THE FIRST SIX TRICKY TUTORIALS (TM)

FOR ATARI 400/800 COMPUTERS (TM)

#1 Display Lists

#2 Horizontal/Vertical Scrolling

#3 Page Flipping

#4 Basics of Animation

#5 Player Missile Graphics

#6 Sounds & Music

An exciting low cost way to learn many of the secrets of programming the Atari Computers.

For new or experienced owners.

By

Educational Software inc.

THE FIRST SIX

TRICKY TUTORIALS(TM)

There are many things that the ATARI computers can do either better or easier than other small computers. The following series of programs is designed for anyone who is at least familiar with BASIC programming. What each tutorial offers is similar to an extensive magazine article with all discussion in as simple language as is possible, plus you get MANY examples already typed in and running. There is little overlap in what is taught, so each tutorial will further convince you that buying an ATARI was the right choice!

#1 DISPLAY LISTS

This program teaches you how to alter the program in the ATARI that controls the format of the screen. For example: when you say graphics 8 the machine responds with a large graphics 8 area at the top of the screen and a small text area at the bottom. Now, you will be able to mix various modes on the screen at the same time. Just think how nice your programs could look with a mix of large and small text, and both high and low resolution graphics. The program has received rave reviews for the way it does all the calculations of the difficult things (like counting scan lines). You will guickly be able to use the subroutines included in your own programs. 16k memory required for tape - 24k for disk.

#2 HORIZONTAL & VERTICAL SCROLLING

The information you put on the screen, either graphics or text, can be moved up, down or sideways. This can make for some nice effects. You could move only the text on the bottom half of the screen or perhaps create a map and then move smoothly over it by using the joystick. Includes 18 examples with several using a small machine language subroutine for smoothness. As always, our examples can easily be used in your own programs. 16k tape-24k disk.

#3 PAGE FLIPPING

Normally you have to redraw the screen every time you change the picture or text. Now you can learn how to have the computer draw the next page you want to see while you are still looking at the previous page, then flip to it instantly. You won't see it being drawn so a complicated picture can seem to just appear. Depending on your memory size and how complicated the picture, you could flip between many pages, thus allowing animation or other special effects. We have found that many people skip this tutorial either because it sounds hard, or they think they will never use it. The basic method takes only 12 lines and the usefulness is infinite. 16k tape - 24k disk.

#4 BASICS OF ANIMATION

This program shows you how to animate simple shapes (with sound using the PRINT and PLOT commands, and also has a nice little PLAYER/MISSILE GRAPHICS game you can play with. The P/M example is well commented and will get you started on this complicated subject (more fully explained in TT#5). This would be an excellent way to start making your programs come alive with movement! Recommended for beginning users. 16k tape - 24k disk.

#5 PLAYER MISSILE GRAPHICS

This is the big one! We start by showing how to create a simple shape called a player, then take you through over 25 examples until you have created a complete business application and a small game. Also, we include a utility to create the shapes and choose the colors, then store the players and missles in data statements. Later YOUR programs bring these shapes back in when needed. Plus much more! 32k tape or disk.

#6 SOUNDS & MUSIC

One of the famous programmers for our great machine offers this one through us. Unless you have spent many hours experimenting with the four voice channels, you will learn alot from this one! The nicest part is the MANY examples of special sound effects that you can refer to when you need them for a program or to impress a friend. 16k tape - 24k disk.

Many other fine programs are available from:

Educational
Software inc.
4565 Cherryvale Avenue
Soquel, Ca. 95073
(408) 476-4901

Introduction To Santa Cruz Software

Hi! First let us thank you for purchasing our programs. We want to start out by telling you a little about ourselves and our company. Santa Cruz Educational Software is the result of several local programmers who initially purchased the ATARI computer as a home machine around the time the 800 was first sold. At that time, and even today, we found a lack of information about how to use the power of the machine. As you have seen in a few of the better programs being sold, the ATARI can do more than any other computer in it's price range, but how could we learn the many "tricks" that were contained withen the machine??

Well, our local club, although it had several HUNDRED members, ignored most of our questions. Fortunately other clubs across the country have some excellant newsletters full of programs and facts. Also, ATARI(tm) has given we owners the Operating System and Hardware manuals. Finally, many magazine articles and several books have been published.

All this is great for programmers, but what about the average owner who doesn't understand much of what is said in the magazines? Even as a programmer, I have to spend many hours studying all of the information that is now available inorder to understand some of unique things about our machine.

The story ends when I asked some local club members to submit their best programs for us to offer to others at the lowest possible price, ie. a price WE WOULD be glad to pay to buy these programs. I myself wrote most of the TRICKY TUTORIALS(tm) and the MASTER MEMORY MAP. Now, others across the country have offered programs for us to sell, all meeting the goal of: minimal on fancy artwork and a few spelling errors, but worth every penny.

Write us with new ideas or memory locations to share, and if it's something we can use, a reward will be sent back to you as soon as we can. Also, please remember that this is only a part time business. We are usually late in getting new programs out due to last minute debugging at midnite(no kidding, I do all this after my regular job). This doesn't mean we don't care!

My thanks to all of you who have raved about the Tutorials. I will try to keep a new one comming out every three months or less, and also to get out those bugs that remain.

ROBIN ALAN SHERER (owner, programmer, and janitor)

****

HOW TO LOAD

****

TAPE......

First, if you haven't cleaned your tape recorder heads lately, please do so now. SEE ANY STEREO SHOP FOR THE RIGHT TOOLS. Place the tape with the label side up in your recorder. Make sure it is rewound and also reset the counter to 0. Push PLAY on the recorder. TYPE RUN"C:" AND PRESS RETURN. If this doesn't work you might try CLOAD. We also include a backup program on the shorter programs we sell (not the TUTORIALS), and these sometimes require CLOAD. If the program won't start to load, try positioning the tape forward or backward a little bit at a time. The easiest way would be to LISTEN to the "noise" on the tape with a regular tape recorder. When you find the steady tone that lasts for about 8 seconds, you have the beginning of each program. We recommend you write down the number on your recorders counter as each program starts. This will make it easier to find each part later on if needed. We occasionally get a bad tape from our supplier, so if yours won't load on both your own 410 and a friends (or your DEALERS), call (408) 476-4901 for a replacement.

Once the program starts, it will load in the remaining parts. The multiple parts are needed so that machines with only 16k can enjoy the TRICKY TUTORIALS. With more memory you can go beyond these simple tutorials. After each part, the computer will beep. This is your signal to press return to load in the next part. Some newer programs will start the next part without pressing return. Most programs will run themselves when done loading.

The reason for several methods of running tapes is that as we improve the programs, we change the masters; however, the manuals are printed in large amounts, so changes are VERY hard to get into your manuals!

****

DISK.....

To load & run the disk you first have to turn on the drive. When the busy light goes out place the disk in the drive. Now turn on the computer with the basic cartridge in place and the program will load each part and run by itself (aren't disk drives nice!).

DISPLAY LISTS

Display Lists consists of a set of programs that are simple to use, but deal with a complicated subject. Until now, only a few programmers have understood enough about modifying the ATARI's Display Lists. Using the examples and manual inside, you have only to follow a few simple directions to create your own custom screens.These screens can consist of any of the ATARI'S regular Text and Graphics modes plus 5 new ones. Imagine up to 20 modes on the screen at once....real special effects in all your programs!

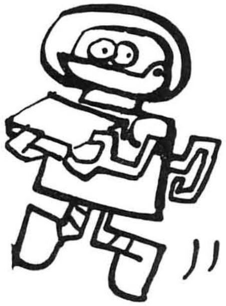
This program requires very little actual programming experience. It is especially designed to allow you to use it now, and go back and learn the actual method within the program at any time in the future.

REQUIRES 16K TAPE OR 24K IF YOU HAVE DISK.

Educational Software

presents

TRICKY TUTORIAL #1



DISPLAY LISTS

TRICKY TUTORIAL #1

DISPLAY LISTS

by
Robin Sherer

INTRODUCTION

Display List modification is a large and complex subject. If you have seen any of the articles recently published in the hobby magazines on the subject, you know you could never learn to modify your own custom Display Lists.....or could you? What if we let the computer do most of the work?

How to Load

TAPE....

Place the tape in your recorder, label side up. Make sure the tape is rewound, and reset the counter to zero. Push PLAY on the recorder, type CLOAD and press RETURN. When the READY prompt appears, type RUN and press RETURN. If the program won't start to load, try positioning the tape forward or backwards a little. A little trick to find the beginning is to first, turn your volume UP. Then POKE 54018,52 to start the cassette motor. Listen to the "noise" on the tape. When you find the high-pitched, steady tone, you have the beginning of the program. We recommend you write down the number on your recorder's counter as each program starts, this will make it easier to find each part later on. POKE 54018,60 to turn the cassette motor off.

DISK....

To load and run the disk, first turn on your disk drive. When the busy light goes out, place the disk in the drive. Now turn on the computer, with the BASIC Cartridge in place and the program will load each part and run by itself.

Any defective tapes or disks should be returned to:

Educational Software
4565 Cherryvale Ave.
Soquel, CA. 95073

SO YOU WANT TO LEARN ABOUT DISPLAY LISTS!

To use this program you actually don't need to understand most of the information we will talk about. Feel free to just run the examples and play with creating your own special screens. Later, when you want to understand more about what you are doing, come back and read the booklet and practice modifing the examples. You can't hurt the machine by changing the numbers in the Display List. At the worst, the machine might "go to sleep" (the keyboard will not respond) if a wrong number is POKEd into memory. Then you'll have to press RESET or turn off the machine and reload in the program. This is one reason tape users should always keep track of the number on their recorder's counter for each program so that they can easily reload a program.

This Tutorial doesn't go to the other extreme of dificulty either. MANY details about DL's are not mentioned. The lesson is designed to explain all the basic's of the subject as well as offer examples already typed in for you! (I've got to keep those two recycled beer cans,Prototype & Mototype busy doing something!) If you don't practice modifing the examples with your own ideas, DL'S probably won't make sense to you.

NOTE - From this point on we are going to refer to Display Lists as DL so please don't get confused.

What is a DL?

Why did you buy this program? A surprising number of people buy my tutorials because they want to know how to use the special tricks that the ATARI can do, not really understanding what these tricks are. With this in mind, we'll start out this lesson by explaining some basics of what a Display List is. EXAMPLE 9 shows one suggested use, but you can come up with you own unique ideas!

The ATARI has a special chip inside it to take the information from within its memory and put it on the screen. This chip is called the ANTIC. This chip is actually a microprocessor by itself and thus has a set of instructions to program it....just like the main microprocessor in the ATARI, the 6502. The difference is we don't use BASIC or Assembly language to program it. Now stop and think a moment, I said TWO microprocessors. This gives the machine much more capability than an Apple or Pet (go brag to your friends for a moment then come back........)

The Display List is what the ANTIC looks at to tell the machine how to treat the screen (plus other things not mentioned in this lesson):

The Display List is what the ANTIC looks at to tell the machine how to treat the screen (plus other things not mentioned in this lesson):

1)What Graphics and text modes to put on the screen.

2)Where to get the data to put into these mode areas.

Now don't worry that the subject is too complex. There are only four types of instructions, and these are POKEd into memory using a few basic statements. Let's review - The DL says something to the computer that means: "Put on the screen some Graphic Mode 0, now some Mode 7, then some Mode 0 again, and finally some Mode 2. Oh, by the way, get the data from this location in memory".

We have to discuss the meaning of a few words before we can go on. You already have used Graphics Modes 0 to 8 as described in the BASIC Manual that came with the computer. If you look at Figure 1 you will see a chart showing something called O.S. Modes. The Display List instructions allow the use of these Operating Systems Graphics Modes from 2 to 15. Notice that Graphics Mode 0 is O.S. Mode 2 and Graphics Mode 8 is O.S. Mode 15. In the middle are some new modes you can use from BASIC that ATARI didn't mention in their BASIC book. Uses for these new modes will be mentioned later. So just remember when we say Graphic Mode we mean the one you normally use from BASIC and the O.S.(Operating System) Mode is the one you place in the Display List(through the use of our examples).

O.S MODE [# for DL]	Basic GR. mode	# of Pixel lines per mode line	# of Colors	Bytes of Memory Used Per Line	Text or Plot Mode	Additional
0	-	-	-	-	blank line	in the DL:
1	-	-	-	-	jump	16=2 blank lines
2	0	8	2	40	text	32=3 " "
3	new	10	2	40	text	48=4 " "
4	new	8	4	40	text	68=5 " "
5	new	16	4	40	text	80=6 " "
6	1	8	5	20	text	96=7 " "
7	2	16	5	20	text	112=8 " "
8	3	8	4	10	plot	65=jump & wait
9	4	4	2	10	plot	
10	5	4	4	20	plot	
11	6	2	2	20	plot	
12	new	1	2	20	plot	
13	7	2	4	40	plot	
14	new	1	4	40	plot	
15	8	1	2	40	plot	

FIGURE 1

Here is another idea to learn about. We are going to talk about both Mode Lines and Pixel Lines. To see a Pixel Line, simply look closely at the screen of your TV set. Those tiny dots you see are pixels, so a row of them across the screen is called a Pixel Line. The Mode Lines are harder to understand. If you look at some regular Mode 0 text(small white letters with blue background) you can see that the letters are made up of eight rows of pixels (including the space at the top and bottom). Now look at the chart for GR. mode 0. Under the value called "Pixel Lines per mode line", it says "8", meaning a character in GR.0 takes 8 rows of pixels. You'll use this value later to help yourself in setting up the DL. To review (see Fig.1), a mode line of GR. 8 takes only 1 row of pixels, and O.S. mode 5 (no equivalent GR. mode) takes 16 rows of Pixel Lines, thus, adding GR.5 to your DL fills up the screen 16 times faster than a GR. mode 8 line (O.S. mode 15....look at the chart!). When you input a mode into the Display List in the examples, you must remember if you take out a mode line that is made up of many Pixel Lines, you should replace it with several Mode Lines that use less pixels per line. This will keep the total number of Pixel Lines drawn on the screen the same. The standard number of Pixel Lines drawn on the screen is 192. If you put too few Pixel Lines on the screen, then the picture will "shrink" with just black on the bottom. Likewise, too many will place part of the picture off the screen and may cause the picture to "roll", which can NOT be stopped with the Vertical Hold controls on your TV. If you practice it will all make sense!

EXAMPLE 1

If you haven't already, please RUN Example 1. This little program allows you to look at any of the standard GR. mode DL's, including the new (to the United States) GTIA modes 9, 10, & 11. Let's try inputting a 0. The numbers that result from the example are those of a standard small text screen. You read the numbers a row at a time.

```
                  EXAMPLE OF
               GRAPHICS MODE 0
              -----------------

112 <---8 blank lines
112 <---8 blank lines
112 <---8 blank lines

 66 <---LMS (includes first line of screen;
              ie. 64(LMS) + 2(O.S. Mode)

 64 <---Low Byte\
                 combine to get location of Data
                 for upper left corner of screen
156 <---High Byte/

  2 <---O.S. mode line (same as GR.0)
  2 <---O.S. mode line
  2 <---O.S. mode line
  2 <---(and all of the other 2's) O.S. mode line

 65 <---Says end of Display List, go to the
        Display List at next location

 32 <---\
         Combine=Low Byte + High Byte * 256
156 <---/
```

FIGURE 2

The first three numbers are 112. Look at Figure 2. You'll see that "112" tells the computer to place 8 blank lines on the screen, starting at the top of your set. Three of these commands causes 24 blank lines at the top of the screen. This number of blank lines is standard. They allow for the difference in individual TV sets, called overscan, so that your whole picture can be seen.

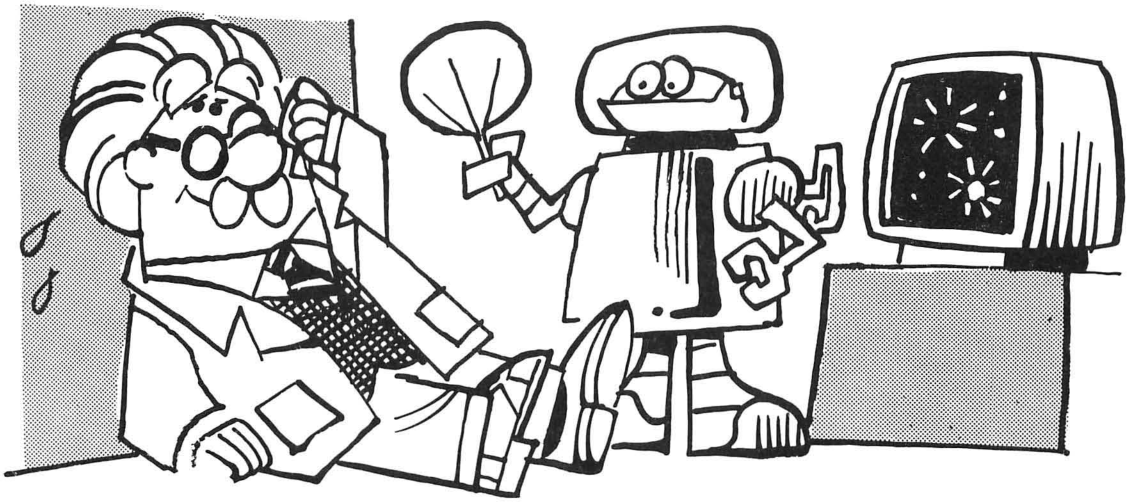
The next number, "66", is the LMS instruction. LMS means Load Memory Scan. LMS tells the computer to "Go find the data for the following modes starting at the address in memory in the next two numbers". We will always use for this number(LMS) the value of 64+the O.S. Mode number.....this means for our GR.0 DL we need 64+2, or 66. Remember that the chart gives GR.0 = O.S. Mode 2. The next two numbers will be the location of the start of the data that is to be put on the screen. To find the value for your machine (if your interested) take the second number of the address(high byte), multiply it by 256 and add the first number (low byte). For example, if the high part of the number was 132 and the low part was 192 then the start of the screen data would be at 256*132+192 which equals 33984. If you POKEd a number into this location that was the value of a letter, number, or Graphics character, you would see it appear at the upper left corner of your TV screen. In this example, POKE 33984,33, an "A" will appear. This is because a 33 in the screen data means "put an A on the screen at this location."

NOTE!

I want to explain an important point about the "Screen Data". The information (data) that is put on the screen is normally stored all in one place in memory, and in the same form. It is the type of mode line you write the data on that determines if it will be large or small text, or graphics pixels. Later some of our examples LIST the program to get data to "flow" through the area where the screen data is stored. Please look closely at the way the same information is INTERPRETED differently as it crosses through the different modes. This concept is IMPORTANT!

Next in the DL come several # 2's. These numbers, we already mentioned, are the O.S. modes that you want. Since we are in GR.0, the DL needs 2's. For each of the 2's(including the one "hidden" in the LMS number before), the computer will put a GR. 0 mode line (8 pixels high!!) across the screen. Don't forget the mode line in the LMS instruction!

Finally comes the number 65. Every DL we will do has one. The chart gives the meaning of the number 65 as "go back to the Display List at the following address and wait for the next time the screen is drawn". Don't worry about this except to include the 65 at the end of your DL. Leave whatever numbers come after this alone. You could actually flip to a completely new DL and it's data by just changing these numbers.



WOW...THAT WAS A LOT!

Sure it was, but that's about it for the technical talk. Now we can have some fun! Input other numbers (0 to 11) intc Example 1. Notice that the DL gets longer as you use modes with less Pixel Lines per Mode Line; ie. for a mode like 7 where a mode line is only 2 pixels high, the DL must have a lot of number 13's in it to fill up the screen. Input 8, then look in the middle of the DL. See the 79. There is already a 79 in the top of the DL for the LMS number. Recall that 79 =64+15(the O.S.mode for GR. 8), but why a second LMS number and a second set of addresses for the screen data? The reason is that the DL can only locate up to 4k (4096 bytes) of data. Then it gets lost! The second LMS comes at the point where it ran out of data and says "Here's where to go find some more data to fill the remainder of screen". The second LMS is due to a hardware limit. You will have to make careful calculations to work with it, but only if you use a DL that requires more than 4k on screen data!

```
1000 GRAPHICS 17:? #6;"  EXAMPLE 1"
1010 FOR W=1 TO 230:POKE 710,W
1020 FOR ZR=1 TO 10:NEXT ZR
1030 NEXT W
1100 GRAPHICS 0
1110 DIM NUM(12):DIM A$(1):DIM N(212)
1112 FOR I=1 TO 12:READ X:NUM(I)=X:NEXT I
1114 DATA 32,34,24,34,54,54,94,94,176,202,202
,202
1120 TRAP 40000:TRAP 1120:? "K";POSITION 1,5:
? "WHICH GRAPHICS MODE WOULD YOU LIKE TO  SEE
THE DISPLAY  LIST FOR";
1125 POKE 710,0
1130 INPUT MODE
1140 IF MODE>11 OR MODE<0 THEN 11\
1150 GOTO 1220
1165 GRAPHICS 17:POKE 708,52
1170 POSITION 1,5:? #6;"MODES 0 TO 11 ONLY":S
OUND 0,120,8,8:FOR W=1 TO 500:NEXT W:SOUND 0,
0,0,0
```

```
1180 FOR W=1 TO 500:NEXT W:GOTO 1120
1220 USE=NUM(MODE+1)
1232 GRAPHICS MODE
1234 NOW=PEEK(559):POKE 559,0
1240 J=1
1245 DL=PEEK(560)+256*PEEK(561)
1250 M(J)=PEEK(DL+J-1):J=J+1
1252 IF J<USE+1 THEN 1250
1254 GRAPHICS 0:K=1:POKE 559,NOW
1260 POSITION 8,0:? "DISPLAY LIST FOR MODE ";
MODE:POSITION 1,1:? "------------------------
-------------"
1270 FOR J=2 TO 22
1280 FOR I=1 TO 40 STEP 4
1290 POSITION I,J:? M(K):K=K+1:IF K>USE THEN
1310
1300 NEXT I:NEXT J
1310 POSITION 2,23:? "WOULD YOU LIKE TO SEE A
NOTHER ";:INPUT A$
 1320 IF A$="Y" THEN 1120
 1330 IF A$="N" THEN POKE 764,12:RUN "D:EX2"
 1340 GOTO 1260
```

EXAMPLE 2 (at last)

If you are still in example 1 type in "n" and press RETURN. Example 2 is for you to practice changing a Display List. The numbers that come up in a data statement on the screen are there for you to edit using the cursor controls. Numbers can be added, changed, or deleted. When ready to see what the screen will look like with your new numbers, press RETURN. In case you don't understand what to do we have already changed some 2's to 4's, so just pressing RETURN will get a custom screen. Now you can see why ATARI didn't tell you about some of the special modes. Mode 4 is a multicolored text mode designed for use with character graphics, but it sure looks funny! It also is difficult to use, so these new modes will not be explained here. Try them on your own, or see ATARI publications.

Play with this example awhile. Try different numbers in the list. Remember (I keep repeating myself), the numbers in the DL are the O.S. Mode numbers from 2 to 15. This is designed as a simple example, so don't get frustrated that you have to keep starting over with the same list. Later examples will remember your changes and have more room in the DATA statements for numbers. ANYTIME YOU WANT TO CHECK THE MODES YOU HAVE PLACED ON THE SCREEN, THE EASIEST WAY TO DO IT IS TO BREAK THE PROGRAM AND TYPE LIST. THE LISTING WILL FLOW ACROSS THE MODES SHOWING YOU WHERE TEXT AND GRAPHICS ARE.

```
2 GRAPHICS 17:? #6;"    EXAMPLE 2"
3 FOR W=1 TO 230:POKE 710,W
4 FOR ZR=1 TO 10:NEXT ZR
5 NEXT W
7 GRAPHICS 0:DIM ZX$(135)
10 DL=PEEK(560)+256*PEEK(561)
15 A=PEEK(DL+4):B=PEEK(DL+5)
20 GRAPHICS 0:? "THIS EXAMPLE ALLOWS YOU TO C
HANGE A    MODE 0 DISPLAY LIST!"
30 ? "PRESS START TO BEGIN"
55 DATA 112,112,112,66,64,156,2,z,2,2,2,2,2,2
,2,4,4,4,4,4,2,2,2,2,2,2,2,2,65,32,156,0,0,
0,0
62 POKE 53279,8
65 Z=PEEK(53279)
70 IF Z=6 THEN 100
80 GOTO 65
100 TRAP 40000:? "↰";POSITION 2,5
105 ? "55 DATA 112,112,112,66,";A;",";B;",2,2
,2,2,x,2,2,2,2,4,4,4,4,4,2,2,2,2,2,2,2,2,2,65
,32,156,0,0,0,0"
110 POSITION 2,15:? "CHANGE THE VALUES YOU WA
NT BY USING    THE CURSOR CONTROLS-THEN PRESS
RETURN TO SEE NEW ";
115 ? "SCREEN"
120 POSITION 0,1
125 INPUT ZX$
130 POSITION 0,5
140 POKE 842,13
142 POSITION 2,12:? "CONT"
144 POSITION 0,3:STOP
150 POKE 842,12
200 RESTORE 55
210 I=0
215 TRAP 100
220 READ G:POKE DL+I,G:I=I+1:IF G=0 THEN 235
230 GOTO 220
235 FOR W=1 TO 900:NEXT W:LIST
240 ? "PRESS OPTION TO GO ON";? "PRESS START
TO DO ANOTHER SCREEN"
245 TRAP 100
250 POKE 53279,8
260 Z=PEEK(53279)
270 IF Z=6 THEN 57
280 IF Z=3 THEN RUN "D:EX3"
300 GOTO 260
```

EXAMPLE 3

This is a small subroutine for you to use in your own programs. The numbers for the DATA statement (the DL) can come from EX. 2 (or better yet EX. 10). Some of the lines in this example we added to make this Tutorial flow smoothly, so after you copy the example, delete all but the call to the subroutine in line 700 plus the subroutine itself, lines 10000-10040 . Here's how the routine works:

```
10010 DL=PEEK(560)+256*PEEK(561)
```

Line 10010 - puts together the two parts of the address stored in 560 & 561. These two locations combine to give you the start of the Display List. Since this varies, you have to look at these locations each time you do a Graphics call, like GR. 0 or GR. 7. The Graphics call will automatically set up a DL and screen data area within memory.

