUTH
220 PLOT 148,80:DRAWTO 150,74:DRAWTO 156,71:DRAWTO 158,66
230 REM  LINES IN HAT
240 PLOT 148,4:DRAWTO 152,12:DRAWTO 156,14:PLOT 157,4:DRAWTO 160,10:DRAWTO 169,14
250 REM  NOSE
260 PLOT 120,69:DRAWTO 113,71:DRAWTO 111,77:DRAWTO 112,81:DRAWTO 117,81:DRAWTO 124,76
270 REM  NOSTRIL
280 PLOT 123,75:DRAWTO 116,80:PLOT 122,74:DRAWTO 115,79
290 REM  STOMACH
300 PLOT 152,80:DRAWTO 155,84:DRAWTO 154,90:DRAWTO 146,96:DRAWTO 143,112:DRAWTO 143,116:DRAWTO 144,116
310 DRAWTO 148,120:DRAWTO 152,124:DRAWTO 163,126
320 REM  LEFT FOOT
330 PLOT 148,122:DRAWTO 146,129:DRAWTO 144,128:DRAWTO 140,129:DRAWTO 128,130:DRAWTO 124,136:DRAWTO 118,140:DRAWTO 133,140
340 REM  CLAWS
350 PLOT 126,140:DRAWTO 128,136
360 REM  RIGHT FOOT
370 PLOT 160,126:DRAWTO 158,132:DRAWTO 150,132:DRAWTO 145,135:DRAWTO 134,136:DRAWTO 133,140
380 DRAWTO 140,144:DRAWTO 148,145:DRAWTO 156,144:DRAWTO 164,143:DRAWTO 172,143:DRAWTO 180,144:DRAWTO 181,140:DRAWTO 180,136:DRAWTO 1
69,130
390 REM  CLAWS
400 PLOT 140,144:DRAWTO 147,138:PLOT 152,144:DRAWTO 157,138
410 REM  GRASS
420 PLOT 106,137:DRAWTO 120,136:PLOT 109,130:DRAWTO 115,138:PLOT 117,130:DRAWTO 116,137
430 PLOT 182,142:DRAWTO 204,139:DRAWTO 211,144:DRAWTO 220,144
440 PLOT 184,137:DRAWTO 183,145:PLOT 188,137:DRAWTO 188,144:PLOT 196,136:DRAWTO 195,144:PLOT 204,137:DRAWTO 197,144
450 REM  BACK
460 PLOT 169,130:DRAWTO 168,124:DRAWTO 171,124:DRAWTO 172,118:DRAWTO 173,108:DRAWTO 172,96:DRAWTO 166,86:DRAWTO 164,80
470 DRAWTO 165,73:DRAWTO 169,71
480 REM  COLLAR
490 PLOT 152,86:DRAWTO 168,85
500 REM  TAIL
510 PLOT 173,109:DRAWTO 179,114:DRAWTO 188,113:DRAWTO 186,119:DRAWTO 182,120:DRAWTO 172,118
520 REM  FRONT PAW
530 PLOT 171,118:DRAWTO 168,119:DRAWTO 163,116:DRAWTO 165,106
540 PLOT 163,116:DRAWTO 159,110:DRAWTO 158,104:DRAWTO 160,90
9980 POKE 752,1
9990 ? ,"    SNOOPY"
10000 GOTO 10000
```





ALL CONTRIBUTIONS,COMMENTS AND SOFTWARE FOR THE PROGRAM EXCHANGE SHOULD BE ADDRESSED TO:-

UK ATARI COMPUTER OWNERS CLUB,
P.O. BOX 3,
RAYLEIGH,
ESSEX,
SS6 2BR.