I'm confused!

Right! We need another chart (next page, Figure 3). On this chart are every standard Display List. Look at the first one, for Graphics 0. The Display List starts about 1k from the top of memory with the DL itself. Then, moving up in memory comes the 960 bytes of data that the screen needs. Example 2 calls GR. 0, then looks at 560 & 561 and changes the DL. The 960 bytes of screen data is enough for the modifications you can make with the small DATA statement we have given you so far. Later, when we give you four DATA statements you may want to start with a Graphics Mode that gives you more room like 5,7,even 8. The key is to use only as much of memory as you need. The real way to calculate what mode to start with is to add up how many lines of each mode you are using, multiply each of these by the number of bytes per mode line from FIG.1 , and get a total for them all. For example:

```
   13 lines of GR. 8 * 40 bytes per line
 +  8 lines of GR. 2 * 20 bytes per line
 + 25 GR. 7 lines * 40 bytes per line
   -------------------------------------
   equals:1680 bytes needed for DATA
      and:191 pixel lines used(close enough to 192!)
```

If you allow another 70 or so bytes for the DL itself, you get a minimum of 1750 bytes needed for your custom DL. By calling a standard GR. 6 (see chart), 2048 bytes would be set up which is enough. At this point (after the call), you would look at 560 & 561 to get the address of the start of the DL. By the way, the estimate of 70 additional bytes for the DL comes from adding up the number of mode lines you are going to call (46 in above example), plus some extras for the blank lines, the LMS bytes, and the 65 & address at the end of every DL.

Let's finish this example and then explain the chart.

```
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
```

Line 10011 - saves the location where the screen's data starts, which always comes at the 5th and 6th value into the DL(right after the LMS).

```
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1
10016 IF I=6 THEN POKE DL+5,Z2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,66,64,156,2,2,2,5,5,5,5,2,2,
      4,4,4,4,4,65,32,124,0,0,0
```

Lines 10012 to 10030 - We now set a counter, and read a value; if it = 0 we're all done; if not, we POKE it into the first location in the DL(thus changing the DL if the number is different). We then test for the 5th and 6th value, and if we just POKEd in these values from the DATA statement, we POKE in another value from line 10011. This is because these two numbers depend on your memory size, so the numbers in the data statement are dummys waiting for these correct numbers to be PEEKed from the DL and then POKEd right back in again(sure it could be done other ways....this is my way). This simple loop continues until the program reads in a 0 from the DATA statement then stops. Notice whenever you change a DL this way it changes down the screen just like a curtainfalling. Neat!

DISPLAY LIST
FOR GRAPHICS 0 to 4+16

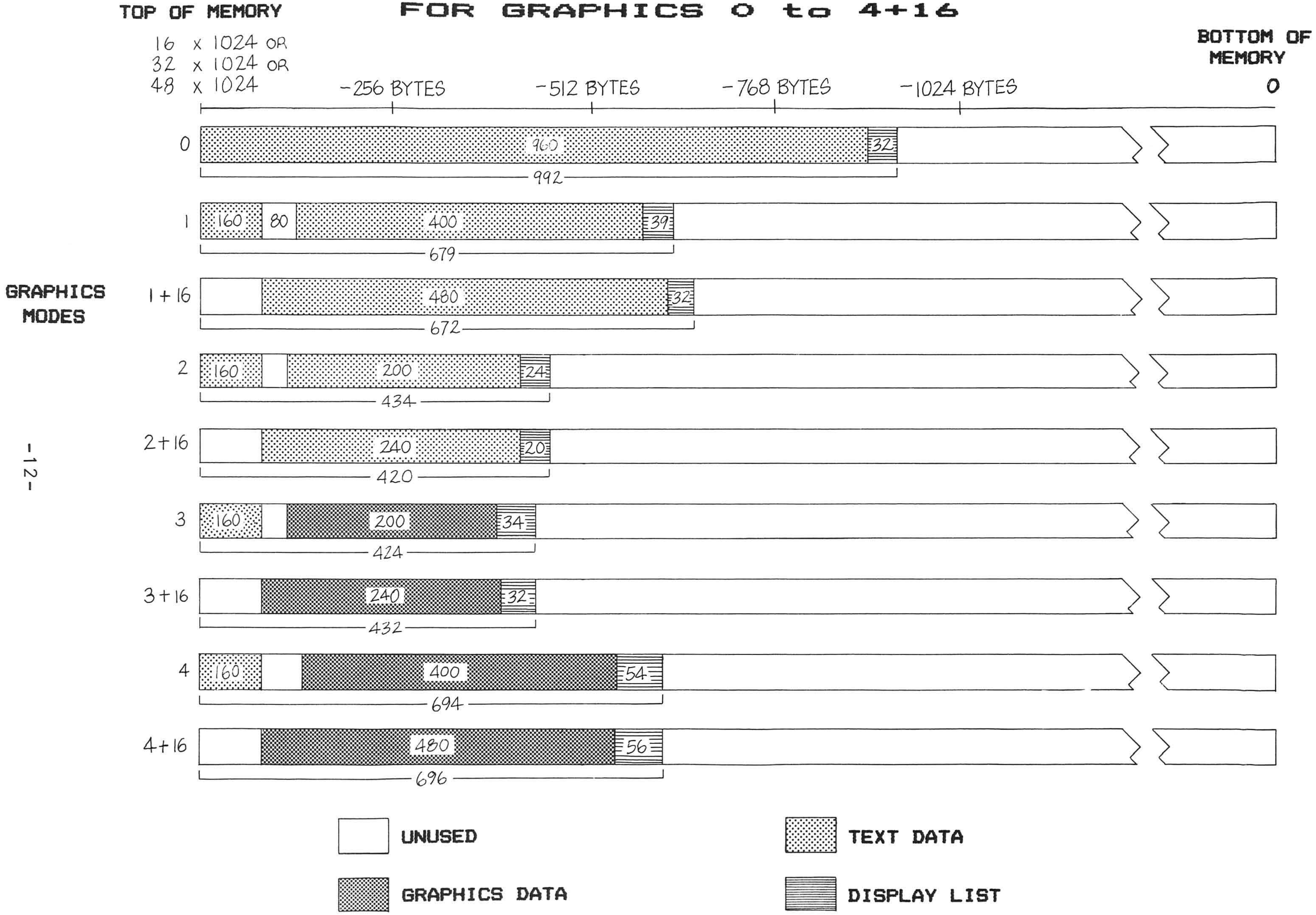


FIGURE 3A (NOT TO SCALE)

DISPLAY LIST

FOR GRAPHICS 5 to 8+16

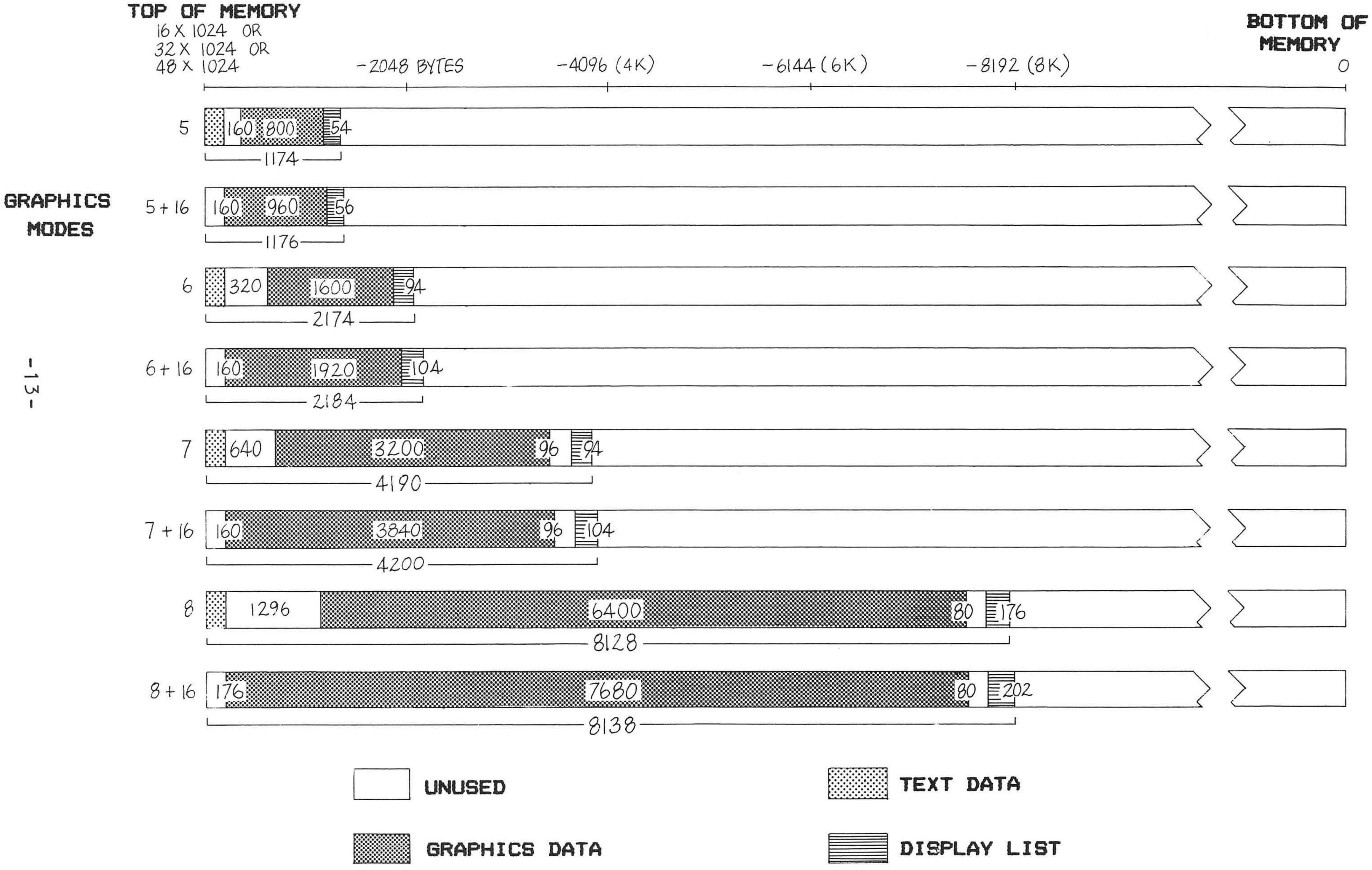
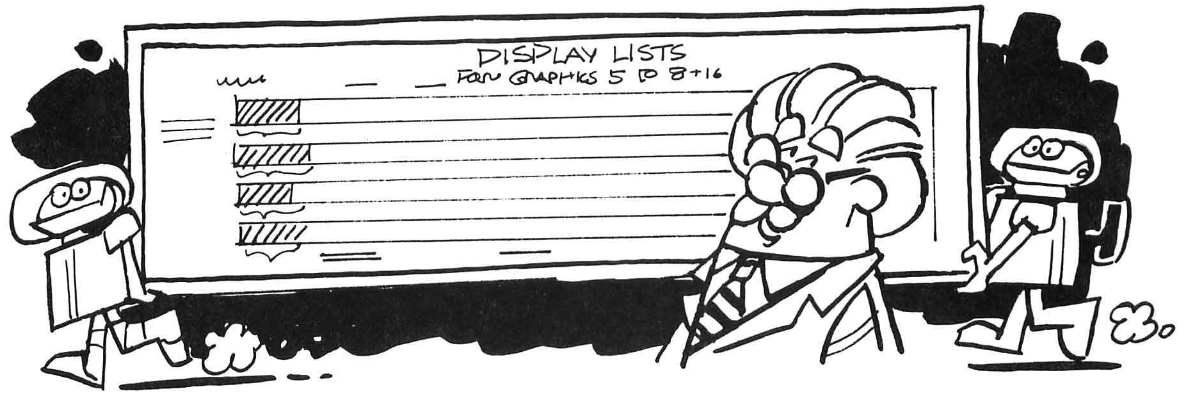


FIGURE 3B (NOT TO SCALE)



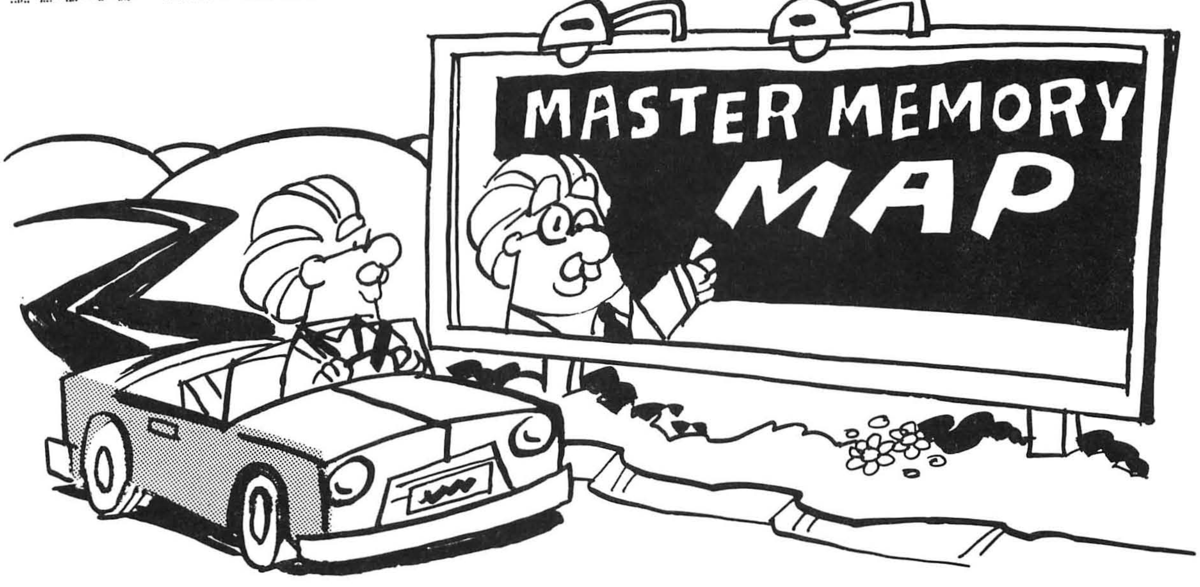
BACK TO THAT BIG CHART AGAIN

It isn't necessary for you to understand Figure 3, so feel free to skip this section. Just pick one of the Graphics Modes you are used to using and look at it on the chart. Lets pick GR.4 (O.S. mode 9). Starting down from the top of memory about 600 bytes, we see the DL uses 54 bytes of memory. Next comes the screen data of 400 bytes, then some empty (unused) memory locations. Finally, 160 bytes of data for the text window. The other Graphics Modes work the same way.

The text window is a slight complication to the DL after the sequence of 112's, the LMS bytes, and the various O.S. Mode numbers, as discussed above, YOU WOULD FIND ANOTHER LMS! And following that will come more mode line instructions (text of course), and now finally comes the 65 address to end the DL. The extra LMS just says to: "Go get some data from another location in memory than the one the computer is at, and then place that data in some text mode lines(usually 4 lines of GR.0), all at the bottom of the screen.

```
10 GRAPHICS 17:? #6;"  EXAMPLE 8"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
500 REM THIS IS A SUBROUTINE FOR YOU TO USE I
N YOUR OWN PROGRAMS. SEE MANUAL FOR DIRECTION
S
700 GOSUB 10000
710 REM PUT THE REST OF YOUR PROGRAM STARTING
ON THIS LINE
715 FOR W=1 TO 2000:NEXT W
720 GRAPHICS 0:? "THIS EXAMPLE IS FOR YOU TO
USE IN YOUR PROGRAMS.";? "PRESS BREAK AND SAV
E IT IF YOU WANT OR";
```

```
730 ? " PRESS OPTION TO GO ON.":POKE 53279,8
740 Z=PEEK(53279)
750 IF Z=3 THEN POKE 764,12:RUN "D:EX4"
760 GOTO 740
10000 TRAP 10040
10010 DL=PEEK(560)+256*PEEK(561)
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1
10016 IF I=6 THEN POKE DL+5,Z2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,66,64,156,2,2,2.5.5,5,
5,2,2,4,4,4,4,4,2,65,32,124,0,0,0
10040 RETURN
```



NOW FOR AN ADVERTISEMENT

All these PEEK and POKE locations are summerized in my MASTER MEMORY MAP. FORK OVER THE $6.95 and order one today!

I hate ads too, but....News flash!!! The ATARI isn't as slow as some reviewers have stated in bench tests. The ATARI ANTIC chip that we have been programming steals time from the main 6502 processor. The bigger the DL and the more data it has to put on the screen, the more time it steals. This means that if a reviewer tested the machine in GR.0 it will run calculations slower than if in GR.2 or 3 which have smaller DL's. In fact, if you want you can customize a DL to have only blank lines with one small message in the middle of the screen that says "Wait a Moment", this will allow the machine to run almost as fast as it can go. In case you want to know the difference; in GR.8 the machine is up to 40% slower than GR.2!

EXAMPLE 4 & 5

Ok students, time for fun. Run example 4. All of this DL stuff may not seem too practical, so the next two examples are to give you some ideas. By simply setting up a GR.1 (large letters) DL, but in the middle doing a second LMS (70,96,127 in the data but the last two numbers are rePOKEd in lines 10023-10024 depending on your memory), we tell the DL that once it gets about halfway down the screen, it should get its data from the same place as the top of the screen did! There are few restrictions on what you can do with the LMS byte. It doesn't have to wait to be used only at the beginning of the DL and at 4K boundaries.

Example 5 is the same thing, but with only one copy of what you type in. Both of these could be used for games, but a more useful idea might be to allow programming in big letters for the hard of seeing or kids to use.

HAVE I LOST YOU?

You should still be in example 4. The program says READY in double letters. Since the program has stopped, you can do what you like. Try a simple line like "Hello, I use BIG letters". The computer will now work like it was in GR.0. Why? Look at line 720 in EX.4--720 POKE 87,0:END. The POKE to location 87 fools the computer. We just (in the subroutine) set up the DL like a GR.1 screen, but the computer looks to location 87 to see what mode it is in, so by POKEing in 0 (the Graphics Mode goes here, not the O.S. mode used in the DL), it thinks it can write listings as though it were in GR.0.

```
10 GRAPHICS 17:? #6;" EXAMPLE 4"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
490 REM *****************************
```

```
500 REM HERE IS A PRACTICAL EXAMPLE OFWHAT YO
U CAN DO WITH A DISPLAY LIST THAT IS MODIFIED
510 REM JUST SAY GOTO 700      WHEN YOU  WANT T
O SEE YOUR PROGRAM IN BIG  LETTERS AND IN DUP
LICATE!!
520 REM **********************************
700 GOSUB 10000
710 REM YOUR BASIC PROGRAM GOES HERE
720 POKE 87,0:END
10000 REM TRAP 10040:GRAPHICS 1
10005 GRAPHICS 1
10010 DL=PEEK(560)+256*PEEK(561)
10015 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
10016 I=1
10017 READ A:IF A=0 THEN 10040
10020 POKE (DL+I-1),A
10021 IF I=5 THEN POKE DL+4,Z1-32
10022 IF I=6 THEN POKE DL+5,Z2+2
10023 IF I=16 THEN POKE DL+15,Z1-32
10024 IF I=17 THEN POKE DL+16,Z2+2
10025 I=I+1:GOTO 10017
10030 DATA 112,112,112,70,96,127,6,6,6,6,6,6,
6,6,70,96,127,6,6,6,6,6,6,6,6,6,65,94,125,0,0
10040 RETURN
```

Example 5 is almost the same, except instead of the POKE 87, we start with a GR.0 mode. This internally changes 87 to 0 (try a PEEK(87) if you don't believe me). Sometimes one method is more desirable than the other. You pick one. The other difference is that there is no second LMS in the middle of the DL. The writing in these two examples is large, but the computer still thinks it is in mode 0 so it trys to fit in the normal 40x24 letters. For this reason, you won't see the text in the way that you expect. Try a LIST command. STRANGE!

NOTE - To get to example 5 & 6 you have to either type in RUN"D:EX5" OR RUN"C:", whichever is appropiate. The same for EX.6. This is because we stopped EX.4 & EX.5 for you to try out.

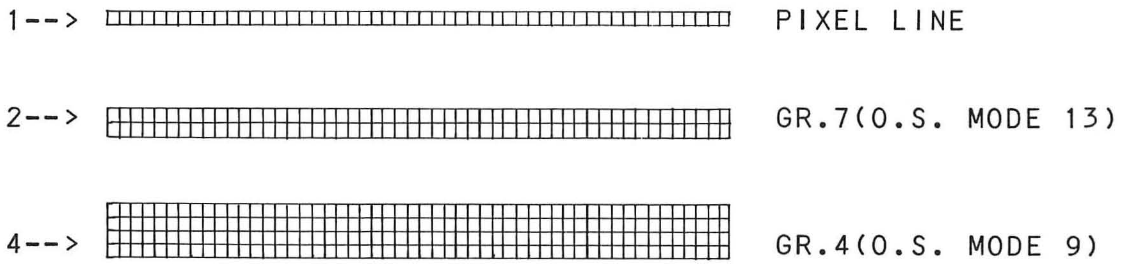
```
10 GRAPHICS 17:? #6;"  EXAMPLE 5"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
500 REM THIS IS A SUBROUTINE TO ALLOW YOU TO
PROGRAM WITH BIG LETTERS.SEE MANUAL FOR DIRE
CTIONS.
700 GRAPHICS 0:GOSUB 10000
710 REM PUT THE REST OF YOUR PROGRAM  STARTIN
G ON THIS LINE
720 LIST :END
10000 TRAP 10040
10010 DL=PEEK(560)+256*PEEK(561)
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
```

```
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1+70
10016 IF I=6 THEN POKE DL+5,Z2+2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,70,74,158,6,6,6,6,6,6,
6,6,6,6,6,6,6,6,6,6,6,6,65,32,124,0,0,0
10040 RETURN
```

EXAMPLE 6

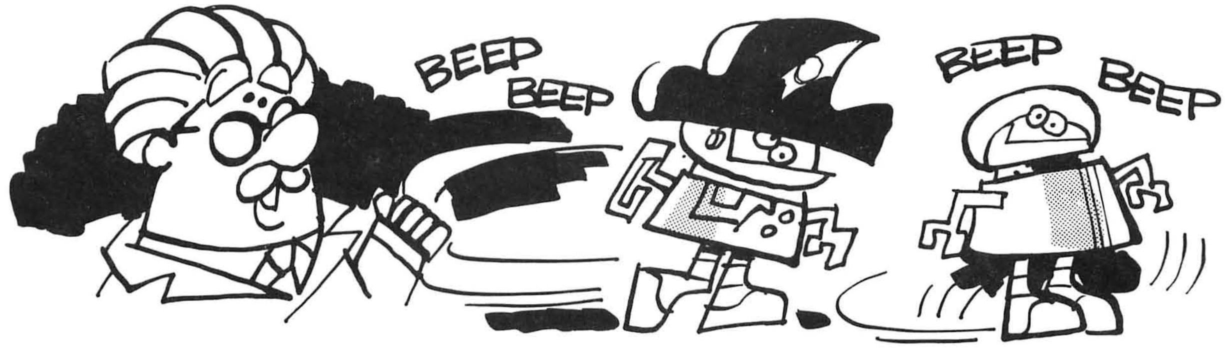
This example is like EX.2, except that it remembers your current DL. This way you can slowly change the DL until you like it. Then copy the numbers and use them in EX.3 in your own programs. After you create a screen, if you want to BREAK the program just type LIST. This will flow the programs data over the DL you just created to better see each region. To restart the program, just type RUN. Other commands are: press OPTION to go to EX.7; press START to do another screen using the current DL numbers; press SELECT to get back to the original numbers in case your custom screen goes crazy, or you get lost in what you're doing.

GRAPHICS MODE LINES AND PIXEL LINES



(Up to 16 Pixels per Mode Line in O.S. Modes 5 & 7 (=GR.2))

FIGURE 4



GOING CRAZY?

Yes, remember we said before that if you put in enough modes in the DL so that too many pixel lines are used up, the TV will start to roll. This is simply because you are saying in the DL to "Draw on the screen more than you have room for". Conversely, if you put too few lines on the screen, the picture will shrink. Simply change the amount of modes or the type of mode(ie., change several modes like 2 which use 8 lines of pixels per mode line of 2(GR.0), to ones like mode 9 which use only 4 pixel rows per mode 9(GR.4) line). Now,if this mode line vs. pixel line stuff has got you lost, we are going to have one more figure (figure 4).This one shows clearly (we hope) the difference between the two.

```
1 GRAPHICS 17:? #6;"  EXAMPLE 3"
3 FOR W=1 TO 230:POKE 710,W
4 FOR ZR=1 TO 10:NEXT ZR
5 NEXT W
7 DIM ZX$(135)
20 GRAPHICS 0:? "THIS EXAMPLE ALLOWS YOU TO C
HANGE A   MODE 0 DISPLAY LIST!"
30 ? "PRESS START TO BEGIN":? "WRITE DOWN THE
 NUMBERS IN THE DL WHEN YOU LIKE IT!"
55 DATA 112,112,112,66,96,144,2,2,2,2,2,2,2,2
,2,4,4,4,4,4,2,2,2,2,2,2,2,2,2,65,32,156,0
56 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
57 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
58 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
62 POKE 53279,8
65 Z=PEEK(53279)
70 IF Z=6 THEN 100
80 GOTO 65
100 GRAPHICS 0:POSITION 2,3
105 LIST 55,58
110 POSITION 2,18:? "CHANGE THE VALUES YOU WA
NT BY USING   THE CURSOR CONTROLS "
120 POSITION 10,0
125 INPUT ZX$
130 POSITION 0,2
```

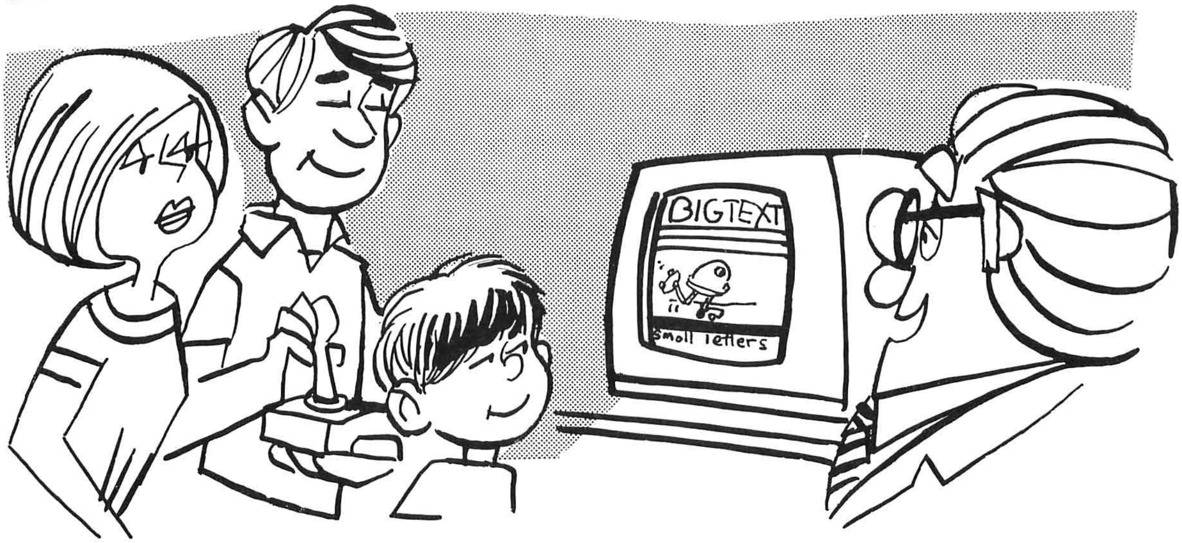
```
140 POKE 842,13
142 POSITION 2,17:? "CONT"
144 POSITION 0,2:STOP
150 POKE 842,12
200 RESTORE 55
210 I=0
215 TRAP 400
217 I=1:GRAPHICS 7+16
218 DL=PEEK(560)+256*PEEK(561)
219 A=PEEK(DL+4):B=PEEK(DL+5)
220 READ G:IF G=0 THEN 233
221 POKE DL+I-1,G
222 IF I=5 THEN POKE DL+4,A
223 IF I=6 THEN POKE DL+5,B
230 I=I+1:GOTO 220
233 FOR N=1 TO 900:NEXT N:POKE 87,0
240 POSITION 2,1:? "PRESS OPTION TO GO ON":?
"PRESS START TO DO ANOTHER SCREEN"
243 ? "PRESS SELECT FOR THE ORIGINAL DISPLAY
LIST"
245 TRAP 5
250 POKE 53279,8
260 Z=PEEK(53279)
265 IF Z=5 THEN 400
270 IF Z=6 THEN 57
280 IF Z=3 THEN POKE 764,12:RUN "D:EX7"
300 GOTO 260
400 TRAP 400:GRAPHICS 0:POSITION 2,4
405 ? "55 DATA 112,112,112,66,"";A;"","";B;"",2,2
,2,2,2,2,2,2,2,4,4,4,4,4,2,2,2,2,2,2,2,2,2,65
,32,156,0"
406 ? "56 DATA 0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
407 ? "57 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
408 ? "58 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
410 GOTO 110
```

EXAMPLE 7

This one is for you to figure out. Some clues: When the "HELLO'S" stop, look at all the modes on the screen. Notice towards the bottom, the mode eight area (obvious by the red and blue pixels). Since the computer displays the data according to whatever data is next "in line", the statements about press OPTION, START, or SELECT end up in this mode 8 area (even though you can't see the statements, you can still use them). Then we hit a 4k boundary, so a LMS instruction was used but to make it interesting, we used the same address as the top of the data. Look at the Display List for GR.8 using EX.1. See the 79 (LMS) instruction in the middle.

```
 2 GRAPHICS 17:? #6;"   EXAMPLE 7"
 4 FOR W=1 TO 230:POKE 710,W
 5 FOR ZR=1 TO 10:NEXT ZR
 6 NEXT W
 7 DIM ZX$(135)
 10 TRAP 40000:TRAP 10
 20 GRAPHICS 0:? "THIS EXAMPLE ALLOWS YOU TO C
HANGE A    MODE 0 DISPLAY LIST!"
 30 ? "PRESS START TO BEGIN":? "WRITE DOWN THE
NUMBERS IN THE DL WHEN YOU LIKE IT!"
40 POKE 710,0
55 DATA 112,112,112,66,96,144,2,2,2,2,6,6,7,7
,14,14,14,14,14,14,14,2,2,15,15
56 DATA 15,15,15,15,15,79,96,144,15,6,7,2,2,6
5,32,156,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65
,32,156,0
 57 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0, 0,0,0,0,0,65,32, 156,0
 58 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
 62 POKE 53279,8
 65 Z=PEEK(53279)
 70 IF Z=6 THEN 100
 80 GOTO 65
 100 ? "K":POSITION 2,3
 105 LIST 55,58
 110 POSITION 2,10:? "CHANGE THE VALUES YOU WA
NT BY USING    THE CURSOR CONTROLS "
115 ? "SCREEN."
120 POSITION 10,0
125 INPUT ZX$
130 POSITION 0,2
140 POKE 842,13
142 POSITION 2,17:? "CONT"
144 POSITION 0,2:STOP
150 POKE 842,12
200 RESTORE 55
210 I=0
215 TRAP 400
217 I=1:GRAPHICS 7+16:POKE 710,0
218 DL=PEEK(560)+256*PEEK(561)
219 A=PEEK(DL+4):B=PEEK(DL+5)
220 READ G:IF G=0 THEN 233
221 POKE DL+I-1,G
222 IF I=5 THEN POKE DL+4,A
223 IF I=6 THEN POKE DL+5,B
230 I=I+1:GOTO 220
233 FOR W=1 TO 900:NEXT W:POKE 87,0:LIST :GOS
UB 5000
240 ? "PRESS OPTION TO GO ON":? "PRESS START
TO DO ANOTHER SCREEN"
243 ? "PRESS SELECT FOR THE ORIGINAL DISPLAY
LIST"
```

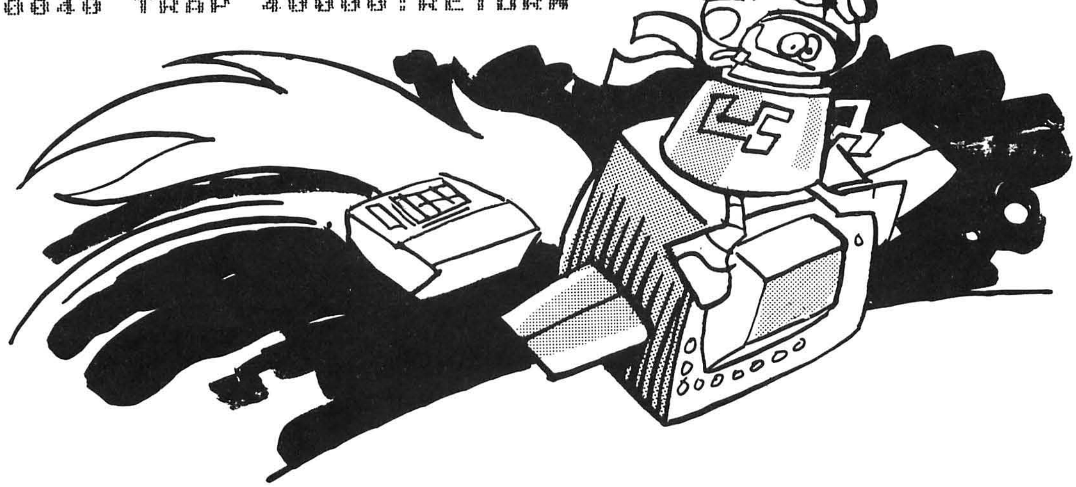
```
245 TRAP 10
250 POKE 53279,8:J=0
260 Z=PEEK(53279)
265 IF Z=5 THEN 400
270 IF Z=6 THEN 57
280 IF Z=3 THEN POKE 764,12:RUN "D:EX8"
300 GOTO 260
400 TRAP 40000:GRAPHICS 0:POSITION 2,4
405 ? "55 DATA 112,112,112,66,";A;",";B;",2,2
,2,2,6,6,7,7,14,14,14,14,14,14,14,2,2,15,15"
406 ? "56 DATA 15,15,15,15,15,79,96,144,15,6,
7,2,2,65,32,156,0,0,0,0,0,0,0,0,0"
407 ? "57 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
408 ? "58 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
410 GOTO 110
5000 J=J+1:? "HELLO":IF J<50 THEN 5000
5010 RETURN
```

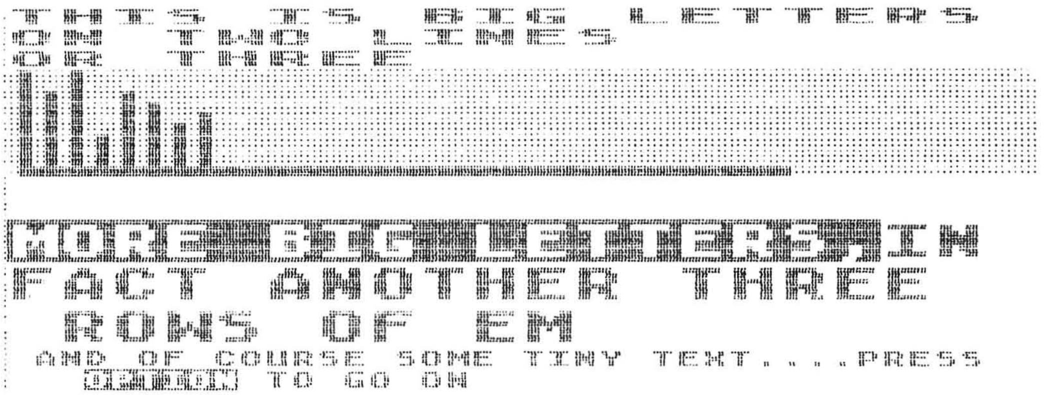


EXAMPLE 8

Get your wife, husband, kids, or friends. You are about to prove what home computers are good for. Remember that the LMS numbers (two numbers after the LMS value) tell the computer where to get the screen data. Well, this example just uses the joystick to change this address so you can see most of the memory in your machine in both graphics and text, some of it changing before your eyes. Moving the joystick forward will move up to the top of memory(remember you were almost at the top to start). Moving down will go almost to the beginning of memory. If we were to look at the very bottom we would crash(stop) the computer because we would interfere with the O.S. OK.....it wasn't that great....send them all back. Press OPTION to go on.

```
10 GRAPHICS 17:? #6;"  EXAMPLE 3"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
500 REM THIS IS A SUBROUTINE FOR YOU TO USE T
O LOOK AT MEMORY BOTH FROM A GRAPHICS AND TEX
TUAL VIEWPOINT!
520 REM USE YOUR JOYSTICK AND MOVE UP OR DOWN
 IN MEMORY !!! PRESS OPTION TO GO ON.
700 GRAPHICS 0:GOSUB 10000
705 I=Z1:LIST :POKE 53279,8
707 FOR F=1 TO 50:? "USE JOYSTICK":NEXT F
710 ST=STICK(0):Z=PEEK(53279):IF Z=3 THEN POK
E 764,12:RUN "D:EX9"
715 IF ST=15 THEN 710
720 IF ST=14 THEN Z2=Z2+1
730 IF ST=13 THEN Z2=Z2-1
740 IF Z2<10 THEN Z2=10
745 IF Z2>PEEK(106) THEN Z2=PEEK(106)
750 POKE DL+5,Z2
760 GOTO 710
10000 TRAP 10040
10010 DL=PEEK(560)+256*PEEK(561)
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1
10016 IF I=6 THEN POKE DL+5,Z2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,66,64,156,2,2,2,2,2,2,
6,6,6,6,6,6,6,6,10,10,10,10,10,10,2,2,2,2,2,6
5,32,124,0,0,0
10040 TRAP 40000:RETURN
```





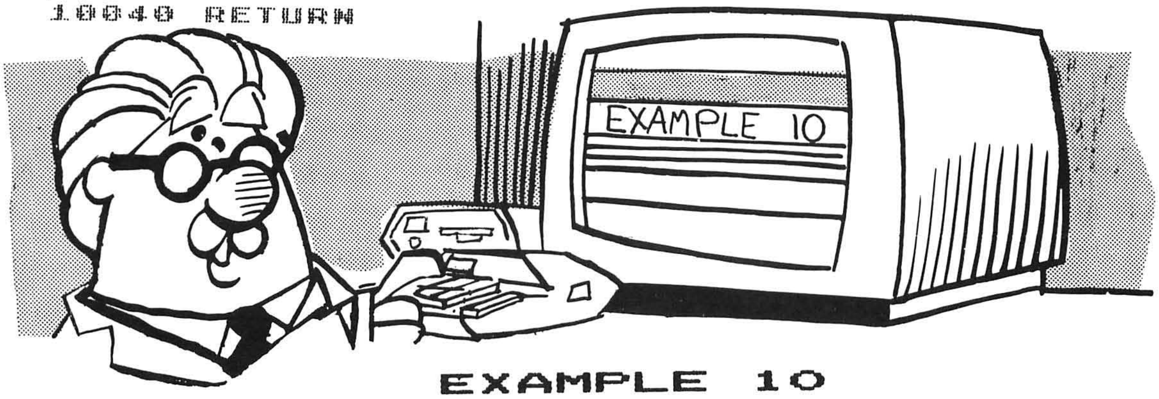
```
THIS IS BIG LETTERS
ON TWO LINES
OR THREE

MORE BIG LETTERS, IN
FACT ANOTHER THREE
 ROWS OF EM
 AND OF COURSE SOME TINY TEXT....PRESS
    START TO GO ON
```

EXAMPLE 9
A PRACTICAL EXAMPLE

Finally an actual example where we PLOT and PRINT stuff. Now the problem with custom DL's is not choosing the numbers as you have already seen. That was easy. But now we have got to get the PLOT or PRINT statements to work with the new screen we have setup. When we start out a custom DL with a POKE 87,GR.mode # or a Graphics statement like GR.0, the O.S. thinks that it is in that mode, not the one on the screen. If the areas we are plotting or writing to are within the normal limits, then we can just experiment to find what mode line we want to PLOT/PRINT to. That is what this example does. It's hit or miss, but it works. Plot something and if it's not where you want change the PLOT statement. Same with text, using the POSITION statement. There is nothing wrong with the "hit and miss" method, but our next example will give you the precise numbers to accurately place yo.ur information on the screen.

```
10 GRAPHICS 17:? #6;"    EXAMPLE 9"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
500 REM THIS IS A SUBROUTINE FOR YOU TU USE I
N YOUR OWN PROGRAMS. SEE MANUAL FOR DIRECTION
S.
505 GRAPHICS 0
510 GOSUB 10000
515 POKE 87,1:POSITION 0,0:? #6;"THIS IS BIG
LETTERS"
520 ? #6;"ON TWO LINES";? #6;"OR :HREE"
550 POKE 87,4:COLOR 1:PLOT 1,5:DRAWTO 1,15:DR
AWTO 60,15:PLOT 15,15:DRAWTO 15.18:PLOT 3,15:
DRAWTO 3,8
555 PLOT 5,15:DRAWTO 5,6:PLOT 7,15:DRAWTO 7,1
2:PLOT 9,15:DRAWTO 9,8:PLOT 11,15:DRAWTO 11,9
:PLOT 13,15:DRAWTO 13,11
560 POKE 87,2:POSITION 0,9:? #6;"MORE BIG LET
TERS,in fact an other three    rows of em"
570 POKE 87,0:POSITION 1,6:? "AND OF COURSE S
OME TINY TEXT....PRESS    START TO GO ON"
```

```
730 POKE 53279,0
740 Z=PEEK(53279)
750 IF Z=3 THEN POKE 764,12:RUN "D:EX10"
760 GOTO 740
10000 TRAP 10040
10010 DL=PEEK(560)+256*PEEK(561)
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1
10016 IF I=6 THEN POKE DL+5,Z2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,70,64,156,6,6,9,9,9,9,
9,9,9,9,9,9,7,7,7,7,2,2,2,2,2,2,65,32,124,0,0
,0
10040 RETURN
```



EXAMPLE 10

This example is the big time! We have placed all of the previous abilities in this one, but added the information we promised in a chart on the screen. RUN the example and follow it through with us.

The program still uses DATA statements to experiment with breaking up the screen into mixed graphic and text modes. Use the cursor controls as before to edit the list, press RETURN to see what you have created. If you used PLOT modes at the top of the screen, you won't see that the controls are different, so pay attention. When the split screen appears, you can :

1) Press START (as before) to go change the current DL.
2) Press SELECT (as before) to go back to the original DL.
3) Press OPTION and START at exactly the same time. This will get you to the last example.
4) Press OPTION for a chart of the custom DL you created.

Now get to the new chart by pressing RETURN and then OPTION as explained above. The first column is the position of each graphics mode you choose:1,2,3...last(maximum of 20!). The second column tells you what number to POKE into location 87. This is simply the regular graphic mode number(0-11). Column three is the O.S.mode # which was the number you used in the DL itself. Yes, it is confusing to use both numbers in the same program when they mean the same thing to us, but remember that when you use a regular GR. mode # you are talking to the BASIC Interpeter which then changes the number you give into the O.S. mode number . We are just bypassing the interpeter to get the special screens that it doesn't allow!

The next column is the number of lines of this mode you asked for, ie. how many times you included it in the DL. This is useful because if you have say six lines of a graphic region and you try to plot to 7,7 you will go out of that mode and into the next, causing garbage to appear on the screen. The same goes for text regions. This column will provide a quick reference. Then comes the two columns of POKE numbers for locations 88 & 89. These two values combined with 87 will fool the computer into acting as if each region starts with 0,0 at it's upper left corner. This makes your PRINT and PLOT statements work like normal. The difference is you PRINT/PLOT to EACH region as if it started with 0,0 at it's upper left corner.

Write down all this information or use in your custom routine, or to modify one of our examples.

The chart also lists the number of lines of pixels you filled on the screen. If it doesn't come out close to 192, or if you want to make some other change, you can go back to re-edit the DL by pressing START. If you have too many lines, then remove modes lines, or add some if you're short.

IF YOU'RE STILL LOST

Use the parts of this program that you understand now, and come back to study and experiment with the rest of it later. Basically, you should eventually understand the following to use this program. All the rest is "additional information":

1) Use EX.10 to get;
   a)the DL to look about right.
   b)the numbers to POKE into 87, 88, and 89.

2) Take EX.11 and change the POKE values to what EX.10 will give you when you tranfer the numbers from EX.9 into the DL data of EX.10.
3) Try to change the graphics and text we used to your own.
4) If you have more regions than the example has, add more. Likewise if you have less.
5) Remember to treat each area on the screen like it was a small screen by itself... don't PLOT or PRINT too far.

```
1 GRAPHICS 17:? #6;"   EXAMPLE 12"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR
4 NEXT W
5 DIM P(220):DIM NUM(20):DIM MODE(20):DIM RAM
(15):DIM START(20):DIM P88(20):DIM P89(20):DI
M LIN(15)
 7 DIM BAM(15)
 10 TRAP 490
 20 DIM ZX$(135)
 30 GRAPHICS 0:? "THIS EXAMPLE ALLOWS YOU TO C
HANGE A    MODE 0 DISPLAY LIST!"
 40 ? "PRESS START TO BEGIN.":? "WRITE DOWN TH
E NUMBERS IN THE DL WHEN YOU LIKE THEM!"
 50 DATA 112,112,112,66,64,156,2,2,2,2,2,2,2,2
,2,4,4,4,4,4,2, 2,2,2,2,2,2,2,2,65,32,156,0
 60 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
 70 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
 80 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0
 90 POKE 53279,0
 100 Z=PEEK(53279)
 110 IF Z=6 THEN 140
 120 GOTO 100
 130 TRAP 490
 140 GRAPHICS 0:POSITION 2,3
 150 LIST 50,80
 160 POSITI0N 2,18:? "CHANGE THE VALUES YOU WA
NT BY USING     THE CURSOR CONTROLS."
 170 ? "PRESS RETURN TO SEE THE NEW SCREEN."
 180 POSITION 10,0
 190 INPUT ZX$
 200 POSITION 0,2
 210 POKE 842,13
 220 POSITION 2,17:? "CONT"
 230 POSITION 0,2:STOP
 240 POKE 842,12
```

```
250 RESTORE 50:MAX=0
260 TRAP 490:READ S:IF S>15 THEN 260
265 IF S>MAX THEN MAX=S
270 IF S=0 THEN 281
280 GOTO 260
281 ON MAX GOTO 285,285,285,285,285,285,285,2
85,285,285,286,287,287,288,288
285 USE=0:GOTO 290
286 USE=22:GOTO 290
287 USE=23:GOTO 290
288 USE=24
290 RESTORE 50
300 TRAP 490
310 I=1:GRAPHICS USE
320 DL=PEEK(560)+256*PEEK(561)
330 A=PEEK(DL+4):B=PEEK(DL+5)
340 READ G:IF G=0 THEN 390
350 POKE DL+I-1,G
360 IF I=5 THEN POKE DL+4,A
370 IF I=6 THEN POKE DL+5,B
380 I=I+1:GOTO 340
390 FOR Q=1 TO 100:NEXT Q:POKE 87,0
400 POSITION 2,1:? "PRESS OPTION FOR CHART OF
PLOT VALUES WHEN DONE."
410 ? "PRESS SELECT FOR THE ORIGINAL DISPLAY
LIST #'S."
420 ? "PRESS START TO CHANGE CURRENT DL
NUMBERS."
425 ? "PRESS START & OPTION TO RUN LAST
EXAMPLE."
430 POKE 53279,8
440 Z=PEEK(53279)
450 IF Z=5 THEN 490
460 IF Z=6 THEN 70
465 IF Z=2 THEN RUN "D:EX11"
470 IF Z=3 THEN 550
480 GOTO 440
490 TRAP 40000:GRAPHICS 0:POSITION 2,4
500 ? "50 DATA 112,112,112,66,";A;",";B;",2,2
,2,2,2,2,2,2,2,4,4,4,4,4,2,2,2,2,2,2,2,2,2,65
,32,156,0"
510 ? "60 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
520 ? "70 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0 ,0,0,0,0,0,0,0,0,65,32,156,0"
530 ? "80 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,65,32,156,0"
540 GOTO 160
550 RESTORE 50:POKE 559,0
560 J=1:K=1:L=2:P(1)=PEEK(DL+3)-64
570 READ PR:P(L)=PR:IF P(L)>15 THEN 570
580 IF P(L)=0 THEN 620
```

```
590 IF P(L)<>P(L-1) THEN MODE(K)=P(L-1):NUM(K
)=J:J=1:L=L+1:K =K+1:GOTO 570
600 J=J+1:L=L+1:GOTO 570
620 MODE(K)=P(L-1):NUM(K)=J
630 RAM(2)=40:RAM(3)=40:RAM(4)=40:RAM(5)=40:R
AM(13)=40:RAM(14)=40:RAM(15)=40
633 LIN(2)=8:LIN(3)=10:LIN(4)=8:LIN(5)=16:LIN
(13)=2:LIN(14)= 1:LIN(15)=1
634 LIN(6)=8:LIN(7)=16:LIN(8)=8:LIN(9)=4:LIN(
10)=4:LIN(11)=2:LIN(12)=1
635 RAM(6)=20:RAM(7)=20:RAM(8)=10:RAM(9)=10:R
AM(10)=20:RAM(1 1)=20:RAM(12)=20
640 FOR H=1 TO K
644 BAM(2)=0:BAM(3)=40:BAM(4)=40:BAM(5)=40:BA
M(13)=7:BAM(14)=40:BAM(15)=8
645 BAM(6)=1:BAM(7)=2:BAM(8)=3:BAM(9)=4:BAM(1
0)=5:BAM(11)=6:BAM(12)=40
660 IF MODE(H)<MODE(H-1) THEN 680
670 SAV=MODE(H)
680 NEXT H
690 IF SAV>6 THEN 720
700 IF SAV>0 THEN SAV=6:GOTO 720
710 IF SAV<>0 THEN ? "ERROR AT 710..SAV SHOUL
D BE 0":STOP
720 MAXRAM=RAM(SAV)
725 FOR M=1 TO K:M2=0:M3=0
735 FOR S=1 TO M-1
740 IF RAM(MODE(S))=20 THEN 760
750 IF RAM(MODE(S))=10 THEN 770
755 GOTO 780
760 M2=NUM(S)+M2:GOTO 780
770 M3=NUM(S)+M3
780 NEXT S
783 R=0
785 FOR W=1 TO M-1:R=R+NUM(W):NEXT W
790 CHRPOS=A+B*256+(R*MAXRAM-M2*(MAXRAM-20)-M
3*(MAXRAM-10))
795 IF M=1 THEN CHRPOS=A+B*256
800 START(M)=CHRPOS
810 NEXT M
820 FOR Z=1 TO K:P89(Z)=INT(START(Z)/256):P88
(Z)=START(Z)-P89(Z)*256:NEXT Z
825 SUM=0
830 FOR Q=1 TO K:SUM=NUM(Q)*LIN(MODE(Q))+SUM:
NEXT Q
840 GRAPHICS 0:POSITION 0,0:? "┌──────────────────
─────────────────────┐"
850 POSITION 0,1:? "│POS│POKE│O.S. │# LINES│
POKE │ POKE │"
860 POSITION 0,2:? "│ON │INTO│MODE │OF THIS│
INTO │ INTO │"
```

```
870 POSITION 0,3:? "|SCR| 87 | #    | MODE   |
  88   |   89   |"
880 POSITION 0,4:? "└──────────────────────────────────────────
─────────────────────┘"
890 FOR W=1 TO K
895 SD=BAM(MODE(W))
898 POSITION 1,4+W:? W;
901 IF SD=40 THEN 903
902 POSITION 6,4+W:? SD;:GOTO 904
903 POSITION 6,4+W:? "NONE";
904 POSITION 12,4+W:? MODE(W):POSITION 18,4+W
:? NUM(W):POSITION 25,4+W:? P88(W);
905 POSITION 33,4+W:? P89(W)
910 NEXT W:? "THE NUMBER OF PIXEL LINES USED=
";SUM;".":? "SUGGEST YOU CHANGE EX UNTIL 192
PIXEL LINES RESULTS!";
915 ? " COPY NUMBERS OR PRESS  START TO REDO
CURRENT EX.":? "GRAPHICS MODE TO USE IN EX.11
 IS ";USE
 920 POKE 559,34
 930 POKE 53279,8
 940 Z=PEEK(53279)
 950 IF Z=6 THEN 140
 960 GOTO 940
 999 END
```

EXAMPLE 11
LAST EXAMPLE

This PLOT is just like EX.9, but now we add POKEs to 88 and 89 to fool the computer into acting as if the upper corner of each mode was position 0,0. The POKE to 87 then says "act as if you're in a regular graphics mode #" (your input). Now we just PLOT or PRINT as if each section was a small screen with 0,0 at the top like normal. If you PLOT too far, the next region down will look funny. We include the code for this example (minus a few lines you don't need) since this example is all you really need to understand. Our other examples will do all the calculating for you. Make sure you change the 5th and 6th numbers in the data list to the correct values for your computer, as explained above.

```
10 GRAPHICS 17:? #6;"   EXAMPLE 11"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR
40 NEXT W
50 ? #6;"   Last ExamPle!"
60 FOR W=1 TO 230:POKE 708,W:POKE 709,W:POKE
710,W:POKE 711,W
```

```
65 FOR ZR=1 TO 5:NEXT ZR
70 NEXT W
80 GRAPHICS 17:? #6;"TO MAKE THIS EXAMPLE"
90 ? #6;"WORK, YOU WILL HAVE "
100 ? #6;"TO CHANGE THE VALUES"
110 ? #6;"EACH TIME POKE 88 OR"
120 ? #6;"89 IS USED. THIS IS "
130 ? #6;"DONE BY [illegible] THE"
140 ? #6;"VALUES IN THE DL OF "
150 ? #6;"THIS EX. INTO [illegible]."
160 ? #6;"the chart will give "
170 ? #6;"you the values for  "
180 ? #6;"each mode area.[illegible]"
190 ? #6;"[illegible]"
200 POKE 764,255
210 IF PEEK(764)<>12 THEN 210
400 REM ****************************************
500 REM THIS IS A SUBROUTINE FOR YOU TO USE A
S A PRACTICE EXAMPLE IN PLOTTING AND WRITING
TO THE AREAS OF THE
503 REM SCREEN THAT YOU SET UP IN THE PREVIOU
S EXAMPLE. JUST PLUG IN THE POKE VALUES FOR L
OCATIONS 87,88,&89
504 REM AND CHANGE THE NUMBERS IN THE DATA ST
ATEMENT TO THOSE YOU WROTE DOWN FROM THE LAST
EXAMPLE . FINALLY.YOU
505 REM SHOULD CHANGE THE PLOT AND PRINT STAT
EMENTS TO THE ONES YOU WANT. HAVE FUN...SEE Y
OU IN THE NEXT TUTORIAL.
506 REM ****************************************
507 GRAPHICS 0:REM [illegible], REMEMBER TO
USE THE GRAPHICS MODE THAT THE CHART GAVE YO
U.
510 GOSUB 10000
515 POKE 87,1:POKE 88,64:POKE 89,156:POSITION
 0,0:? #6;"THIS IS BIG LETTERS"
520 ? #6;"ON TWO LINES":? #6;"OR THREE"
540 POKE 88,124:POKE 89,156
550 POKE 87,4:COLOR 1:PLOT 1,0:DRAWTO 1,10:DR
AWTO 60,10:PLOT 15,10:DRAWTO 15,5:PLOT 3,10:D
RAWTO 3,3
555 PLOT 5,10:DRAWTO 5,1:PLOT 7,10:DRAWTO 7,7
:PLOT 9,10:DRAWTO 9,3:PLOT 11,10:DRAWTO 11,4
:PLOT 13,10:DRAWTO 13,6
557 POKE 88,234:POKE 89,156
560 POKE 87,2:POSITION 0,0:? #6;"[illegible]
[illegible]in fact another three   rows of em"
565 POKE 88,58:POKE 89,157
570 POKE 87,0:POSITION 0,0:? "AND OF COURSE S
OME TINY TEXT....PRESS   RETURN TO GO ON"
730 POKE 53279,8
740 Z=PEEK(53279)
750 IF Z=6 THEN 20000
```

```
760 GOTO 740
10000 TRAP 10040
10010 DL=PEEK(560)+256*PEEK(561)
10011 Z1=PEEK(DL+4):Z2=PEEK(DL+5)
10012 I=1
10013 READ A:IF A=0 THEN 10040
10014 POKE (DL+I-1),A
10015 IF I=5 THEN POKE DL+4,Z1
10016 IF I=6 THEN POKE DL+5,Z2
10020 I=I+1:GOTO 10013
10030 DATA 112,112,112,70,64,156,6,6,9,9,9,9,
9,9,9,9,9,9,9,7,7,7,7,2,2,2,2,2,2,2,65,32,124
,0,0,0
10040 RETURN
20000 GRAPHICS 0:POKE 710,0
20001 ? " THANKS FOR BUYING FROM SANTA CRUZ E
DUCATIONAL SOFTWARE!!!";:GOTO 20001
```

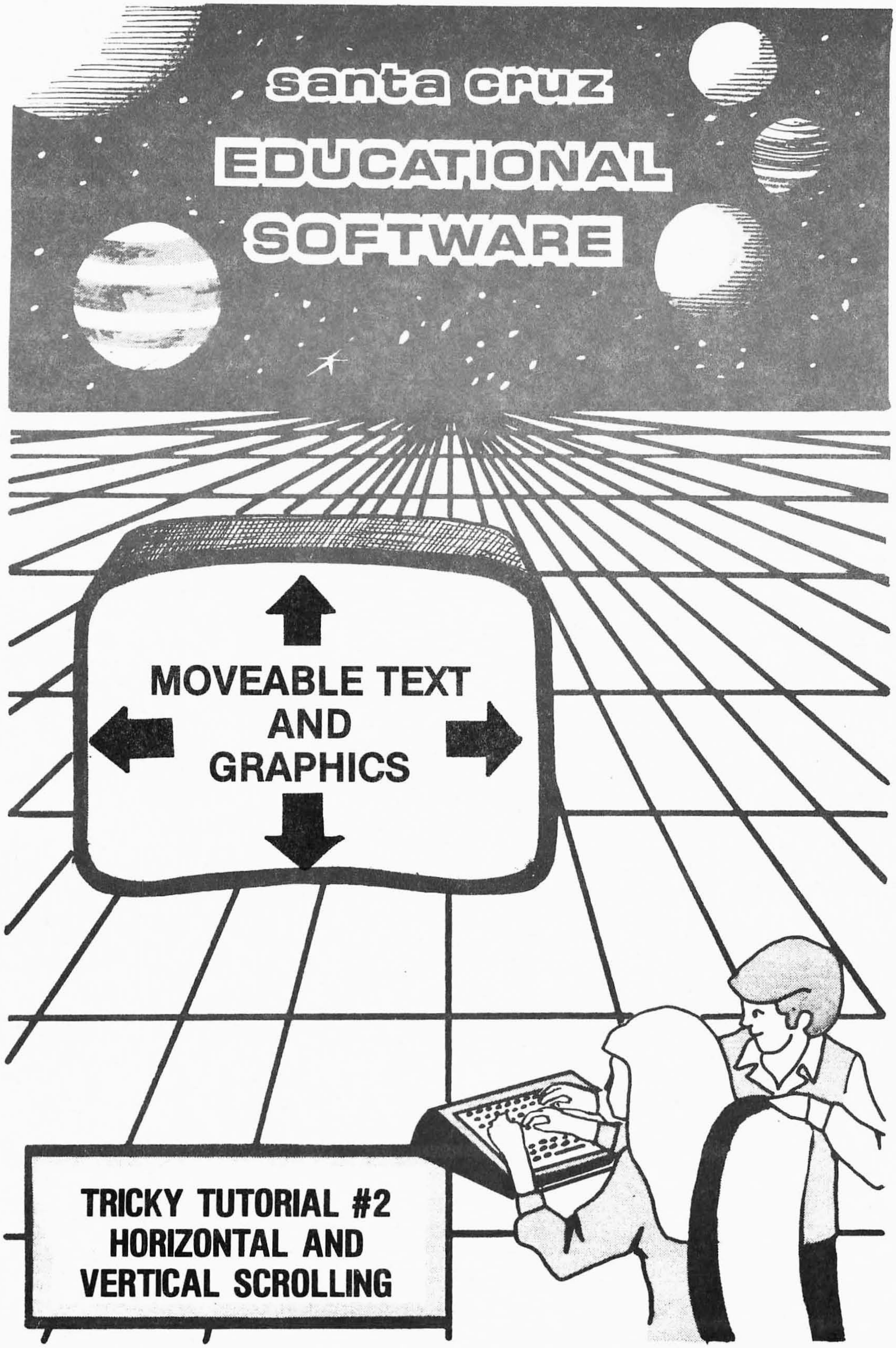
Final Notes

Sometimes when you do your custom screens you probably will have your text or plot suddenly shift over like this
                                        to the other side of the screen. This is because your Display List is going along at,say, modes using 40 bytes per line when you shift and use three lines of a mode using 20 bytes per mode. The system is still trying to place the data where it used to go, so if you use 40's then switch to 20's or 10's, keep it in increments of the largest value. For example, if you use 40 byte modes then if you have 10 byte modes either use 4, 8, 12 , etc., so that the computer can find 40 at a time. DON'T FOOL MOTHER ATARI!

We also want to warn you that EX.10 occasionally can be fooled by unusual inputs, so if the chart seems incorrect it may be. This is not a serious problem because you are going to learn to do all the calculations yourself, AREN'T YOU???

The other modes in the chart (FIG.1) that aren't explained are for things like lowercase descenders and multi-colored letters. See, I told you not to ask. These would be good topics for another Tricky Tutorial(tm). Well, maybe SOMEDAY.....

Well that's it. If even a new programmer will experiment with these examples, he or she will at least be able to make some nice custom screens without having to understand it all.



santa cruz EDUCATIONAL SOFTWARE

MOVEABLE TEXT AND GRAPHICS

TRICKY TUTORIAL #2 HORIZONTAL AND VERTICAL SCROLLING

HORIZONTAL AND VERTICAL SCROLLING

This program will teach you the basic principles of moving text or graphics on the screen. Movement can be in any direction that you choose. A total of eighteen examples are provided plus a 15 page manual to explain them.

Each example, including those with some assembly language subroutines, are designed to be easily placed in your own programs. Anyone with some knowledge of the BASIC programming language can use the examples now, and later study the manual in more detail. This program is part of a continuing series and goes particularly well with #1 - Display Lists.

REQUIRES 16K OR 24K IF YOU HAVE DISK.

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

Educational Software

presents

TRICKY TUTORIAL #2

Horizontal & Vertical Scrolling

TRICKY TUTORIAL #2

HORIZONTAL & VERTICAL SCROLLING

by
Robin Sherer

INTRODUCTION - Hi. I'm going to start out this tutorial with a sales pitch for one of the other tutorials I teach. As phony as this sounds, I can't help it. To fully understand how to scroll you need to also study Display Lists, Tricky Tutorial(tm) # 1. However, if you just want to be able to do the different types of scrolling in your own programs, THIS WILL DO IT FOR YOU. I'll tell you what to modify without great explaination. All of the examples have different features. Some are faster, some smoother in their scrolling. It's up to you to take parts of our examples and make them a part of your programs.

HOW TO LOAD

TAPE....

Place the tape in your recorder, label side up. Make sure the tape is rewound, and the BASIC Cartridge is in place, and always reset the counter to zero. Push PLAY on the recorder and type RUN"C: and press RETURN. If the program won't start to load, try positioning forward or backwards a little. The easiest way to find the beginning is to listen to the "noise" on the tape with a regular recorder. When you find the steady tone, you have the beginning of the program. We recommend you write down the number on your recorder's counter as each program example starts, this will make it easier to find each part later on.

DISK....

To load and run the disk, first turn on your disk drive. When the busy light goes out, place the disk in the drive. Now turn on the computer, with the BASIC Cartridge in place and the program will load each part and run by itself.

Any defective tapes or disks should be returned to:

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

ALL RIGHT STUDENTS! LET'S BEGIN WITH (SURPRISE!)

EXAMPLE 1

Basic Ideas

If you haven't already, run the first example. To get to the next example each time, you only have to press START. Note - cassette examples should run themselves when loading is done. If not, type RUN and press RETURN. You may have to rewind some examples to the end of the last example and try several reloads. Remember that this tape has many examples, so it is harder to LOAD them all than some small games.

Plug in a Joystick into Port 1 on the front of your console. You are now looking at a listing of the example that is running. I listed it as an easy way to get text on the screen to scroll. This example, and all others where we use LIST to get text on the screen, will work exactly the same if you take out the LIST statement and substitute your own PRINTing or PLOTing. Do this as soon as you feel you are beginning to understand the method.

If you move your Joystick to the left or right, the text will move in that direction. Look closely at the text as it moves. Notice that it moves one character at a time in either direction. This is called COARSE SCROLLING. Later, we will learn how to move one pixel at a time which is called FINE scrolling. Eventually we will combine both methods for (guess what?) COMBINED SCROLLING.

Take a Break.....

OK! Back to Work:

Warning!

In most of our examples we wanted to keep the amount of BASIC code short, so if you move too far in any direction, the POKEs we are doing may cause the program to error and stop. To avoid this problem in actual examples, you will have to make sure you don't try to POKE a number larger than 255 into memory. Some of examples will automatically do this for you. To restart a stopped example, just press RESET and type RUN.

Here is the code for Example 1, a simple coarse horizontal scroll:

```
10 DL=PEEK(560)+256*PEEK(561)
20 DL4=DL+4:DL5=DL+5
30 PDL4=PEEK(DL4)
40 ST=STICK(0)
50 IF ST=11 THEN PDL4=PDL4+1
60 IF ST=7 THEN  PDL4=PDL4-1
70 POKE DL4,PDL4
80 FOR W=1 TO 50 :NEXT W
90 GOTO 40
```

THAT'S IT!

Another note - Yes we know that if you list Example 1 it has more lines than this. The missing lines above were put in Example 1 to make it have a title and call Example 2 when you are ready. The same differences will occur with my other examples.

To better understand this method, first a few basics from the Display List Tutorial. The Display List is a set of instructions in memory that the computer uses to find out what to put on the screen. By changing it, we can do many programming tricks that are not only interesting to look at, but useful. Scrolling is one of those tricks.

Line 10 above looks at locations 560 & 561 by use of the PEEK command. The values in these two memory locations combine to give you the location (called the address) of the first byte of the Display List. They are combined into the address as line 10 shows. ANYTIME you have a 2 byte address, the method in line 10 is used. Whenever you use the "GRAPHICS #" command, the computer creates a standard Display List somewhere in memory and puts the location where the DL starts in memory locations 560/561.

Now that we know where the DL (Display List) is, line 20 stores the address of the low part (DL4) and high part (DL5) of a number that is always the 5th and 6th numbers in the DL. Note that we call the 5th number into the DL, DL4, because it is equal to the value at DL plus 4 more, or the 5th number from the start. Confused? Then just remember that these two numbers tell the computer where to go get the data for the screen.

IMPORTANT!

You must understand what these two numbers do. Scrolling is based on changing these values. Let's look at a sample GR. 0 Display List :

```
112 112 112 66 95 34 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 65 89 56
```

The first three numbers are 112's that tell the computer to display 8 blank lines each. The next number is called the Load Memory Scan byte (LMS for short). This is the number we are about to change. The LMS says to the computer "Go get data starting at the location in memory that follows in the next two numbers...and by the way, start the screen with some of this kind of text (or graphics)".

Wow. I can't learn this!

After reading my explainations, you may think I'm as confused as you are, but it's not that hard, and my lessons not that bad! You must practice on your own beyond these examples. Try making modifications to what I have given you. Later I'll give you a chart that shows what number goes into the LMS byte (4th # in DL). In this case the chart would say that for a Graphics 0 LMS, we use 66. The next two numbers (5th and 6th bytes) are the low and high parts of the location where the screen data starts in memory. The example above was made up, so your numbers will be different depending on your memory size. That's why we store the value of the 5th #(DL+4 means the 5th number) in lines 20 & 30. We need to know where the data is being stored in YOUR computer.

The next numbers within the DL, the 2's tell the computer to put a line of Graphics 0 on the screen for each 2. Again the chart will give you these numbers later for other modes and types of scrolling. There are 23 2's plus the line of mode 0 that the LMS created gives 24 lines of mode 0. If you want, look ahead at the chart to see all of this. Under Graphics mode 0 it says to put 2 in the DL, ie. 2 is the instruction that MEANS put some GR.0 on the screen. Finally comes a 65 that means all done, go start all over again using the DL at the location in the next two numbers (usually the same DL).

Ok! So far we have found where the data to put on the screen is located (lines 20-30). Now, in line 40 we read the Joystick. If it is held left, then we take the value in DL+4, called here PDL4, and add 1. If it is held right, we subtract 1 in line 60. Now line 70 POKEs this changed value back into the DL at DL+4. This says to the computer "start displaying the data on the screen one byte more (or less) than where you did before." SIMPLE..??...Sure, why not!

Line 80 is a delay loop to slow the action down so we can see what is happening. Line 90 then goes back to read the Joystick to see if you want to move the display some more.

Notes:

1) You can speed up this method by changing or leaving out the delay loop.

2) So far we haven't changed the high part of the data address (DL+5). If you try to move continually in one direction the value of PDL4 will reach 0 or 255. Recall that the ATARI uses 8 bits per word in memory, ...and 8 bits in binary only counts to 255. When this happens, the number you POKE into DL+4 will no longer have any effect. Actually it is causing errors that the example is TRAPing to keep the program going. It's sort of like bottoming out. The simple solution is to do some mathamatics to change the high byte (DL+5) when needed. We do this in Example 2.

EXAMPLE 2

This time we scroll vertically:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 DL4=DL+4:DL5=DL+5
40 NUML=PEEK(DL4)
50 NUMH=PEEK(DL5)
60 ST=STICK(0)
70 IF ST=14 THEN NUML=NUML+40
80 IF ST=13 THEN NUML=NUML-40
90 IF NUML<0 THEN 140
100 IF NUML<256 THEN 160
110 NUML=NUML-256
120 NUMH=NUMH+1
130 GOTO 160
140 NUML=NUML+256
150 NUMH=NUMH-1
160 IF NUMH<0 THEN 60
170 IF NUMH>255 THEN 60
180 POKE DL4,NUML:POKE DL5,NUMH
190 GOTO 60
```

The only real difference in COARSE vertical scrolling over that of horizontal is that to move vertically we want to add or subtract 40 instead of 1. Do you know why? Of course! It's because we are using the ATARI's standard 40 characters per line for mode 0. Thus, to move the LMS address up one line, add 40. This is done in line 70. For other modes you may need to use 20 or 10 instead of 40. Look ahead at the chart under the row "Bytes of memory per line". Also this time we include the mathamatics to change both numbers of the address of the data to display. The math won't be explained.....just use it as is. It is standard computer math explained in most references. Finally, note that no delay loop was used. The extra math delays the computer enough!

EXAMPLE 2A

This is a new example added by suggestion from one of our customers. Some people didn't see how to combine horizontal and vertical scrolling, so here it is. You can later do the same thing for FINE scrolling. See Micheal, I told you I listen to my students!

```
1 GRAPHICS 17:? #6;"    EXAMPLE 2A"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
10 GRAPHICS 0
20 LISI
30 DL=PEEK(560)+256*PEEK(561)
40 DL4=DL+4:DL5=DL+5
50 NUML=PEEK(DL+4)
60 NUMH=PEEK(DL+5)
65 POKE 53279,8
70 ST=STICK(0)
80 IF ST=14 OR ST=10 OR ST=6 THEN NUML=NUML+40
90 IF ST=13 OR ST=9 OR ST=5 THEN NUML=NUML-40
92 IF ST=6 OR ST=7 OR ST=5 THEN NUML=NUML-1
94 IF ST=10 OR ST=11 OR ST=9 THEN NUML=NUML+1
95 CON=PEEK(53279):IF CON=6 THEN 1234
100 IF NUML<0 THEN 160
110 IF NUML<256 THEN 175
120 NUML=NUML-256
130 NUMH=NUMH+1
150 GOTO 175
160 NUML=NUML+256
170 NUMH=NUMH-1
175 IF NUMH<0 THEN 70
177 IF NUMH>255 THEN 70
180 POKE DL4,NUML:POKE DL5,NUMH
190 GOTO 70
1000 POKE 53279,8
1010 CON=PEEK(53279):IF CON=6 THEN 1234
1234 POKE 764,12:RUN "D:EX3"
```

EXAMPLE 3

Lets sketch out an abbreviated screen of data:

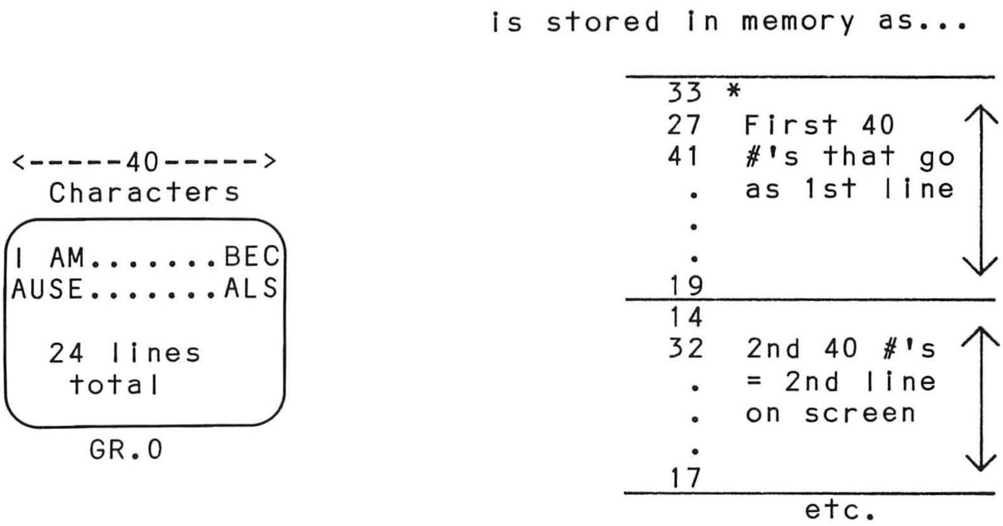


Figure 1

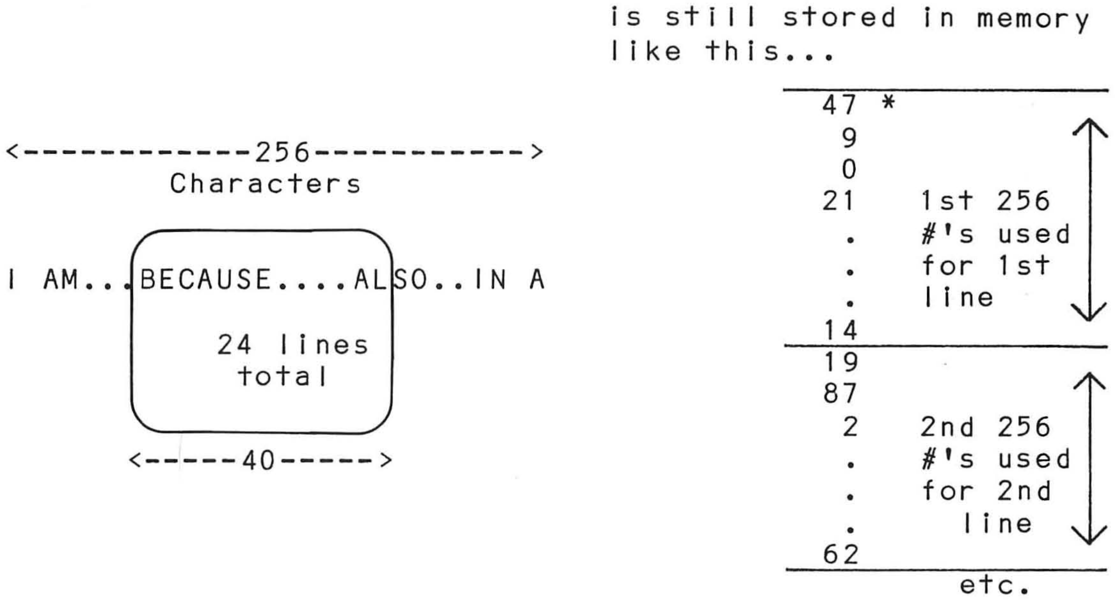


Figure 2

* This number is stored in your memory at (DL+4)+(DL+5)*256

Look at the top of Figure 1. The box drawn represents a screen area which is 40 characters wide by 24 down. You can see that if we attempt to scroll horizontally over this area, we have nowhere to go, so the data on the last line just moves up to the next line one character at a time. What can we do? Figure 2 shows some data that is now wider. How wide is optional with you. Since memory is always stored 1 value at a time, you can set up lines of any width. It is the use of more than one LMS byte that sets up different line widths. The box that represents the screen now has room to move to either side when horizontally scrolled.

The idea in the above paragraph is not easy to grasp. Let's start with some basics. Memory is organized like a very long strip of numbered locations. When we say to PEEK at (look at) a certain memory location, what we are really doing is to go down this strip of locations until we get to the number in question. Then we read what is written there. Got that? Good! When the computer starts to draw the screen... 60 times a second ....it goes out to the location we discussed earlier (DL+4 and DL+5). There it gets the location for the start of the screen data. In a 48k machine in GR. 0, this would be at 40000. If an "A" is currently being displayed in the upper left corner, a PEEK at this location will show "33".

To draw the rest of the screen, the computer looks at the next 39 locations (for a total of 40) and draws them across the screen. Now it says to itself: "Since I'm in GR.0, I must place the next value back at the start of the screen, but one row down. This means the 41st value goes right under the 1st. This goes on until the screen is filled (after 40*24 values have been placed on the screen). What we do to make memory seem like Figure 2 is to fool the computer into taking every 256th value as the start of a new line. This allows us to move sideways over the data. The sideways comment is from the perspective of the screen... memory is still just that long strip of locations, and we are still taking 40 memory values at a time for each row of the screen in GR.0. But after 40 values are used, we skip 256-40 values before the next line is drawn. The other 216 values are being saved for when we scroll. In Figure 2, imagine that the initial screen shows us values from 8 to 47. To scroll left, just have the screen show values from 7 to 46, 6 to 45, etc. When we get to showing 0 to 39 we must stop or the values will jump up a line as we would now be showing 255 of the last row through 38 of the current row. This is what we saw in Example 2.

```
1 GRAPHICS 17:? #6;"    EXAMPLE 3"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
5 P106=PEEK(106)
10 POKE 106,PEEK(106)-24
15 GRAPHICS 0:LIST
20 DL=PEEK(560)+256*PEEK(561)
30 P560=PEEK(560):P561=PEEK(561)
40 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
50 START=DL4+256*DL5
60 FOR I=1 TO 24
70 POKE DL+3*I,66
80 POKE DL+1+3*I,DL4
90 POKE DL+2+3*I,DL5+I
100 NEXT I
110 POKE DL+78,65
120 POKE DL+79,P560:POKE DL+80,P561
130 K=0
135 POKE 53279,8
140 ST=STICK(0)
145 CON=PEEK(53279):IF CON=6 THEN 1234
150 IF ST=11 THEN K=K+1
160 IF ST=7 THEN K=K-1
170 FOR L=0 TO 23
180 POKE DL+4+3*L,DL4+K
190 NEXT L
200 GOTO 140
1234 POKE 106,P106:RUN "D:EX4"
```

WE'LL TAKE IT SLOW: Location 106 holds the number of pages that your memory size allows. In my 48k machine it is 106. In line 10 we store the original value to restore in line 230 when the program is done. Now, in line 20, we fool the computer by subtracting 24 pages from 106 (a page is 256 bytes of memory). Then, in line 30 when we say Graphics 0, the computer looks at location 106, thinks memory is 24*256 bytes less than it is, and places the Display List down by this much in memory. What we have so easily accomplished is to save that many extra bytes of memory for a lot of data for the screen. Lines 40 to 60 store the location of the DL and the start of screen data as before.

The next lines, 70 to 110, do a nice trick. Now that we have such a large area to use, we POKE the DL with a whole bunch of LMS, low, high instructions, with each address pointing to data 256 bytes further down in memory. So what? Well, now there are lines on the screen (run Example 3) that are 256 bytes long. With the Joystick you can scroll and see NEW information appear....not from the row below, as before. Remember that there has to be enough of these LMS POKEs to fill the screen, which for GR.0 equals 24, thus the loop 1 to 24 on line 70.

The data we are scrolling over is again placed on the screen by the LIST command. Now you can see where the ending of each 40 bytes is, because that is where each new line of the program listing starts.

To better visualize this, look at Figure 2 again. The LMS addresses should point to the data for the left corner of each line. It sounds complicated, but by using 256 long lines we can just add 1 to the original DL5 value for each new LMS we set up. The new value will point down by one line of data...256 bytes!! Remember that values in the high byte of a number equal 256 times those in the low byte (just use the example if this is too hard...come back to it later) . Look at line 100 to see the above.

You could, with simplemathamatics, set up lines of any length you want, but lines of 256 characters were easier and clearer for a demonstration like this. Lines 160 to 220 increase or decrease the DL4 number by one, as in Example 1. But now the lines are 256 wide, so we can scroll without having characters jump up to the next line. Also note in line 200 that we only POKE every third line. This is because the DL must now use three lines of instructions for each line generated on the screen.

What if I'm lost

That is what is nice about the Tricky Tutorials--you don't have to understand all this. Just use the examples by copying them to your tape/disk and modify them with different PRINT and PLOT commands. The technical stuff is included to make a complete package for your use when ready.

EXAMPLE 4

Please run Example 4. Notice that we used Graphics 2 for this example. This was to show you that any mode will scroll. Also, we relocated the entire Display List to page six of memory (a reserved area for us to use) so that it could be modified while leaving the original intact.

Different Graphics modes!

What happens when you change Graphics modes? Well look at that famous chart we keep promissing and read the discussions later.

LOOK AT GRAPHICS MODE (0-8) THEN GO DOWN UNTIL DESIRED OPTION IS FOUND

Desired Option	GRAPHICS MODE								
	0	1	2	3	4	5	6	7	8
DL #	2	6	7	8	9	10	11	13	15
HScroll	18	22	23	24	25	26	27	29	31
VScroll	34	38	39	40	41	42	43	45	47
HS&VS	50	54	55	56	57	58	59	61	63
LMS	66	70	71	72	73	74	75	77	79
HS & LMS	82	86	87	88	89	90	91	93	95
VS & LMS	98	102	103	104	105	106	107	109	111
HS,VS,LMS	114	118	119	120	121	122	123	125	127
# Pixel lines per mode line	8	8	16	8	4	4	2	2	1
# Rows to fill screen	24	24	12	24	48	48	96	96	192
Bytes of memory per line	40	20	20	10	10	20	30	40	40
	65 = Start again at top of Display list 112= 8 Blank lines								

```
10 GRAPHICS 17:? #6;"     EXAMPLE 4"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 POKE 1536,112
50 POKE 1537,112
60 POKE 1538,112
70 FOR I=1 TO 12
80 POKE 1536+3*I,71
90 POKE 1536+3*I+1,0
100 POKE 1536+3*I+2,I
110 NEXT I
120 POKE 1575,65
130 POKE 1576,0
140 POKE 1577,6
150 POKE 560,0
160 POKE 561,6
170 K=130
180 POKE 53279,8
190 ST=STICK(0)
200 IF ST=7 THEN K=K-1
210 IF ST=11 THEN K=K+1
220 FOR J=1 TO 12
230 CON=PEEK(53279):iF CON=6 THEN 270
240 POKE 1536+3*J+1,K
250 NEXT J
260 GOTO 190
270 RUN "D:EX5"
```

EXAMPLE 5

This example generates some random characters and POKEs them right into the screen data area. To some of you, this will look like a useless example, but please think a moment. In order to get the text YOU want onto the screen, just replace the RANDOM statements in lines 120 to 170 with PRINT statements. If your text is not 256 characters wide, then fill in with blanks. The text could be stored in DATA statements and then read in as needed. Think what a nice business spread sheet could be done!

Did you notice that this example scrolls faster. The reason for this is that in ATARI BASIC, FOR-NEXT loops (which we use) go back to the first line and count forward on every loop. This means that they run faster if near the front. Also, if several lines are placed on one line speed increases. Finally, any math calculation should be done only once if possible. This example has these changes for you to study:

```
    50 X=DL+4:FOR L=0 TO 69 STEP 3:POKE X+L,Y:NEXT L:RETURN
```

```
270 ST=STICK(0):K=K+(ST=11)-(ST=7) :Y=DL4+K

310 GOSUB 50
```

LINE 50 does the exact same thing as lines 190 to 220 in Example 3. It is a subroutine to allow it to be moved to the top. Line 270 is much faster than the two IF statements. I better explain it. If ST = 11 then the ST=11 part will be TRUE, so (1) will be added to K. Likewise with ST=7. Since both can't be true at once, K will either go up or down by one each pass if the stick is moved. ATARI BASIC has it's nice points too!

```
10 GRAPHICS 17:? #6;"    EXAMPLE 5"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 GOTO 55
50 X=DL+4:FOR L=0 TO 69 STEP 3:POKE X+L,Y:NEXT L:RETURN
55 P106=PEEK(106)
60 POKE 106,PEEK(106)-24
70 GRAPHICS 0
80 DL=PEEK(560)+256*PEEK(561)
90 P560=PEEK(560):P561=PEEK(561)
100 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
110 DL5=DL5+1:POKE DL+5,DL5:POKE 89,DL5
120 START=DL4+256*DL5
130 FOR G=0 TO 4
140 FOR H=0 TO 255
150 POKE START+G*256+H,G+1*RND(9)*100
160 NEXT H
170 NEXT G
190 FOR I=1 TO 24
200 POKE DL+3*I,66
210 POKE DL+3*I+1,DL4
220 POKE DL+3*I+2,DL5+I-1
230 NEXT I
240 POKE DL+78,65
250 POKE DL+79,P560:POKE DL+80,P561
260 POKE 53279,8
270 ST=STICK(0):K=K+(ST=11)-(ST=7):Y=DL4+K
300 CON=PEEK(53279):IF CON=6 THEN 330
310 GOSUB 50
320 GOTO 270
330 POKE 106,P106:RUN "D:EX6"
```

EXAMPLE 6

What about a practical use? Well, how about a word processor that would allow you to input the width and then not wrap around, but rather scroll right or left as you want. To use Example 6, just type in a bunch of lines of text (almost filling the page is best). When ready, press RETURN. Now use the good old Joystick to scroll. If you look at the code, we added only a few lines. The POKE 559 at line 140 turns off the screen until we want it on at line 220. This speeds up the ATARI and looks more professional.

```
10 GRAPHICS 17:? #6;"    EXAMPLE 6"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
35 P106=PEEK(106)
40 POKE 106,PEEK(106)-24
50 GRAPHICS 0
60 DIM A$(255)
70 DL=PEEK(560)+256*PEEK(561)
80 P560=PEEK(560):P561=PEEK(561)
90 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
100 DL5=DL5+1:POKE DL+5,DL5:POKE 89,DL5
110 NON=PEEK(559)
120 START=DL4+256*DL5
130 INPUT A$
140 POKE 559,0
150 FOR I=1 TO 24
160 POKE DL+3*I,66
170 POKE DL+1+3*I,DL4
180 POKE DL+2+3*I,DL5+I-1
190 NEXT I
200 POKE DL+78,65
210 POKE DL+79,P560:POKE DL+80,P561
220 POKE 559,NON
230 K=0
240 POKE 53279,8
250 ST=STICK(0)
260 IF ST=11 THEN K=K+1
270 IF ST=7 THEN K=K-1
280 CON=PEEK(53279):IF CON=6 THEN 330
290 FOR L=0 TO 23
300 POKE DL+4+3*L,DL4+K
310 NEXT L
320 GOTO 250
330 POKE 106,P106:RUN "D:EX7"
```

EXAMPLE 7

This looks just like the last example, doesen't it? Well, when you press RETURN and try to scroll something funny will happen. The screen will go "crazy" because we POKE the wrong numbers into the DL. Look at the code and find the error. This error is typical of the type that occurs when modifing displays.

Now, just like the contests that a certain computer company holds, we will give $50. to the LAST person to send us the correct answer (Can you believe some people actually wrote us hoping they would be the last one and thus win a prize!). It's a joke, folks...don't write in.

```
10 GRAPHICS 17:? #6;"    EXAMPLE 7"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
35 P106=PEEK(106)
40 POKE 106,PEEK(106)-24
50 GRAPHICS 0
60 DIM A$(255)
70 DL=PEEK(560)+256*PEEK(561)
80 P560=PEEK(560):P561=PEEK(561)
90 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
100 DL5=DL5+1:POKE DL+5,DL5:POKE 89,DL5
110 NON=PEEK(559)
120 START=DL4+256*DL5
130 INPUT A$
140 POKE 559,0
150 FOR I=1 TO 24 STEP 3
160 POKE DL+3*I,66
170 POKE DL+1+3*I,DL4
180 POKE DL+2+3*I,(DL5+I-1)
190 NEXT I
200 POKE DL+78,65
210 POKE DL+79,P560:POKE DL+80,P561
220 POKE 559,NON
230 K=0
240 POKE 53279,8
250 ST=STICK(0)
260 IF ST=11 THEN K=K+1
270 IF ST=7 THEN K=K-1
280 CON=PEEK(53279):IF CON=6 THEN 350
290 FOR L=0 TO 23
300 POKE DL+4+3*L,DL4+K
310 NEXT L
320 GOTO 250
330 POKE 53279,8
340 CON=PEEK(53279):IF CON=6 THEN 350
350 POKE 106,P106:RUN "D:EX8"
```

EXAMPLE 8

This time we point out that you don't have to scroll all the lines on the screen. You can enable scrolling for only the lines you want. Look at lines 170 to 190 in this example. We POKE only the 3rd to 5th lines on the screen. You can scroll from one line to a screen full. Any mode line in the DL can be set from regular (DL #'s 2 to 15) to scrolling (see the chart; for example...GR.6) horizontally (27), vertically (43), or BOTH directions for diagonal scrolling (59). This effect can, for example, be used to make a nice scrolling "marque" of text across the screen.

The other possibilites for a mode line are LMS (LOAD MEMORY SCAN) mentioned above; HS & LMS, which is when you want tell the computer where to get its screen data and also have that line (remember LMS sets up a line on the screen also) be able to scroll. The same applies to vertical scrolling & LMS, and HS, VS, & LMS.

Yes, I am using a lot of terms and abbreviations. You may feel better knowing that even I, as a programmer, was confused at first. The concepts we are teaching here are NOT BASIC programming, but rather Graphics programming, which is a new and complex area. After this lesson, some of you non-programmers will be way ahead of many professionals unfamiliar with graphics.

```
1 GRAPHICS 17:? #6;"    EXAMPLE 8"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
5 P106=PEEK(106)
10 POKE 106,PEEK(106)-24
15 GRAPHICS 0:LIST
20 DL=PEEK(560)+256*PEEK(561)
30 P560=PEEK(560):P561=PEEK(561)
40 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
50 START=DL4+256*DL5
60 FOR I=1 TO 24
70 POKE DL+3*I,66
80 POKE DL+1+3*I,DL4
90 POKE DL+2+3*I,DL5+I
100 NEXT I
110 POKE DL+78,65
120 POKE DL+79,P560:POKE DL+80,P561
130 K=0
135 POKE 53279,8
140 ST=STICK(0)
150 IF ST=11 THEN K=K+1
160 IF ST=7 THEN K=K-1
```

```
165 CON=PEEK(53279):IF CON=6 THEN 1234
170 FOR L=2 TO 4
180 POKE DL+4+3*L,DL4+K
190 NEXT L
200 GOTO 140
1234 POKE 106,P106:RUN "D:EX9"
```

EXAMPLE 9

This example is coarse scrolling, but in the vertical direction with GRAPHICS. Just substitute your own PLOT commands if you like. The method is the same, only the numbers have been changed using the chart.

```
1 GRAPHICS 17:? #6;"    EXAMPLE 9"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
10 GRAPHICS 5+16
20 COLOR 1:PLOT 1,1:DRAWTO 79,39
30 DL=PEEK(560)+256*PEEK(561)
40 DL4=DL+4:DL5=DL+5
50 NUML=PEEK(DL+4)
60 NUMH=PEEK(DL+5)
65 POKE 53279,8
70 SI=STICK(0)
80 IF ST=14 THEN NUML=NUML+40
90 IF ST=13 THEN NUML=NUML-40
95 CON=PEEK(53279):IF CON=6 THEN 1234
100 IF NUML<0 THEN 150
110 IF NUML<256 THEN 170
120 NUML=NUML-256
130 NUMH=NUMH+1
130 NUMH=NUMH+1
140 GOTO 170
150 NUML=NUML+256
160 NUMH=NUMH-1
170 IF NUMH<0 THEN 70
180 IF NUMH>255 THEN 70
190 POKE DL4,NUML:POKE DL5,NUMH
200 GOTO 70
1234 RUN "D:EX10"
```

EXAMPLE 10

If you look on the chart, you'll see that for GR. 0 the regular number for the DL is 2, but to scroll (fine) vertically use 34. In the code for this example we POKE 34 into the DL in lines 25 to 40. This tells these lines to fine scroll, but how much? Fine scrolling means instead of moving a row (column) of letters or graphics one character or graphics pixel (these come in different sizes - see your BASIC manual) at a time, we move these characters one TV pixel at a time. TV pixels are the tiny dots you see when you look CLOSELY at your TV, or Monitor.

To fine scroll, we change the value in a new location: 54277. This location may contain 0 (normal) to 15. These numbers are the number of TV pixels that the line will be moved. This example is in a loop to scroll up 7 then reset to 0, then loop again. If we were in another Graphics mode, we would look at the chart to see how many pixels per mode line are being used. You then can scroll from 0 to 1 less than this number (for example, 0 to 7 is 8, the number of pixels in a mode 0 line). For GR.2, we scroll 0 to 15; for GR.8 you can't fine scroll (a coarse scroll here is the same as fine, one pixel line up).

The chart doesn't give values for every mode that the ATARI has. If you look at the row marked DL#, you will notice missing #'s 3,4,5,12,& 14. Until we do a special tutorial on these modes, you can use the Operating System manuals that ATARI sells to explore them. Also missing are Graphics modes 9,10 & 11 which were not in U.S. machines when this was written. These modes use #15 (GR. 8) with a few POKEs into the Operating System to change the way the data is interpreted.

```
1 GRAPHICS 17:? #6;"    EXAMPLE 10"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
10 GRAPHICS 0:LIST
20 DL=PEEK(560)+256*PEEK(561)
25 FOR S=9 TO 13
30 POKE DL+S,34
40 NEXT S
45 POKE 53279,8
50 FOR Y=0 TO 7
60 POKE 54277,Y:POSITION 2,20:? "THE NUMBER OF PIXELS
   SCROLLED IS ";Y:FOR W=1 TO 50:NEXT W
70 NEXT Y
100 CON=PEEK(53279):IF CON=6 THEN 1234
110 GOTO 50
1234 RUN "D:EX11"
```

EXAMPLE 11

Same thing as Example 10, but using the Joystick so you can practice moving up or down a pixel at a time. This will make sure everyone understands the diference between fine and coarse scrolling.

```
10 GRAPHICS 17:? #6;"    EXAMPLE 11"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 GRAPHICS 3
50 COLOR 1:PLOT 1,1:DRAWTO 1,10:DRAWTO 10,10:DRAWTO 10,1:
   DRAWTO 1,1
60 DL=PEEK(560)+256*PEEK(561)
70 FOR S=6 TO 15
80 POKE DL+S,40
90 NEXT S
100 POKE 53279,8
110 I=0
120 SI=STICK(0)
130 IF ST=14 THEN I=I+1
140 IF ST=13 THEN I=I-1
150 IF I<0 THEN I=0
160 IF I>7 THEN I=7
170 POKE 54277,I:? I
180 CON=PEEK(53279):IF CON=6 THEN 200
190 GOTO 120
200 RUN "D:EX12"
```

EXAMPLE 12

There is also a horizontal fine scroll register similar to the one for vertical fine scrolling. It is at 54276, right next to the other. To use it, look again at the chart. POKE the value into each line we want to scroll in the DL. For example, the correct value for a GR. 0 line that allows a horizontal scroll is 18. Now POKE the amount of clock cycles to scroll, 0 to 15 into 54276. No, I am not going to explain "clock cycles". It's exact meaning isn't important here and besides, like so many of the numbers we have been discussing, you will learn more if you try it yourself. It would be the exceptional person who could write a program to scroll without first trying it. Paper descriptions are not easy to read. Practice......

```
1 GRAPHICS 17:? #6;"    EXAMPLE 12"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
10 GRAPHICS 0:LIST
20 DL=PEEK(560)+256*PEEK(561)
30 POKE DL+9,18
40 POKE DL+11,18
50 POKE 53279,8
80 FOR X=0 TO 15
85 CON=PEEK(53279):IF CON=6 THEN 1234
90 POKE 54276,X:POSITION 2,20:? "   ":POSITION 2,20:? X
95 FOR W=1 TO 100:NEXT W
100 NEXT X
110 GOTO 40
1000 POKE 53279,8
1010 CON=PEEK(53279):IF CON=6 THEN 1234
1234 RUN "D:EX13"
```

EXAMPLE 13

We left a few goofs in this example. The LMS byte was not changed per the chart from 66 to 98. Also, a jump occurs on the screen. The method is just a combination of the two previous vertical scroll examples. After the great instruction you have been getting, you can figure this one out in a few minutes...CAN'T YOU ???

```
10 GRAPHICS 17:? #6;"    EXAMPLE 13"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 GRAPHICS 0:LIST
50 DL=PEEK(560)+256*PEEK(561)
60 DL4=DL+4:DL5=DL+5
70 NUML=PEEK(DL+4)
80 NUMH=PEEK(DL+5)
90 FOR S=6 TO 27
100 POKE DL+S,34
110 NEXT S
120 POKE 53279,8
130 I=0
130 I=0
140 POSITION 35,15:? I:ST=STICK(0)
150 IF ST=13 THEN I=I-1
160 IF ST=14 THEN I=I+1
170 IF I>7 THEN POKE 54277,0:I=0:GOTO 220
180 IF I<0 THEN POKE 54277,7:I=7:GOTO 220
190 POKE 54277,I:POSITION 2,20:FOR W=1 TO 10:NEXT W
195 CON=PEEK(53279):IF CON=6 THEN 350
```

```
200 GOTO 140
220 IF ST=14 THEN NUML=NUML+40
230 IF ST=13 THEN NUML=NUML-40
240 IF NUML<0 THEN 290
250 IF NUML<256 THEN 310
260 NUML=NUML-256
270 NUMH=NUMH+1
280 GOTO 310
290 NUML=NUML+256
300 NUMH=NUMH-1
310 IF NUMH<0 THEN 140
320 IF NUMH>255 THEN 140
330 POKE DL4,NUML:POKE DL5,NUMH
340 GOTO 140
350 RUN "D:EX14"
```

EXAMPLE 14

This is the best you can do to combine fine and coarse vertical scrolling. There are no new tricks to learn, just look at the sequence of how we put two previous examples together. Especially notice lines 190 and 300. This is pretty nice for BASIC! This example could be renumbered into a subroutine for you to use in programs or....... Since there are so many possible combinations of scrolling, we don't show you each combination. It is easy to combine whichever types of scrolling you need. Really!

```
10 GRAPHICS 17:? #6;"    EXAMPLE 14"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 GRAPHICS 0:LIST
50 DL=PEEK(560)+256*PEEK(561)
60 DL4=DL+4:DL5=DL+5
70 NUML=PEEK(DL+4)
80 NON=PEEK(559)
90 NUMH=PEEK(DL+5)
100 POKE DL+3,98
110 FOR S=6 TO 27
120 POKE DL+S,34
130 NEXT S
140 POKE 53279,8
150 I=0:POKE 752,1
160 REM **** MAIN LOOP ************
170 POSITION 35,14:? I:ST=STICK(0)
180 CON=PEEK(53279):IF CON=6 THEN 340
190 I=I-(ST=13)+(ST=14)
200 IF I>7 THEN I=0:GOTO 230
210 IF I<0 THEN I=7:GOTO 230
```

```
220 POKE 54277,I:GOTO 170
230 NUML=NUML+(ST=14)*40-(ST=13)*40
240 IF NUML<0 THEN 270
250 IF NUML<256 THEN 280
260 NUML=NUML-256:NUMH=NUMH+1:GOTO 280
270 NUML=NUML+256:NUMH=NUMH-1
280 IF NUMH<0 THEN 170
290 IF NUMH>255 THEN 170
300 POKE 559,0:POKE DL4,NUML:POKE DL5,NUMH:POKE 54277,I:POKE
    559,34
310 GOTO 170
320 POKE 53279,8
330 CON=PEEK(53279):IF CON=6 THEN 340
340 RUN "D:EX15"
```

EXAMPLE 15

When vertical scrolling, as in the example above (BASIC Language only), the screen seems to blink as it scrolls. This is because we are moving a line up 1,2,...7 pixels. Then we move it back to 0 and, as quickly as possible, coarse scroll it up 8 pixels. This gets the text to move up or down, but the "jump" shows on the screen. To hide the jump, the screen is POKEd off (559,0) before the jump and POKEd on (559,34) after. Thus, the blink.

NOW WE GIVE YOU...ASSEMBLY LANGUAGE.

Geoff Caras, of our little group, wrote a small Assembly subroutine to do the POKEs and you can see how quick and smooth it works. I won't feel bad if you use this example instead of my # 14 in your programs. Just change the Graphics to use this example as a subroutine in your programs!

```
10 GRAPHICS 17:? #6;"    EXAMPLE 15"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 DIM A$(38):TRAP 60:I=1
50 READ X:A$(I)=CHR$(X):I=I+1:GOTO 50
60 GRAPHICS 0:LIST :POKE 752,1
70 POKE 712,148
80 DL=PEEK(560)+256*PEEK(561)
90 DL4=DL+4:DL5=DL+5
100 NUML=PEEK(DL+4)
110 NON=PEEK(559)
120 NUMH=PEEK(DL+5)
130 POKE DL+3,98
```

```
140 FOR S=6 TO 27
150 POKE DL+S,34
160 NEXT S
170 POKE 53279,8
180 I=0
190 POSITION 35,20:? I:ST=STICK(0)
200 CON=PEEK(53279):IF CON=6 THEN 410
210 I=I-(ST=13)+(ST=14)
220 IF I>7 THEN I=0:GOTO 250
230 IF I<0 THEN I=7:GOTO 250
240 POKE 54277,I:GOTO 190
250 NUML=NUML+((ST=14)*40)-((ST=13)*40)
260 IF NUML<0 THEN 300
270 IF NUML<256 THEN 310
280 NUML=NUML-256:NUMH=NUMH+1
290 GOTO 310
300 NUML=NUML+256:NUMH=NUMH-1
310 IF NUMH<0 THEN 190
320 IF NUMH>255 THEN 190
330 X=USR(ADR(A$),I,DL4,NUML,DL5,NUMH)
340 GOTO 190
350 DATA 160,0,140,47,2,104,104,104
360 DATA 141,5,212,104,133,225,104,133
370 DATA 224,104,104,145,224,104,133,225
380 DATA 104,133,224,104,104,145,224,169
390 DATA 34,141,47,2,96,0
400 CON=PEEK(53279):IF CON=6 THEN 410
410 RUN "D:EX16"
```

EXAMPLE 16

This is an example of changing Graphics modes by the use of the chart. The titles could be brought in from DATA statements or disk if you wanted. These examples are now POKEing the border color to match the page (you really should have one of our Master Memory Maps to learn these POKEs). The examples are really starting to be professional looking.

```
10 GRAPHICS 17:? #6;"    EXAMPLE 16"
20 FOR W=1 TO 230:POKE 710,W
30 FOR ZR=1 TO 10:NEXT ZR:NEXT W
40 GRAPHICS 17:POSITION 2,10:? #6;"TITLE GOES HERE"
45 POKE 712,152:POKE 708,1
50 DL=PEEK(560)+256*PEEK(561)
60 DL4=DL+4:DL5=DL+5
70 NUML=PEEK(DL+4)
80 NON=PEEK(559)
90 NUMH=PEEK(DL+5)
```

```
100 POKE DL+3,102
110 FOR S=6 TO 27
120 POKE DL+S,38
130 NEXT S
140 POKE 53279,8
150 I=0:POKE 752,1
160 REM **** MAIN LOOP ************
170 POSITION 35,14:ST=STICK(0)
```

EXAMPLE 17

Another example of scrolling a Graphics line up and down, but this time with the Assembly subroutine. Think of the games you can write, or great educational programs! Don't fall into the trap of thinking that this is just a simple example, and a "real" example will be harder. All you have to change is the GRAPHICS statements!

```
1 GRAPHICS 17:? #6;"    EXAMPLE 17"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
10 DIM A$(38):TRAP 30:I=1
20 READ X:A$(I)=CHR$(X):I=I+1:GOTO 20
30 GRAPHICS 3:COLOR 1:PLOT 1,1:DRAWTO 19,19
40 DL=PEEK(560)+256*PEEK(561)
50 DL4=DL+4:DL5=DL+5
60 NUML=PEEK(DL+4)
70 NON=PEEK(559)
80 NUMH=PEEK(DL+5)
90 POKE DL+3,104
100 FOR S=6 TO 30
110 POKE DL+S,40
120 NEXT S
130 I=0
135 POKE 53279,8
140 ST=STICK(0)
145 CON=PEEK(53279):IF CON=6 THEN 1234
150 I=I-(ST=13)+(ST=14)
160 IF I>7 THEN I=0:GOTO 190
170 IF I<0 THEN I=7:GOTO 190
180 POKE 54277,I:GOTO 140
190 NUML=NUML+((ST=14)*10)-((ST=13)*10)
200 IF NUML<0 THEN 240
210 IF NUML<256 THEN 250
220 NUML=NUML-256:NUMH=NUMH+1
230 GOTO 250
240 NUML=NUML+256:NUMH=NUMH-1
250 IF NUMH<0 THEN 140
```

```
260 IF NUMH>255 THEN 140
270 X=USR(ADR(A$),I,DL4,NUML,DL5,NUMH)
280 GOTO 140
400 DATA 160,0,140,47,2,104,104,104
410 DATA 141,5,212,104,133,225,104,133
420 DATA 224,104,104,145,224,104,133,225
430 DATA 104,133,224,104,104,145,224,169
440 DATA 34,141,47,2,96,0
1234 RUN "D:EX18"
```

EXAMPLE 18

No, we aren't going to offer an Assembly program for horizontal scrolling. This is a tutorial to learn the principals of scrolling.

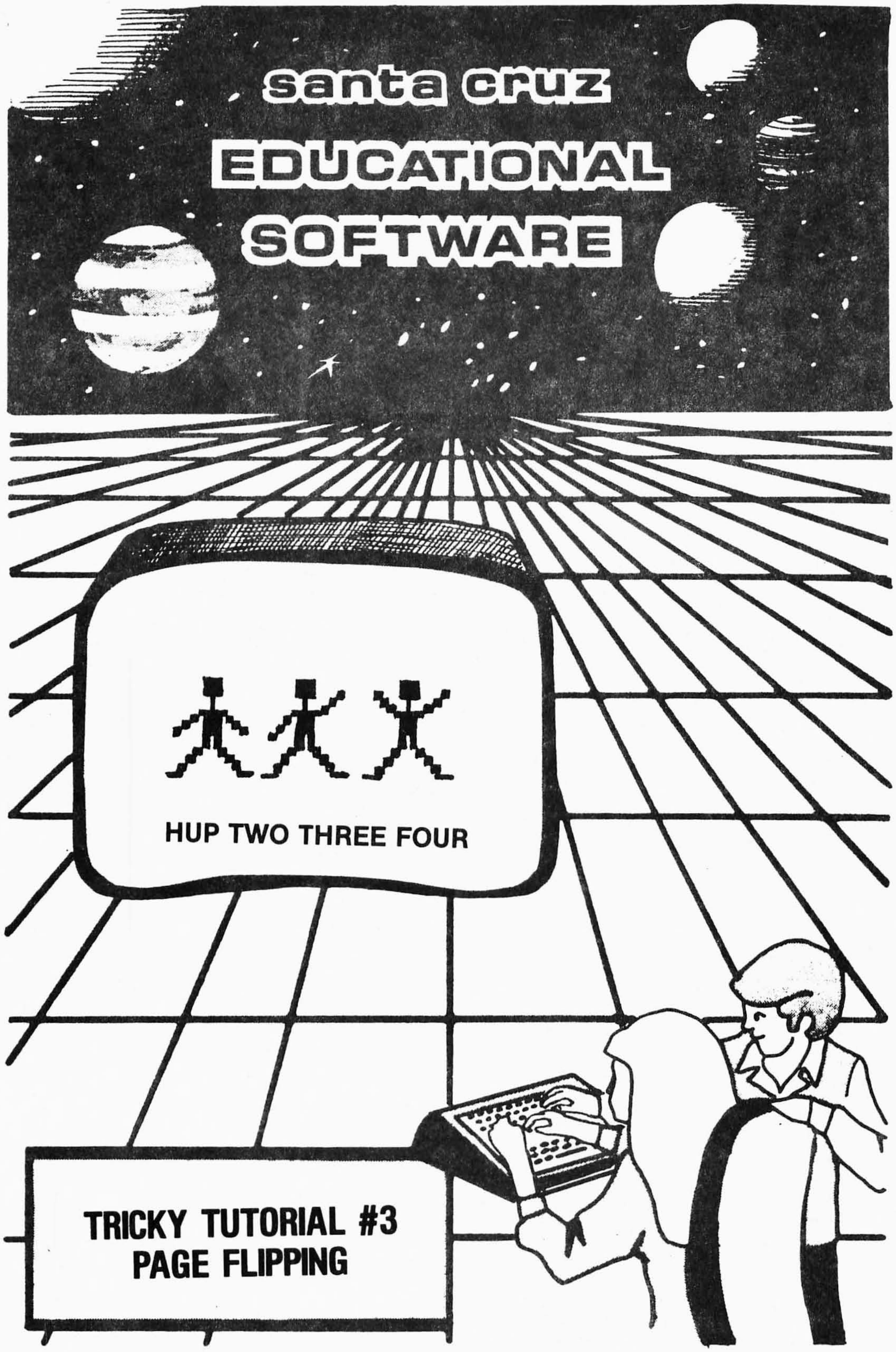
This example, our last, is again combining two previous examples to allow continuous scrolling horizontally. The POKE to turn off the screen is still needed, so that is the blink you see. However, the junk on the screen is because of a point we didn't mention yet. Whenever you POKE a Display List, you should wait for the time when the TV is blank and waiting to do the next screen. Well, this happens 30 times a second! BASIC can't go that fast, but if you program in Assembly use WSYNC to prevent the flashing junk that this BASIC example shows. For the rest of us, we'll wait for some more subroutines like example 15.

There are many details left out of this discussion. For example try Poking 559,33 and 559,35. This gives you wide or small playfields of 48 or 32 characters width automatically, without the trouble we went through earlier for the 256 width.

```
1 GRAPHICS 17:? #6;"    EXAMPLE 18"
2 FOR W=1 TO 230:POKE 710,W
3 FOR ZR=1 TO 10:NEXT ZR:NEXT W
4 ? #6;"   LAST EXAMPLE":FOR W=1 TO 200:POKE 710,W:NEXT W
10 TRAP 40000:TRAP 300
15 P106=PEEK(106)
20 NON=PEEK(559):GRAPHICS 0:GOTO 50
40 M=M+(ST=11)*5-(ST=7)*5:Y=DL4+M:POKE 559,0:FOR L=0 TO 72
   STEP 3:POKE X+L,Y:NEXT L:POKE 54276,K:POKE 559,NON
45 GOTO 130
50 POKE 106,PEEK(106)-24:POKE 766,1
60 GRAPHICS 0:LIST
70 DL=PEEK(560)+256*PEEK(561)
80 P560=PEEK(560):P561=PEEK(561)
```

```
90 DL4=PEEK(DL+4):DL5=PEEK(DL+5)
100 FOR I=1 TO 25:I3=3*I:POKE DL+I3,82:POKE DL+1+I3,DL4:POKE
    DL+2+I3,DL5+I:NEXT I
110 POKE DL+78,65:POKE DL+79,P560:POKE DL+80,P561
120 K=0:X=DL+4:M=0
125 POKE 53279,8
130 POKE 54276,K:ST=STICK(0):K=K-(ST=11)+(ST=7)
135 CON=PEEK(53279):IF CON=6 THEN 1234
140 IF K>15 THEN K=0:GOTO 40
150 IF K<0 THEN K=15:GOTO 40
160 GOTO 130
300 GRAPHICS 0:? "†":? "YOU WENT TO FAR...PRESS RESET AND
    TYPE RUN....PRESS RETURN."
310 END
1234 POKE 106,P106:GRAPHICS 0:POKE 709,0
1235 ? " THANK FOR BUYING OUR TRICKY TUTORIALS....";:GOTO
     1235
```

I appreciate that all of you students have stayed awake so long. I'll see you in my next Tricky Tutorial. BYE!



TRICKY TUTORIAL #3
PAGE FLIPPING

PAGE FLIPPING

Page Flipping is a set of simple programs designed to teach those new to the ATARI how to store information in memory and then bring it back on the screen instantly! With the two methods taught here, even a new programmer can learn to do simple animation, or present nice slide-like displays. The simplest of these examples has only 12 short lines of basic code!

The person using this lesson should be familiar with BASIC programming so that he or she can read the code that is included. Since the program is not protected, the user is encouraged to try their own modifications inorder to gain a greater perspective of the art of flipping pages of memory.

The program is split up into smaller pieces that will load and run on ATARI(tm) 400/800 computers with 16k RAM for cassette users and 24k RAM for those using disk.

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

Educational Software

presents

TRICKY TUTORIAL #3

PAGE FLIPPING

TRICKY TUTORIAL #3

PAGE FLIPPING

by
Robin Sherer

No doubt some of you hesitated before ordering a program to "flip pages". For this reason I want to start by assuring you that once you learn either of the two simple methods taught here, your programs will look and run much nicer.

Not very long ago, I had to learn the method of flipping screens. Since I still remember how confusing it was at first, only the material you need is presented here. A few details have been left out because only those trying to become experts will care, and they will have to read the technical manuals anyway. The style of the manual is informal mostly because I have never heard a good explaination for instructions that are written in a cold manner. We bought our computers to learn and have fun!.

HOW TO LOAD

TAPE....

Place the tape in your recorder, label side up, and insert your BASIC Cartridge. Make sure the tape is rewound, and reset the counter to zero. Push PLAY on the recorder, type RUN"C:", and press RETURN. If the program won't start to load, try positioning the tape forwards or backwards a little. The easiest way to find the beginning is to listen to the "noise" on the tape with a regular recorder. When you find the steady tone, you have the beginning of the program. We recommend you write down the number on your recorder's counter as each program starts, this will make it easier to find each part later on.

DISK....

To load and run the disk, first turn on your disk drive. When the busy light goes out, place the disk into the drive, with the BASIC Cartridge in place. The program will load each part and run by itself.

Any defective tapes or disks should be returned to:
Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

What is Page Flipping

The idea of flipping pages is somewhat unique in small home computers. You're lucky, the ATARI just happens to be able to do page flipping. The reason for this is that the ATARI allows both its DISPLAY LIST and DISPLAY DATA to be stored in any free area of memory. A Display List is the small set of instructions every computer needs to tell itself how to put the display data on the screen. This is especially important for us to learn since the ATARI computers offer so many types of graphics modes. This special capability means that you can store the information for numerous pages (TV screens) of graphics and/or text and then later, in your program, go to any of these pages instantly. This makes things look much more professional.

—BASICS—

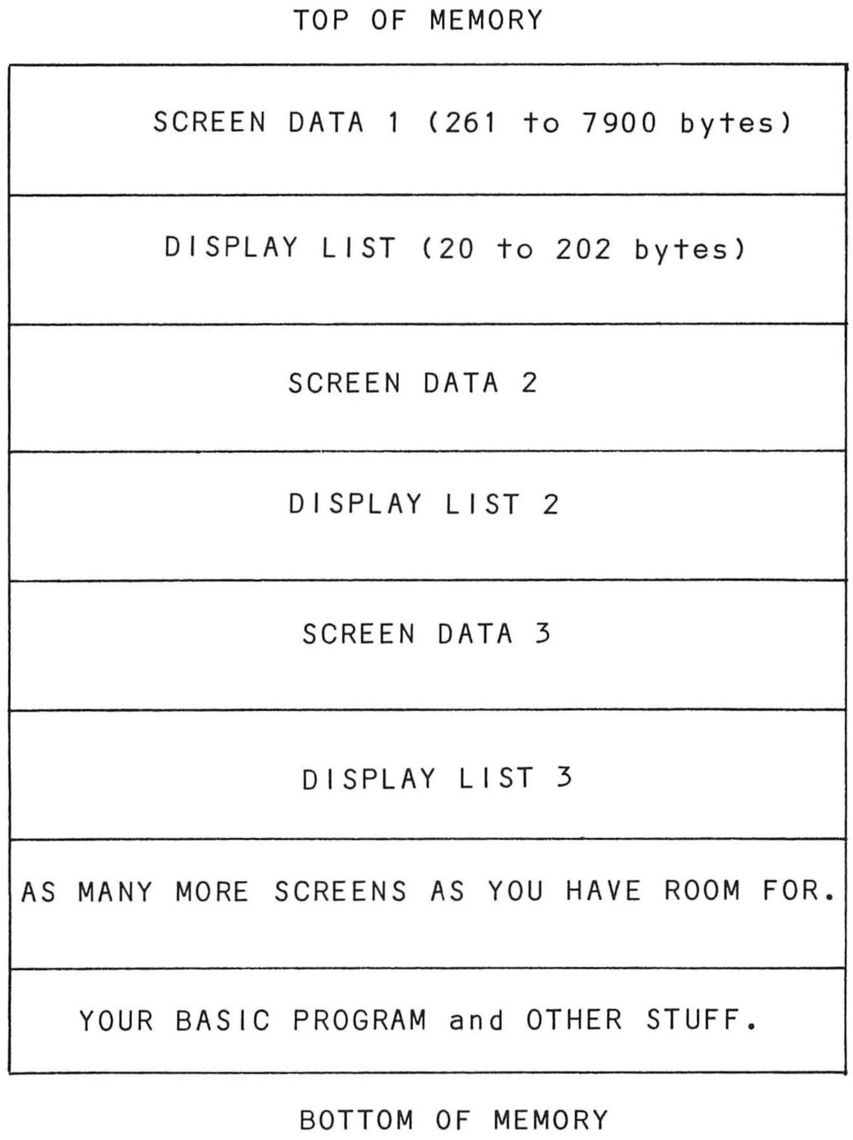
First, we are going to cover some basics. Memory is stored in something called RAM, which stands for random access memory. This is because you can either read or write to any place within it. Usually discussion of memory is in terms of bytes. For example, your memory(RAM) may consist of 16k,24k,32k,40k, or 48k bytes. Remember that 1k is actually 1024 bytes. Confused? GOOD! Then you may appreciate the fact that to flip screens of data we are going to just move down in memory by increments of 256 bytes at a time. These increments are called "pages" (ain't life simple???). So 4 pages is 4*256 = 1024 bytes = 1k . The last sentence is all you have to remember. Four pages of memory =1k bytes of memory. Oh yes, I should tell you that "k" is the symbol for 1000.

The term flipping pages would now become confusing - we are NOT flipping 256 bytes at a time. For this reason we will refer to the process as flipping screens or displays,ie., the stuff you see on the TV screen at any one time.

There will be two methods shown in our examples. DON'T JUST VIEW THEM. COPY THEM TO ANOTHER DISK OR TAPE AND MODIFY THEM FOR YOUR OWN USE. WE WANT YOU TO COPY OUR SOFTWARE FOR USE IN YOUR OWN PROGRAMS. EACH EXAMPLE IS SET UP FOR GENERAL PURPOSES, SO JUST SUBSTITUTE YOUR OWN TEXT OR GRAPHICS. IF YOU HAVE 32K OR MORE OF MEMORY, CHANGE TO HIGH RES, GRAPHICS MODE 8 AND ADD TEXT AS WELL AS GRAPHICS. The method and examples all work, but could be much better....especially if written for more memory than the small amount this lesson runs on (don't brag about your 48k machine. My ATARI has 160k!)

— METHODS 1 —

Normally when you say Graphics 0 to 8 (11 for newer units) to the computer, it sets up both a Display List and a data area just below the top of your memory. Method one simply tells the computer after a first screen is drawn, "memory ain't where it used to be, but it's now lower, so set up a new (additional) Display List and data area lower in memory". By doing this as often as you require (and memory space allows), you allow a number of screens full of data to be seen by just redirecting the one location that points to the start of each Display List. EASY! Figure one will show you this graphically:



TOP OF MEMORY

SCREEN DATA 1 (261 to 7900 bytes)

DISPLAY LIST (20 to 202 bytes)

SCREEN DATA 2

DISPLAY LIST 2

SCREEN DATA 3

DISPLAY LIST 3

AS MANY MORE SCREENS AS YOU HAVE ROOM FOR.

YOUR BASIC PROGRAM and OTHER STUFF.

BOTTOM OF MEMORY

Fig. 1

EXAMPLE 1

Run Example 1. Notice that you can see it draw out some random lines, then they disappear and a new set of lines is drawn out. Then...these two screens seem to appear and disappear, first one then another.
Here's how it works:

```
10 GRAPHICS 6
12 GOSUB 4000
15 TRAP 20
20 COLOR 1:FOR I=1 TO 20:COLOR 2*RND(4):DRAWTO 140*RND(4),70
   *RND(9):NEXT I
25 ? "THIS IS FLIPPING BETWEEN TWO AREAS OF MEMORY.  PRESS
   RETURN TO CONTINUE."
30 A=PEEK(106)
40 DLISTL1=PEEK(560):DLISTH1=PEEK(561)
50 POKE 106,A-32
60 GRAPHICS 6
61 GOSUB 4000
70 DLISTL2=PEEK(560):DLISTH2=PEEK(561)
75 TRAP 80
80 COLOR 1:FOR I=1 TO 20:COLOR 2*RND(4):DRAWTO 140*RND(4),70
   *RND(9):NEXT I
85 ? "THIS IS FLIPPING BETWEEN TWO AREAS OF MEMORY.  PRESS
   RETURN TO CONTINUE."
86 POKE 764,255
90 POKE 561,DLISTH1:FOR W=1 TO 2:NEXT W:POKE 561,DLISTH2:IF
   PEEK(764)=12 THEN 110
100 GOTO 90
110 POKE 106,A:RUN "D:NEXT1"
4000 X=PEEK(16):IF X=128 THEN 4020
4010 POKE 16,X-128:POKE 53774,X-128
4020 RETURN
```

Lines 10 to 25 draw a screen just like normal using COLOR, PLOT, and PRINT commands (See your BASIC referance manual for these commands).

Line 30 stores the original top of memory (in number of 256 byte pages) read from location 106 using the PEEK command.

Line 40 stores the two numbers that locate the Display List. Remember that it takes two eight bit numbers to address all of the computers memory. The LOW part can hold from 0 to 255 and the HIGH part holds from 0 to 255 (times 256)...again see your manual or our MASTER MEMORY MAP. We will only use the part of the address in location 561. This is because we are moving memory down in whole page increments and 561 holds the number of whole pages. We left in the low part, in case you want to experiment with your own ideas.

— NOTE —

Often I leave out pieces of code that you may not see mentioned in the manual. You should assume that it either is something you will see the need for later, or something needed just to make all the examples run together smoothly.

Line 50 says "the top of memory is located down in RAM by 32 pages (8k bytes) from the previous value. This is more than is needed for this example. The extra amount will allow you room to add more screens later when you come back to this example to experiment with your own changes.... and you will come back, won't you?

Line 60 to 85 again draw some graphics just like normal. The computer looks at location 106 and sees the value we put there. It is thus "fooled" into placing the new Display List and data starting 32 pages lower than it otherwise would have. Actually,the amount of pages you go down should correspond to the amount of memory the graphics modes you are using requires. Look in the first page of the BASIC Manual's Graphics section to see how much you need. We used mode 6, so 2k or 8 pages would have been enough. This way, though, we left room for you to easily copy this program and substitute your own graphics!

Line 70 stores the location or address of this second Display List. You can do this before (like here, line 70) or after the graphics statements. The location will be used to flip the screens later.

Line 90 then does the flipping by just POKEing 561 with the address (high part only!) of the first Display List, then the second, then the first, etc. This says "use this Display List..no this one...no use the first..etc.".

ANOTHER NOTE...

You can see in the program code in Example 1 it is line 110 that POKEs 106 back to the original value. If you are going to go on and use your computer after confusing it like this, you had better tell it what the real top of memory is. If, on the other hand, you are going to turn it off after our program is finished, don't bother. Any time you power up or press RESET, the computer will store the correct value in 106.

Also, while running our program, don't press RESET or BREAK unless it crashes (it shouldn't, but...). If any example stops then press RESET and type RUN and press RETURN. Of course, you should press RESET and then experiment with each example after you have seen the entire program at least once. This is how you will learn!

EXAMPLE 2

This example is made out of Example 1 to show you how to expand the basic idea of changing screens. Here we set up FOUR Display Lists and print four simple messages on the screen. As you press a button the message appears instantly! If you think about it, you will realize that there is no diference in flipping between four one line messages on a page, or four completely filled pages of text. Each time you press a button, another area of memory is displayed on the screen. The one line messages were just to keep the program simple for you to study. Please add many more lines of text to this example and see what we mean. You just put more words between the quotes in lines 20, 70, 110, & 150.

Lines 10 to 40 write text and store the needed values for Display List one (DL1).

Lines 50 to 80 POKEs 106 down 8 pages (2k, more than enough for GR.0), writes more text, and stores these new values.

Lines 90 to 120 same thing another 8 pages down in memory, but different text.

Lines 130 to 160 same thing again for a fourth screen!

Lines 168 to 185 look for you to press keys 1 to 4 (or whatever!) so that lines 190 to 220 can tell the computer which of the four Display List you previously created to now use. The test for CH=12 is to see if you decided to go on to something else by pressing return. The POKE of 255 to 764 clears the location that holds the internal code for the last key pressed. The variable CH on line 170 will tell the system which key you just pressed (if any).

NOTE

You can put in your own text or have the computer read in the text from the keyboard or disk and place the first 960 bytes (the size of GR.0) in page one, the next 960 in page 2, etc. The way to read in data from keyboard or disk is explained in your reference manuals.

```
5 TRAP 10
10 GRAPHICS 0
20 POSITION 4,10:? "THIS IS PAGE 1.":POSITION 2,20:? "PRESS
   1,2,3 OR 4 FOR THAT PAGE."
25 ? "PRESS RETURN TO CONTINUE."
30 A=PEEK(106)
```

```
40 DLL1=PEEK(560):DLH1=PEEK(561)
45 REM ******************************
50 POKE 106,A-8
60 GRAPHICS 0
70 POSITION 10,10:? "THIS IS PAGE 2.":POSITION 2,20:? "PRESS
   1,2,3 OR 4 FOR THAT PAGE."
75 ? "PRESS RETURN TO CONTINUE."
80 DLL2=PEEK(560):DLH2=PEEK(561)
85 REM ******************************
90 POKE 106,A-16
100 GRAPHICS 0
110 POSITION 15,10:? "THIS IS PAGE 3.":POSITION 2,20:? "PRESS
    1,2,3 OR 4 FOR THAT PAGE."
115 ? "PRESS RETURN TO CONTINUE."
120 DLL3=PEEK(560):DLH3=PEEK(561)
125 REM *****************************
130 POKE 106,A-24
140 GRAPHICS 0
150 POSITION 20,10:? "THIS IS PAGE 4.":POSITION 2,20:? "PRESS
    1,2,3 OR 4 FOR THAT PAGE."
155 ? "PRESS RETURN TO CONTINUE."
160 DLL4=PEEK(560):DLH4=PEEK(561)
165 REM *****************************
168 POKE 764,255:GOSUB 4000
170 CH=PEEK(764)
180 IF CH=31 THEN 190
181 IF CH=30 THEN 200
182 IF CH=26 THEN 210
183 IF CH=24 THEN 220
184 IF CH=12 THEN 230
185 GOTO 170
190 POKE 560,DLL1:POKE 561,DLH1:GOTO 170
200 POKE 560,DLL2:POKE 561,DLH2:GOTO 170
210 POKE 560,DLL3:POKE 561,DLH3:GOTO 170
220 POKE 560,DLL4:POKE 561,DLH4:GOTO 170
230 POKE 106,A:RUN "D:NEXT2"
4000 X=PEEK(16):IF X=128 THEN 4020
4010 POKE 16,X-128:POKE 53774,X-128
4020 RETURN
```

EXAMPLE 3

Example 3 is the same as Example 2, but with graphics instead of text. Although it will seem obvious to some of you, all you need to do is substitute graphics type commands for the text commands of Example 2. We used four simple bar graphs, but you could draw very complicated pictures if you wanted. Remember that the material that you place on a screen doesn't effect the basic method we are using. Even if you fill up the screen, the computer just keeps looking at the Display List to see where to get it's screen data. No more watching complicated pictures drawn out every time you need them. Now you can store them ahead of time in memory.

```
5 TRAP 10
10 GRAPHICS 5
15 COLOR 1:PLOT 5,5:DRAWTO 5,35:DRAWTO 75,35
20 COLOR 2:PLOT 10,34:DRAWTO 10,25:DRAWTO 5,25:POSITION 5,34
   :POKE 765,2:XIO 18,#6,0,0,"S:"
25 ? "PRESS 1,2,3, OR 4 FOR BAR GRAPHS OF":? "THE YEARS 1981,
   1982,1983 OR 1984.":? "PRESS RETURN TO GO ON."
30 A=PEEK(106)
40 DLL1=PEEK(560):DLH1=PEEK(561)
45 REM *******************************
50 POKE 106,A-8
60 GRAPHICS 5
65 COLOR 1:PLOT 5,5:DRAWTO 5,35:DRAWTO 75,35
70 COLOR 3:PLOT 15,34:DRAWTO 15,20:DRAWTO 10,20:POSITION 10,
   34:POKE 765,3:XIO 18,#6,0,0,"S:"
75 ? "PRESS 1,2,3, OR 4 FOR BAR GRAPHS OF":? "THE YEARS 1981,
   1982,1983 OR 1984.":? "PRESS RETURN TO GO ON."
80 DLL2=PEEK(560):DLH2=PEEK(561)
85 REM *******************************
90 POKE 106,A-16
100 GRAPHICS 5
105 COLOR 1:PLOT 5,5:DRAWTO 5,35:DRAWTO 75,35
110 COLOR 1:PLOT 20,34:DRAWTO 20,15:DRAWTO 15,15:POSITION 15,
    34:POKE 765,1:XIO 18,#6,0,0,"S:"
115 ? "PRESS 1,2,3, OR 4 FOR BAR GRAPHS OF":? "THE YEARS 1981,
    1982,1983 OR 1984.":? "PRESS RETURN TO GO ON."
120 DLL3=PEEK(560):DLH3=PEEK(561)
125 REM *******************************
130 POKE 106,A-24
140 GRAPHICS 5
145 COLOR 1:PLOT 5,5:DRAWTO 5,35:DRAWTO 75,35
150 COLOR 2:PLOT 25,34:DRAWTO 25,10:DRAWTO 20,10:POSITION 20,
    34:POKE 765,2:XIO 18,#6,0,0,"S:"
155 ? "PRESS 1,2,3, OR 4 FOR BAR GRAPHS OF":? "THE YEARS 1981,
    1982,1983 OR 1984.":? "PRESS RETURN TO GO ON."
160 DLL4=PEEK(560):DLH4=PEEK(561)
165 REM *******************************
168 POKE 764,255:GOSUB 4000
170 CH=PEEK(764)
180 IF CH=31 THEN 190
181 IF CH=30 THEN 200
182 IF CH=26 THEN 210
183 IF CH=24 THEN 220
184 IF CH=12 THEN 230
185 GOTO 170
```

```
190 POKE 560,DLL1:POKE 561,DLH1:GOTO 170
200 POKE 560,DLL2:POKE 561,DLH2:GOTO 170
210 POKE 560,DLL3:POKE 561,DLH3:GOTO 170
220 POKE 560,DLL4:POKE 561,DLH4:GOTO 170
230 POKE 106,A:RUN "D:NEXT3"
4000 X=PEEK(16):IF X=128 THEN 4020
4010 POKE 16,X-128:POKE 53774,X-128
4020 RETURN
```

EXAMPLE 4

This example draws a shape on each page. Then, by flipping screens you can animate the shape! Remember that any set of data can be used for the shape so why not copy this program to your disk/cassette and try your own shapes. Another idea would be to draw a business logo and move it across a chart of profits. Use your imagination, or just play if you like. The posibilities are endless!

```
3 GRAPHICS 0
5 TRAP 10
10 GRAPHICS 5
15 COLOR 1
20 READ X,Y:IF X=0 THEN 30
25 PLOT X,Y:GOTO 20
27 DATA 9,23,10,23,11,23,9,24,10,24,11,24,10,25,8,26,9,26,
   10,26,11,26,12,26,7,27,9,27,10,27,11,27,13,27
28 DATA 6,28,9,28,10,28,11,28,14,28,9,29,10,29,11,29,9,30,
   10,30,11,30,9,31,11,31,8,32,12,32,7,33,13,33,7,34,13,34
29 DATA 7,35,8,35,9,35,13,35,14,35,15,35,0,0,0
30 A=PEEK(106)
35 ? "     HUP!"
40 DLL1=PEEK(560):DLH1=PEEK(561)
45 REM ******************************
50 POKE 106,A-8
60 GRAPHICS 5
63 COLOR 1:RESTORE 77
65 ? "     TWO!"
70 READ X,Y:IF X=0 THEN 80
75 PLOT X,Y:GOTO 70
77 DATA 19,23,20,23,21,23,19,24,20,24,21,24,20,25,18,26,19,
   26,20,26,21,26,22,26,17,27,19,27,20,27,21,27,23,27
78 DATA 17,28,19,28,20,28,21,28,23,28,19,29,20,29,21,29,19,
   30,20,30,21,30,19,31,21,31,19,32,21,32,18,33,22,33
79 DATA 18,34,22,34,18,35,19,35,20,35,22,35,23,35,24,35,0,0
80 DLL2=PEEK(560):DLH2=PEEK(561)
85 REM ******************************
```

```
90 POKE 106,A-16
100 GRAPHICS 5
103 ? "     THREE!"
105 COLOR 1:RESTORE 117
110 READ X,Y:IF X=0 THEN 120
115 PLOT X+20,Y:GOTO 110
117 DATA 9,23,10,23,11,23,9,24,10,24,11,24,10,25,8,26,9,26,
    10,26,11,26,12,26,7,27,9,27,10,27,11,27,13,27
118 DATA 6,28,9,28,10,28,11,28,14,28,9,29,10,29,11,29,9,30,
    10,30,11,30,9,31,11,31,8,32,12,32,7,33,13,33,7,34,13,34
119 DATA 7,35,8,35,9,35,13,35,14,35,15,35,0,0,0
120 DLL3=PEEK(560):DLH3=PEEK(561)
125 REM ******************************
130 POKE 106,A-24
140 GRAPHICS 5
143 ? "     FOUR!"
145 COLOR 1:RESTORE 157
150 READ X,Y:IF X=0 THEN 160
155 PLOT X+20,Y:GOTO 150
157 DATA 19,23,20,23,21,23,19,24,20,24,21,24,20,25,18,26,19,
    26,20,26,21,26,22,26,17,27,19,27,20,27,21,27,23,27
158 DATA 17,28,19,28,20,28,21,28,23,28,19,29,20,29,21,29,19,
    30,20,30,21,30,19,31,21,31,19,32,21,32,18,33,22,33
159 DATA 18,34,22,34,18,35,19,35,20,35,22,35,23,35,24,35,0,0
160 DLL4=PEEK(560):DLH4=PEEK(561)
165 REM ******************************
168 POKE 764,255
170 CH=PEEK(764)
180 IF CH=31 THEN 190
181 IF CH=30 THEN 200
182 IF CH=26 THEN 210
183 IF CH=24 THEN 220
184 IF CH=12 THEN 230
185 GOTO 170
190 POKE 560,DLL1:POKE 561,DLH1:GOTO 170
200 POKE 560,DLL2:POKE 561,DLH2:GOTO 170
210 POKE 560,DLL3:POKE 561,DLH3:GOTO 170
220 POKE 560,DLL4:POKE 561,DLH4:GOTO 170
230 POKE 106,A:RUN "D:NEXT4"
```

EXAMPLE 5

Notice line 7 which stores the value held in memory location 559. Then line 13 POKEs 559 with a 0. Well, this neat trick turns off the screen so that the pictures are drawn without your seeing them. It also speeds up the computer by about 30% depending on graphics mode. Want to know why? Send us $40. and we..., oh well, I'll tell you. Location 559 controls the ANTIC Chip which puts the video on the screen. Use the original value, stored in "NON" to turn it on. Use 0 to turn it off. Line 169 is where we turn the display back on.

THIS WORKS FOR ANY PROGRAM!

```
5 TRAP 10
7 NON=PEEK(559)
10 GRAPHICS 5
13 POKE 559,0
15 COLOR 1
20 READ X,Y:IF X=0 THEN 30
25 PLOT X,Y:GOTO 20
27 DATA 9,23,10,23,11,23,9,24,10,24,11,24,10,25,8,26,9,26,
   10,26,11,26,12,26,7,27,9,27,10,27,11,27,13,27
28 DATA 6,28,9,28,10,28,11,28,14,28,9,29,10,29,11,29,9,30,
   10,30,11,30,9,31,11,31,8,32,12,32,7,33,13,33,7,34,13,34
29 DATA 7,35,8,35,9,35,13,35,14,35,15,35,0,0,0
30 A=PEEK(106)
35 ? "      HUP!"
40 DLL1=PEEK(560):DLH1=PEEK(561)
45 REM ********************************
50 POKE 106,A-8
60 GRAPHICS 5
62 POKE 559,0
63 COLOR 1:RESTORE 77
65 ? "      TWO!"
70 READ X,Y:IF X=0 THEN 80
75 PLOT X,Y:GOTO 70
77 DATA 19,23,20,23,21,23,19,24,20,24,21,24,20,25,18,26,19,
   26,20,26,21,26,22,26,17,27,19,27,20,27,21,27,23,27
78 DATA 17,28,19,28,20,28,21,28,23,28,19,29,20,29,21,29,19,
   30,20,30,21,30,19,31,21,31,19,32,21,32,18,33,22,33
79 DATA 18,34,22,34,18,35,19,35,20,35,22,35,23,35,24,35,0,0
80 DLL2=PEEK(560):DLH2=PEEK(561)
85 REM ********************************
90 POKE 106,A-16
100 GRAPHICS 5
102 POKE 559,0
103 ? "      THREE!"
105 COLOR 1:RESTORE 117
110 READ X,Y:IF X=0 THEN 120
115 PLOT X+20,Y:GOTO 110
117 DATA 9,23,10,23,11,23,9,24,10,24,11,24,10,25,8,26,9,26,
    10,26,11,26,12,26,7,27,9,27,10,27,11,27,13,27
118 DATA 6,28,9,28,10,28,11,28,14,28,9,29,10,29,11,29,9,30,
    10,30,11,30,9,31,11,31,8,32,12,32,7,33,13,33,7,34,13,34
119 DATA 7,35,8,35,9,35,13,35,14,35,15,35,0,0,0
120 DLL3=PEEK(560):DLH3=PEEK(561)
125 REM ******************************
```

```
130 POKE 106,A-24
140 GRAPHICS 5
142 POKE 559,0
143 ? "     FOUR!"
145 COLOR 1:RESTORE 157
150 READ X,Y:IF X=0 THEN 160
155 PLOT X+20,Y:GOTO 150
157 DATA 19,23,20,23,21,23,19,24,20,24,21,24,20,25,18,26,19,
    26,20,26,21,26,22,26,17,27,19,27,20,27,21,27,23,27
158 DATA 17,28,19,28,20,28,21,28,23,28,19,29,20,29,21,29,19,
    30,20,30,21,30,19,31,21,31,19,32,21,32,18,33,22,33
159 DATA 18,34,22,34,18,35,19,35,20,35,22,35,23,35,24,35,0,0
160 DLL4=PEEK(560):DLH4=PEEK(561)
165 REM ****************************
168 POKE 764,255
169 POKE 559,NON
170 CH=PEEK(764)
180 IF CH=31 THEN 190
181 IF CH=30 THEN 200
182 IF CH=26 THEN 210
183 IF CH=24 THEN 220
184 IF CH=12 THEN 230
185 GOTO 170
190 POKE 560,DLL1:POKE 561,DLH1:GOTO 170
200 POKE 560,DLL2:POKE 561,DLH2:GOTO 170
210 POKE 560,DLL3:POKE 561,DLH3:GOTO 170
220 POKE 560,DLL4:POKE 561,DLH4:GOTO 170
230 POKE 106,A:RUN "D:NEXT5"
```

METHOD 2

This method differs only slightly from the first, but allows you more control of what goes on. The examples will explain the differences.

EXAMPLE 6

Instead of using a variable called "A", we use "P106" to store the original # of pages in your memory. This will be more meaningful to us. P106 stands for the value to POKE into location 106.

```
4 P106=PEEK(106)
5 ? "†":? "AT PAGE ONE!":? "PRESS 1 OR 2 FOR THAT PAGE."
7 ? "PRESS RETURN TO GO ON."
```

```
10 DP=PEEK(560)+PEEK(561)*256
12 POKE 16,64
15 SAV=PEEK(DP+5)
16 POKE 106,P106-4:POKE 89,SAV-4
17 ? "AT PAGE TWO!":? "PRESS 1 OR 2 FOR THAT PAGE.":? "PRESS
   RETURN TO GO ON."
30 POKE 764,255:TRAP 80
33 CH=PEEK(764)
35 IF CH=31 THEN 45
36 IF CH=12 THEN 60
37 IF CH=30 THEN 40
38 GOTO 33
40 POKE DP+5,SAV-4
43 GOTO 33
45 POKE DP+5,SAV
55 GOTO 33
60 POKE 106,P106:RUN "D:NEXT6"
```

Line 10 stores the location of the start of the DL as one decimal number. Line 15 store the number we are after. It comes 5 bytes after the start of the DL, so we PEEK at DL+5, ie. it is the sixth number in the DL.

Line 16 POKEs memory location 106 down by a number of pages (4 in this case). This line also stores a new value we need: location 89. This one is a copy of the value in DP+5 which tells the system where the start of the display data is. The computer looks at the DL whenever a GRAPHICS command is used, and stores that value here so that it will know where the start of your data is. After a Graphics call, we are free to change this number (in 89) to "fool" the computer into doing what we want. After POKEing both of these locations down far enough, we now write some text to the new area of memory on line 17. This could be done many times if you have enough RAM.

Now when lines 30 to 38 choose which screen you want, lines 40 or 45 just change the value in the first Display List that controls where the first DL gets it's data from. We don't care about a second (or 3rd or 4th...) Display List as in Example 1.

SO WHAT?

Well, now by just changing one value at the start of the first Display List+5 (DP+5), you can tell the system to go display different data from all over memory. This change could be easily controlled with a joystick as we do in our Scrolling Program, Tricky Tutorial #2.

Also, you might want to look at one screen while you are drawing several others; for example while a decision was being made about options on the first.

EXAMPLE 7

This does what we just suggested. You can not only look at two screens (press 0 for screen 1 and 4 for screen two), but by inputing other positive numbers you can look down in memory. By inputing negative numbers, you look up in memory until you reach the top. The stuff you see on the screen will be the alpha-numeric equivalent of the BASIC program, your screen data, the Operating System or whatever you are looking at. The only changes to this program are:

Lines 20 & 33 input a number.

Line 35 tests that number to see if it is too big.

Line 40 redirects the DL as before, but with your value instead of the previous fixed value of 4.

```
4 P106=PEEK(106)
5 ? "†":? "AT PAGE ONE!":? "PRESS START AND RETURN AT THE
  SAME    TIME TO GO ON.":? "PRESS BETWEEN 0 AND ";P106-5;
7 ? " TO LOOK AT    MEMORY IN 1/4 PAGE SCREEN INCREMENTS.":?
  "THEN PRESS RETURN. REPEAT AS DESIRED."
10 DP=PEEK(560)+PEEK(561)*256
12 POKE 16,64
15 SAV=PEEK(DP+5)
16 POKE 106,P106-4:POKE 89,SAV-4
17 ? "AT PAGE TWO!":? "   PRESS START AND RETURN AT THE SAME
   TIME TO GO ON"
20 DIM A(2)
25 POKE 53279,8
30 TRAP 30:Z=PEEK(53279):IF Z=6 THEN 80
33 INPUT A
35 IF A>(P106) THEN 60
40 POKE DP+5,SAV-A
55 GOTO 30
60 ? "NUMBER TOO LARGE, MUST BE LESS THAN";P106-5:? "WE ARE
   NOW AT 4 PAGES DOWN IN MEMORY"
65 POKE DP+5,SAV-4:POKE 89,SAV-4
70 GOTO 30
80 TRAP 40000:POKE 106,P106:POKE 89,SAV:POKE DP+5,SAV:RUN
   "D:NEXT7"
```

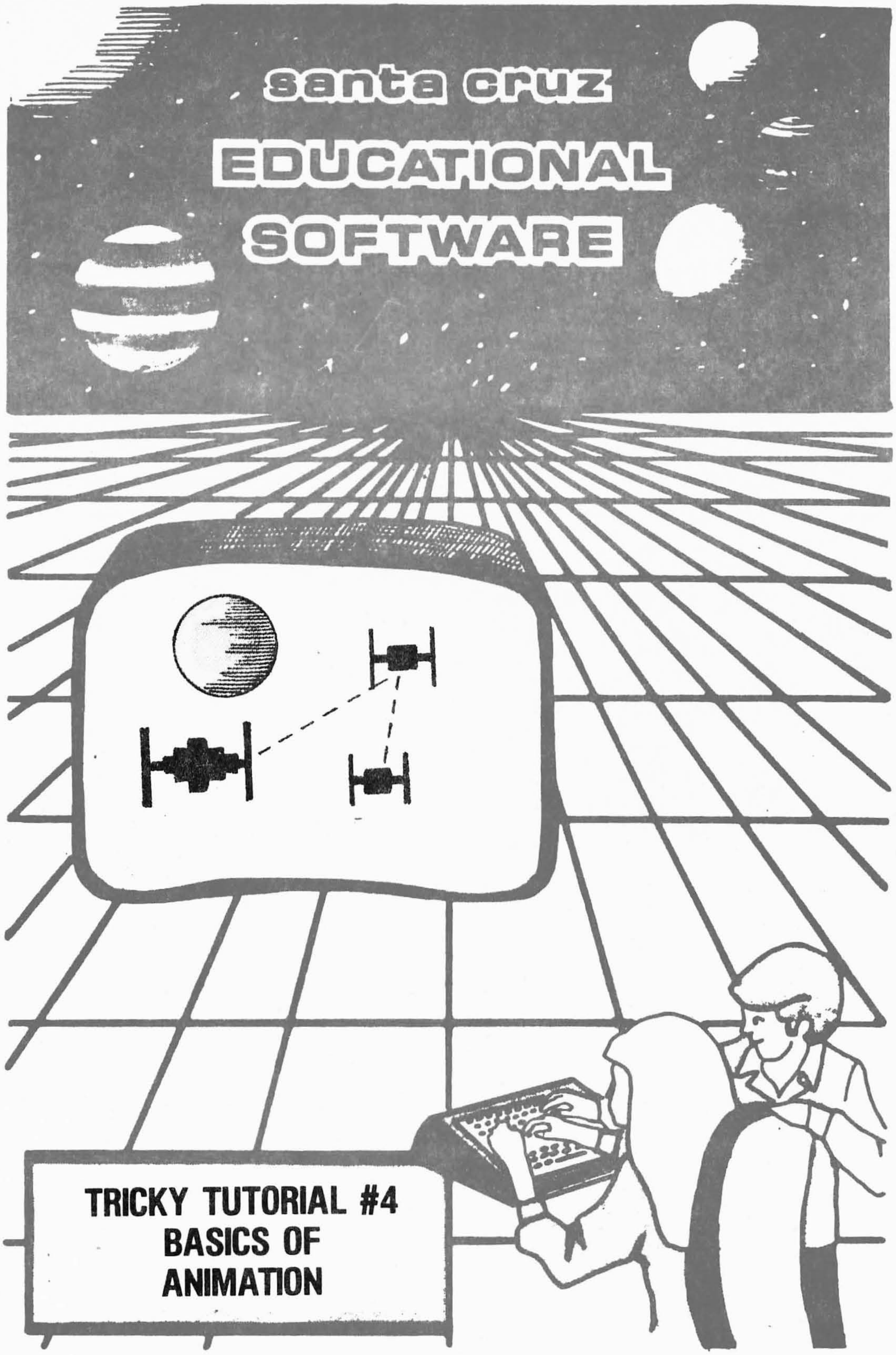
EXAMPLE 8

The last example is exactly the same as Example 7, except it looks at memory using a colorful graphics viewpoint of the data there. Be sure to try negitive numbers on the last two examples also. Since the program is designed to look up or down in memory, the negitive numbers look down while the positives look higher in RAM. You can look most anywhere from within the BASIC Cartridge to the data flowing in and out of the Operating System (try large negative numbers for this)...See if you can find an area that is changing for real special effects.

```
4 P106=PEEK(106)
5 GRAPHICS 5:COLOR 1:PLOT 10,10:DRAWTO 10,20:DRAWTO 40,20:
  DRAWTO 40,10:DRAWTO 10,10
10 DP1=PEEK(560)+PEEK(561)*256
12 POKE 16,64
15 SAV1=PEEK(DP1+5)
16 POKE 106,P106-8
17 GRAPHICS 5:COLOR 2:PLOT 20,20:DRAWTO 20,30:DRAWTO 30,30:
   DRAWTO 30,20:DRAWTO 20,20
20 DIM A(2)
21 DP2=PEEK(560)+PEEK(561)*256
22 SAV2=PEEK(DP2+5)
25 POKE 53279,8
27 ? "PRESS START AND RETURN TOGETHER TO GO ON."
30 TRAP 30:Z=PEEK(53279):IF Z=6 THEN 80
33 INPUT A
35 IF A>(P106) THEN 60
40 REM POKE DP1+5,SAV-A
41 POKE DP2+5,SAV1-A
55 GOTO 30
60 ? "NUMBER TOO LARGE, MUST BE LESS THAN";P106-5:? "WE ARE
   NOW AT 4 PAGES DOWN IN MEMORY."
65 POKE DP+5,SAV-4:POKE 89,SAV-4
70 GOTO 30
80 TRAP 40000:POKE 106,P106:POKE 89,SAV:POKE DP+5,SAV:RUN
   "D:NEXT8"
```

THAT'S IT

Just take any of our examples and try to modify them so that you can both understand the methods and make your programs look and run much cleaner. I hope you find many new uses for PAGE FLIPPING. Please write and tell me about your accomplishments using techniques in TRICKY TUTORIALS. BYE!!



santa cruz EDUCATIONAL SOFTWARE

TRICKY TUTORIAL #4 BASICS OF ANIMATION

BASICS OF ANIMATION

Basics of Animation is a set of simple programs designed to teach those new to computers how to make shapes appear to move around on the screen. The three methods demonstrated are animation using the PRINT COMMAND, PLOT COMMAND, and ATARI'S great PLAYER/MISSILE GRAPHICS.

The person using this lesson should be familiar with BASIC programming so that he or she can read the code that is included. Since the program is not protected, the user is encouraged to try their own modifications inorder to gain a greater perspective of the art of animation.

This program requires 16k of memory for TAPE users and 24k for those using DISK.

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408)476-4901

EDUCATIONAL SOFTWARE

presents

BASICS of ANIMATION

TRICKY TUTORIAL tm #4

TRICKY TUTORIAL #4

BASICS OF ANIMATION

by
Robin Sherer

HOW TO LOAD

TAPE...

Place the tape in your recorder, label side up. Make sure the tape is rewound, and the BASIC Cartridge is in place. Also, reset the counter to zero. Push PLAY on the recorder and type RUN"C: and press RETURN. If the program won't start to load, try positioning forward or backward a little. The easiest way to find the beginning is to listen to the "noise" on the tape with a regular recorder. When you find the steady tone, you have the beginning of the program. We recommend you write down the number on your recorder's counter as each program example starts. This will make it easier to find each part later on.

DISK...

To load and run the disk, first turn on your disk drive. When the busy light goes out, place the disk in the drive. Now turn on the computer, with the BASIC Cartridge in place. The program will load each part and run by itself.

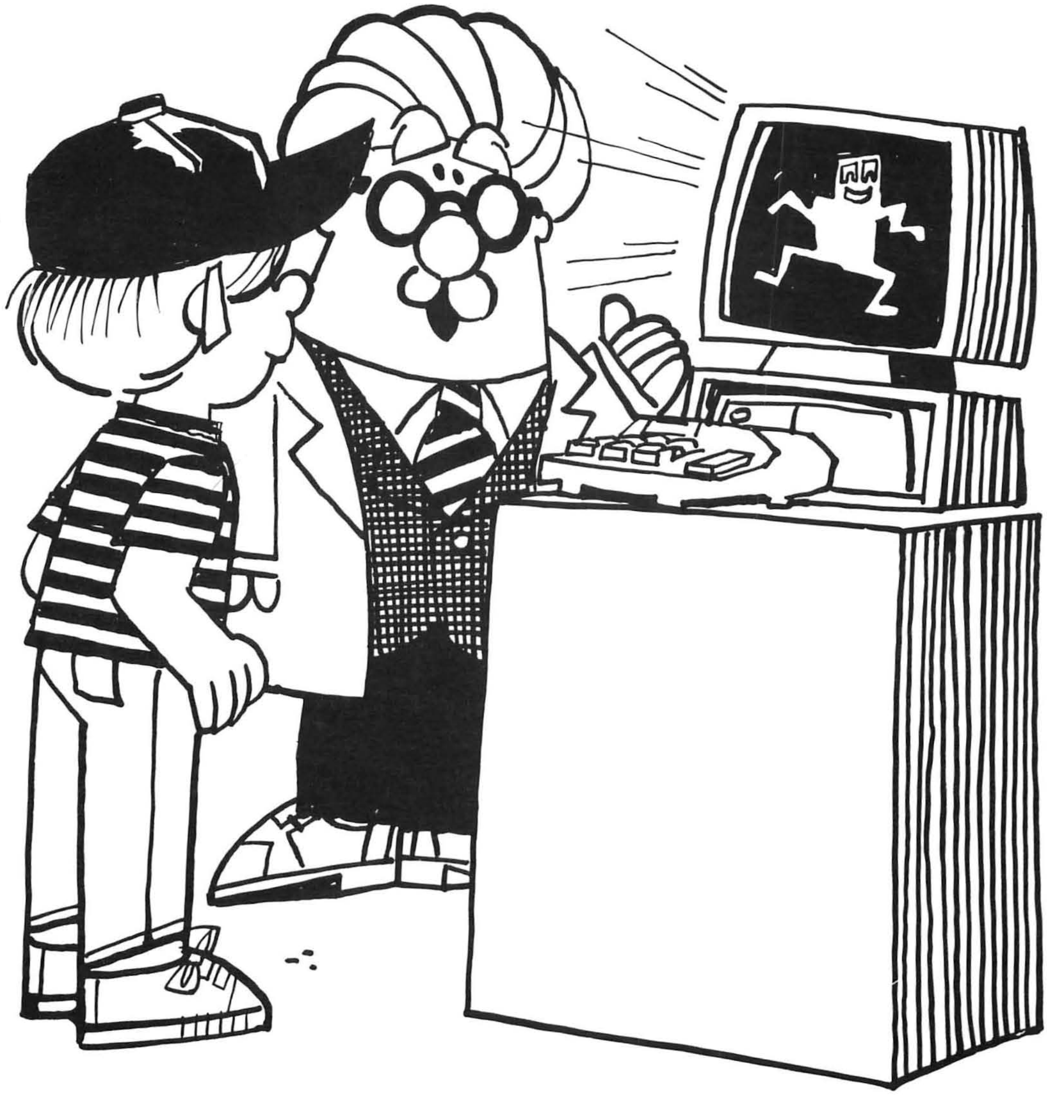
Any defective tapes or disks should be returned to:

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

LET THE ANIMATION BEGIN!

What does animation mean to you? Were you hoping that by purchasing this program you would then be able to do cartoons on the screen in intricate detail...or did you perhaps just hope to learn some basics to move a few shapes around the screen for a game or business application you are programming? Well, we're going to start out this lesson by discussing what can and also cannot be done on your ATARI.

Animation requires two main qualities to appear nicely on your screen. First, you must control enough points so that something seems to be happening. Then you must move these points around fast enough so that they appear life-like.



Rule number one:
IT CAN'T BE DONE FROM BASIC

Wait! Don't panic and return this program to us before you read on. We know you are not a Assembly language programmer. Neither are we, except when necessary.

BASIC in the ATARI is slow because it is an interpreter, meaning that every time you tell it to go move some point around on the screen, it has to first go and figure out exactly how to do what you've asked. This takes so much time that even with the machine doing hundreds of thousands of steps per SECOND, it can only move a single point around as fast as our first two examples. The way around this is to do one of four things:

1) You can program in the machine's language, where it doesn't have to interpret. This, however, would be too difficult for the average ATARI user.

2) You can use machine language routines built into the machine. A simple example of this is what we are doing in Part 2 using the PRINT command to draw some what complicated shapes very quickly. In fact if you look at the code for the space bug, you will see a delay loop was put in to slow it down! Also, although too complicated to explain in a introductory program like this, you can redefine the characters you are printing to draw almost anything you can imagine. The monsters in SPACE INVADERS are redefined letters of the alphabet, and are moved about by PRINT commands.

3) You could also go buy a larger BASIC that includes some machine language movement routines to move your shapes around. (Basic A+ or Microsoft Basic).

4) You can use PLAYER/MISSILE GRAPHICS like the tie fighter program we include as an example.

ANIMATION USING THE PLOT COMMAND

If you haven't already done so, now is the time to load in the program. Follow the instructions in the program and use your Joystick to move a simple square around the screen. This is similiar to the movement in games of the SURROUND type. As you can see while you follow the program , it is really quite simple to move a square! We show you the code here so you can see the necessary steps to move the square. Don't worry about learning it all now.

THIS IS THE CODE FOR MOVING A SQUARE:

```
10100 GRAPHICS 4:POKE 764,255:X=20:Y=12
10103 ? "HERE IS OUR SQUARE":? "USE YOUR JOYS
TICK TO MOVE IT ....WHEN DONE PRESS ANY KEY"
10105 V=STICK(0)
10110 Y=Y+(V=13)+(V=9)+(V=5)-(V=10)-(V=14)-(V
=6)
10120 X=X-(V=10)-(V=11)-(V=9)+(V=6)+(V=7)+(V=
5)
10145 IF X<1 THEN X=1
10146 IF X>79 THEN X=79
10147 IF Y<1 THEN Y=1
10148 IF Y>39 THEN Y=39
10150 COLOR 1:PLOT X,Y:IF PEEK(764)=255 THEN
10105
```

The better games of this type add another player to compete with and leave a controlled trail behind them. The trail is kept track of by the computer. Then, when you touch a location where the computer's "records" show a trail is, it scores for the other player

After you have run the program through to the point where we show you how to erase the trail, we suggest you stop the program (press BREAK and remove your tape or disk). You can always come back and finish it later. To continue with disk, type RUN "D:PLOT2.DSK". For tape, use RUN"C:". The reason for stopping now is for you to write a small program that makes use of a square being moved using the PLOT statement. The easiest way would be for you to look at the listing.

HERE'S THE CODE FOR MOVING A SQUARE, BUT WITHOUT A TRAIL:

```
10520 V=STICK(0)
10525 SETCOLOR 2,0,0:COLOR 2
10530 IF V=14 THEN Y=Y-1:PLOT X,Y+1
10540 IF V=13 THEN Y=Y+1:PLOT X,Y-1
10550 IF V=11 THEN X=X-1:PLOT X+1,Y
10560 IF V=7 THEN X=X+1:PLOT X-1,Y
10575 IF X<1 THEN X=1
10586 IF X>79 THEN X=79
10590 IF Y<1 THEN Y=1
10600 IF Y>39 THEN Y=39
10610 COLOR 1:PLOT X,Y:IF PEEK(764)=255 THEN
10520
```

Lines numbered from 10520 to 10610 are the ones that move the square around. Add to our program using your own ideas. Also, since the code exists in the program we sent you, please feel free to delete the lines you don't need, and start with them already in the computer to save time (the Assembler cartridge could do this quickly!). Please be sure if you save your work, it is on a separate tape or disk.

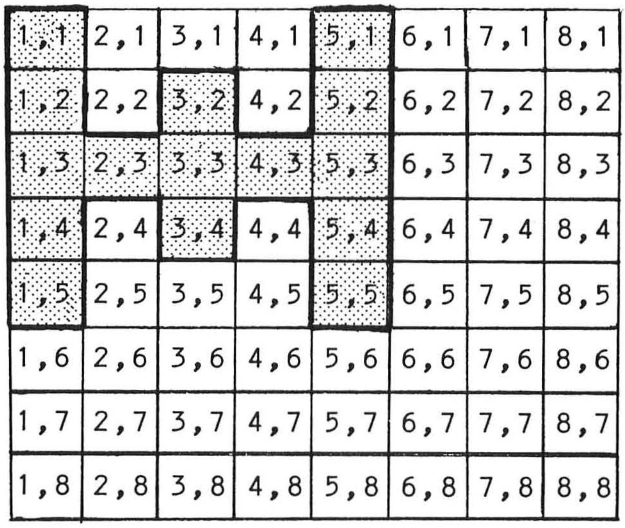
NOTE: Even if you intend to do only non-game applications, the learning you gain from writing a simple game will help other attempts at animation.

Before going on, we want to point out that when using PLOT and erasing your "trail", you have to actually PLOT the position you are erasing with the color of the background. This makes that position seem to disappear as you PLOT the next position in the color of your choice.

The next thing I do to show you the BASICS OF ANIMATION is to draw out the famous "Tie Fighter" shape and move it using the PLOT command.

PLEASE NOTE THAT THIS IS ONLY AN EXAMPLE AND THE SHAPE COULD BE ANYTHING FROM A BUSINESS TO A HOME APPLICATION. IT ALSO COULD BE MUCH LARGER THAN WE DREW. I keep our programs simple to allow everyone to understand the PRINCIPLES INVOLVED, but with more memory and time you can do great things like this:

Here is how to input points to plot:



1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1
1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2
1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3
1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4
1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5
1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6
1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7
1,8	2,8	3,8	4,8	5,8	6,8	7,8	8,8

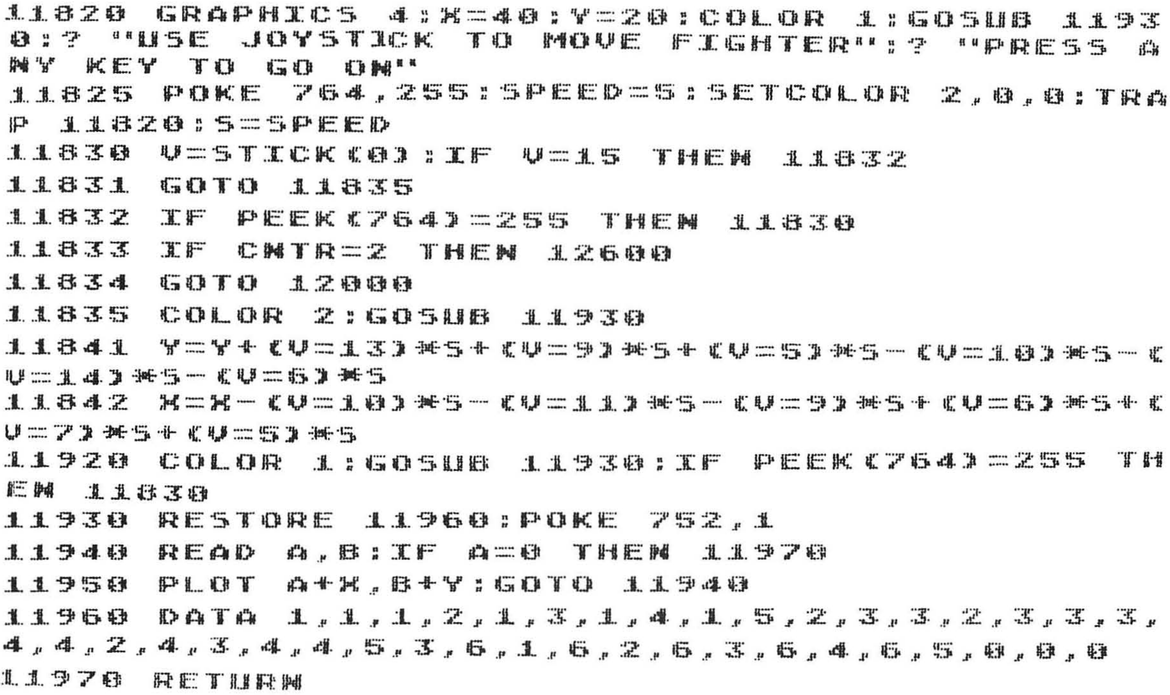
Figure 1.

Look at Figure 1. I drew out a shape at the upper left corner of what is called an X-Y coordinate system. The upper left most point that is filled in is at x=1 and y=1. Moving down the figure the next point is at x=1,y=2, then x=1,y=3, etc. with the last point of OUR shape at x=5,y=5. The computer doesn't care what order you input points into PLOT statements, just input them all. This means you don't have to use the points in the same order as we did, or even use the same points. Now, to put these into a BASIC program we use the DATA statement (read your BASIC manual!). It looks as follows:

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

Every two numbers represent one point to plot.

Here is the complete TIE FIGHTER code:



```
11820 GRAPHICS 4:X=40:Y=20:COLOR 1:GOSUB 1193
0:? "USE JOYSTICK TO MOVE FIGHTER":? "PRESS A
NY KEY TO GO ON"
11825 POKE 764,255:SPEED=5:SETCOLOR 2,0,0:TRA
P 11820:S=SPEED
11830 V=STICK(0):IF V=15 THEN 11832
11831 GOTO 11835
11832 IF PEEK(764)=255 THEN 11830
11833 IF CNTR=2 THEN 12600
11834 GOTO 12000
11835 COLOR 2:GOSUB 11930
11841 Y=Y+(V=13)*S+(V=9)*S+(V=5)*S-(V=10)*S-(
V=14)*S-(V=6)*S
11842 X=X-(V=10)*S-(V=11)*S-(V=9)*S+(V=6)*S+(
V=7)*S+(V=5)*S
11920 COLOR 1:GOSUB 11930:IF PEEK(764)=255 TH
EN 11830
11930 RESTORE 11960:POKE 752,1
11940 READ A,B:IF A=0 THEN 11970
11950 PLOT A+X,B+Y:GOTO 11940
11960 DATA 1,1,1,2,1,3,1,4,1,5,2,3,3,2,3,3,3,
4,4,2,4,3,4,4,5,3,6,1,6,2,6,3,6,4,6,5,0,0,0
11970 RETURN
```

The last part of this lesson allows you to move the cursor up to the data line we used for the tie fighter shape. The program is now stopped, waiting for you to enter new numbers into the DATA statements. If a mistake occurs, you can try pressing RESET and typing RUN, but if the DATA Statements get too messed up it may be easier to reload in the program. You probally won't have to go through this since it is easy to change the required lines in the program.

All you need to do is type in new numbers that you have chosen, based on a drawing of YOUR shape made on standard grid paper. Once the DATA statement looks correct, press RETURN to enter it and then type CONT and press RETURN again to restart the program. Also, you may use numbers bigger than the 8,8 for your shape, but when you move it, if it goes off screen it will bomb the program since we only allowed in our code for the smaller shape. The more points you use, the slower it goes. Again, feel free to copy and modify our program or, if you like, just slowly change a small part of it to see how your own changes can add to it or make it better!!!

ANIMATION USING THE PRINT COMMAND

We included the "Bird at the Ocean" from the ATARI BASIC manual as a convenience for those who never typed it in. It demonstrates the simple use of alternating between two shapes in a PRINT command in order to obtain a feeling of motion. We will use this method now. Again, after finishing this part please modify the shapes or the background to get a feel for your own ideas. Backgrounds are simply created with PLOT & DRAWTO commands and the special Graphics Characters. We next show you animation of a single character in the horizontal and vertical direction. They both have simple sounds to demonstrate how easy sound is to add.

HERE'S THE CODE FOR THE SPACE BUG:

```
21310 GRAPHICS 18:? #6;"NOW LETS TRY A SPACE
BUG":FOR W=1 TO 600:NEXT W
21320 TRAP 23512
21330 ? "↰":X=1:Y=10:POKE 752,1:CNT=0
21340 POSITION X,Y
21345 ? "  ":POSITION X,Y+1
21350 ? "╲╱":POSITION X,Y+2
21360 ? "◢◣":POSITION X,Y+3
21370 ? "▌ ▐"
21391 SOUND 1,210,10,CNT+2
21392 FOR W=1 TO 20:NEXT W:POSITION X,Y
21393 ? "  ":POSITION X,Y+1
21394 ? "  ":POSITION X,Y+2
21395 ? "  ":POSITION X,Y+3
21396 ? "  "
21397 X=X+2:POSITION X,Y
21400 ? "  ":POSITION X,Y+1
21410 ? "╲╱":POSITION X,Y+2
21420 ? "◢◣":POSITION X,Y+3
21430 ? "╲╲"
21431 FOR W=1 TO 20:NEXT W:POSITION X,Y
21432 ? "  ":POSITION X,Y+1
21433 ? "  ":POSITION X,Y+2
21434 ? "  ":POSITION X,Y+3
21435 ? "  "
21440 SOUND 1,160,10,CNT+4:X=X+2:IF X>37 THEN
21460
21450 GOTO 21340
21460 X=1:CNT=CNT+1:IF CNT=5 THEN 21480
21470 GOTO 21340
21480 SOUND 1,0,0,0:IF COUNT=1 THEN 22560
```

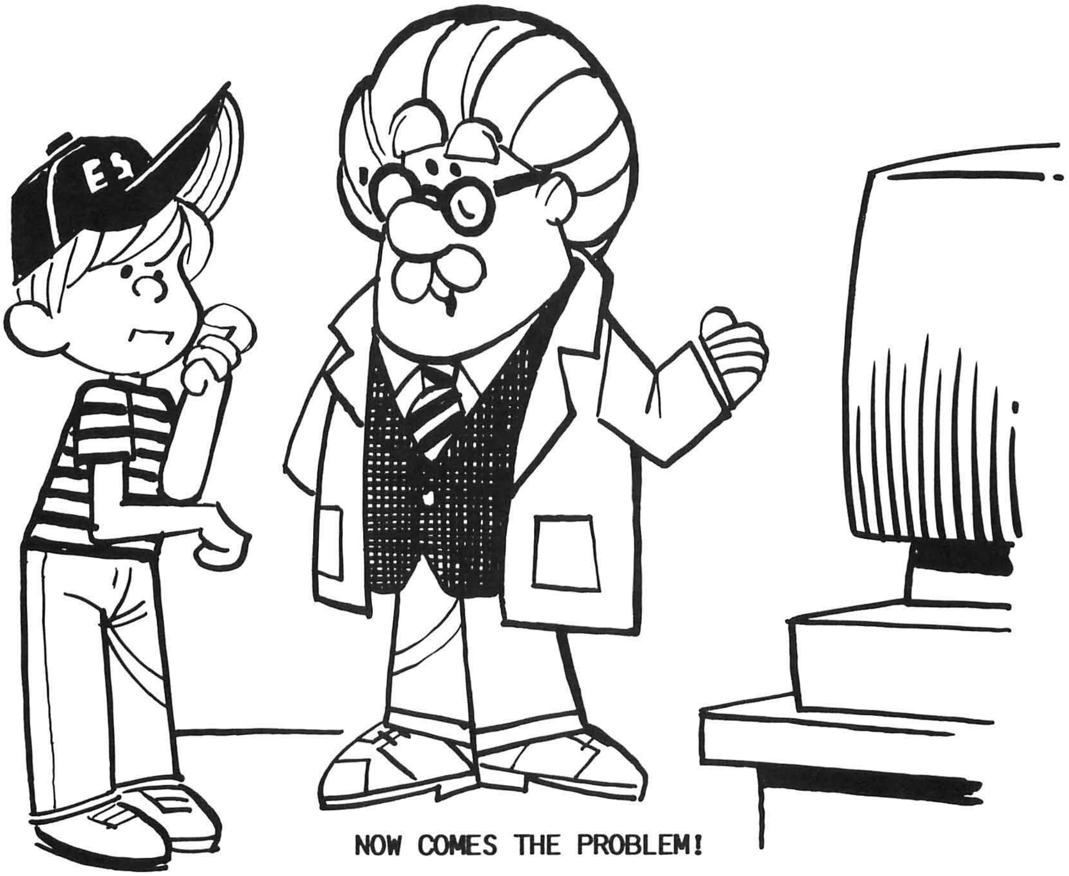
The program will stop and allow you to move the cursor up to the six lines that hold the shape for the "SPACE BUG". Change (using the special graphic characters) the shapes ONLY BETWEEN the quotes. When done, move the cursor to the top line and press RETURN until all six lines are re-entered into memory. If you goof up, clear the page and type GOTO 23500 and press RETURN. If you enter the new shape correctly move to a BLANK area on the screen and type "CONT", then press RETURN to see your shape move across the screen. When you have played enough with our code, try adding backgrounds first before your shape is printed or perhaps try several shapes. You will quickly notice a main drawback to animation using the PRINT command (and PLOT too); everything you move over is rewritten and thus disappears

-Note-

To do full animation many additional tricks are taught in our other Tutorials. This is not intended just to sell more programs, but you deserve to know how to use the full power of your ATARI. Some of these other Tutorials include PAGE FLIPPING, MODIFICATIONS TO THE DISPLAY LIST AND SCROLLING.

PLAYER MISSILE GRAPHICS

You probally already have heard of the ATARI's unique feature called PLAYER MISSILE GRAPHICS (PMG). What PMG does is allow you to animate simple shapes around on the screen without having to redraw them as we did in the earlier lessons using PRINT and PLOT. This capability is built into the hardware, so all we have to do is program the hardware with simple POKE commands.



NOW COMES THE PROBLEM!

What makes PMG hard to explain is that there are so many built in features, and so many POKEs to do. We wanted to reach a compromise with an introductory lesson like this, so we include an example using PMG with some explaination. Like before, if you will just take the time to modify our program you will see the effects of each change. It would be even better if you had a specific goal in mind, say to change the shapes of the Players, or a different background. Please note that the way this example was written is not the best it could have been done. It was writtrn long ago, but it works and that was the goal. The various POKEs are listed in more detail in our MASTER MEMORY MAP, and PMG is more fully explained in a seperate Tutorial(#5).

HERE'S THE PGM CODE:

```
20 K=15:REM INITIAL POS OF P/M 1
30 TRAP 20
40 SOUND 3,13,8,2
50 PRINT "K"
60 GOSUB 680:REM BACKGROUND SUBROUTINE
70 SETCOLOR 2,0,0:X=120:Y=90:REM SET BACKGROU
ND COLOR AND PLAYER POSITION
80 POKE 704,133:REM SETCOLOR PLAYER0 TO BLUE
90 POKE 705,198:REM SETCOLOR PLAYER1 TO GREEN
100 A=PEEK(106)-8:POKE 54279,A:PMBASE=256*A:R
EM SET PLAYER-MISSLE STARTING ADDRESS
110 POKE 559,46:POKE 53277,3:REM ENABLE PM GR
APHICS WITH 2-LINE RESOLUTION
120 POKE 752,1:REM MAKE CURSOR INVISABLE
130 FOR I=PMBASE+512 TO PMBASE+640:POKE I,0:N
EXT I:REM CLEAR OUT PLAYER0 FIRST
140 FOR I=PMBASE+640 TO PMBASE+768:POKE I,0:N
EXT I:REM CLEAR OUT PLAYER1 FIRST,THIS IS TO
PREVENT RANDOM JUNK.
150 FOR I=PMBASE+640+K TO PMBASE+644+K:READ B
:POKE I,B:NEXT I:REM DRAW PLAYER1
160 DATA 153,189,255,189,153
170 REM DATA FOR TIE SHAPE
180 FOR I=PMBASE+512+Y TO PMBASE+516+Y:READ A
:POKE I,A:NEXT I:REM DRAW PLAYER0
190 POKE 53256,0:POKE 53260,18:REM SIZE OF PL
AYER0 AND ALL MISSLES
200 POKE 53257,0:REM SIZE OF PLAYER1
210 DATA 153,189,255,189,153
220 POKE 656,3:? "PRESS 1 TO STOP GAME"
230 REM SAME SHAPE FOR OTHER PLAYER
240 REM NOW COMES THE MOTION/COLLISION ROUTIN
ES
250 IF PEEK(764)=31 THEN 1650
260 F=PTRIG(0):REM READ TRIGGERS
270 G=PTRIG(1)
280 IF F=0 OR MIS=1 THEN GOSUB 390:REM SLOW M
ISSLE ROUTINE
290 IF G=0 THEN GOSUB 520:REM FAST MISSLE MOV
E ROUTINE
300 POKE 656,0:IF FS=1 OR GS=1 THEN ? "GREENS
 SCORE=";SCOREG:"            BLUES SCORE=";SCOREB
310 FS=0:GS=0:REM THESE TELL ATARI TO PRINT S
CORE
320 POKE 53278,0:REM CLEAR OUT COLLISION REGI
STERS TO USE AGAIN
330 A=PADDLE(0)
340 B=PADDLE(1)
350 POKE 53248,A:IF MIS=0 AND A<>0 THEN POKE
53252,A-1:REM MOVE PLAYER0(AND MISSLE0) TO N
EW LOCATION INSTANTLY!
```

```
360 POKE 53249,B:POKE 53253,B-1:REM SAME FOR
PLAYER1
370 GOTO 250:REM GO READ PADDLES AGAIN
380 REM SOUBROUTINE FOR BLUE  MISSILE
390 SOUND 0,227-V*2,8,INT(16-V/10):REM MAKE S
OUND DECREASE AS MISSILE MOVES
400 V=V+1:POKE PMBASE+384+Y-V,0:POKE PMBASE+3
83+Y-V,1
410 E=PEEK(53256):IF E>0 THEN HIT=1:REM TEST
FOR HIT!
420 IF V>95 THEN 500:REM MISSILE MOVED FAR EN
OUGH
430 MIS=1:RETURN
440 HIT=0:FS=1:REM SAYS TO PRINT NEW SCORE,MI
S & HIT ARE COUNTERS
450 SOUND 1,221,10,12:POKE 53249,250:FOR W=1
TO 200:NEXT W
460 SOUND 1,0,0,0
470 POKE 53249,120:POKE 53278,0:RcM REPOSITIO
N PLAYER1 AND CLEAR COLLISION REGISTER
480 SCOREB=SCOREB+1
490 V=0:MIS=0:SOUND 0,0,0,0:RETURN
500 IF HIT=1 THEN 440
510 V=0:MIS=0:SOUND 0,0,0,0:RETURN
520 SOUND 0,200,8,8:P=0
530 FOR I=15 TO 130:POKE PMBASE+383+I,0:POKE
PMBASE+384+I,4:  P=PEEK(53257)
540 IF P=3 THEN 590
550 NEXT I
560 SOUND 0,0,0,0
570 RETURN
580 REM P=PEEK(53257):IF P=2 THEN RETURN
590 SOUND 0,0,0,0:GS=1:REM SAYS TO PRINT NEW
SCORE
600 RESTORE 630:SOUND 1,121,8,12
610 FOR I=PMBASE+512 TO PMBASE+640:POKE I,0:N
EXT I:REM CLEAR OUT PLAYER FIRST
620 Y=75:FOR I=PMBASE+512+K+Y TO PMBASE+516+K
+Y:READ B:POKE I,B:NEXT I:REM DRAW PLAYER
630 DATA 153,189,255,189,153
640 FOR I=384 TO 512:POKE PMBASE+I,0:NEXT I
650 SOUND 1,0,0,0
660 SCOREG=SCOREG+1
670 RETURN
680 GRAPHICS 8:CNT=0
690 COLOR 1:X=RND(0)*319:Y=RND(0)*159:CNT=CNT
+1:PLOT X,Y:IF CNT>150 THEN RETURN
700 GO TO 690
```

Player Missile Graphics Explanation

LINE 50 : Clear the screen

LINE 70 : The X and Y position of player one is established here.

LINES 80-90 : Player colors are stored in 704 to 707.

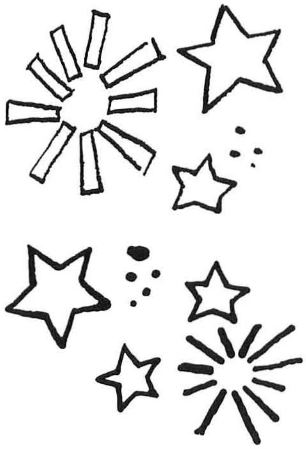
LINE 100 : Memory location 106 holds the value (in # of PAGES) of your top of memory. We simply subtract 8 pages to make room for the shapes of the Player to be stored.

LINES 130-140 : To create or erase a player, you POKE the shape into memory where we reserved it in line 100. We first erase the player by Poking in all 0's. Player 0 starts at the value of PMBASE plus 512 and goes for 128 locations in memory. Player 1 starts at 640.

LINES 150-160 : Read the shape into memory.

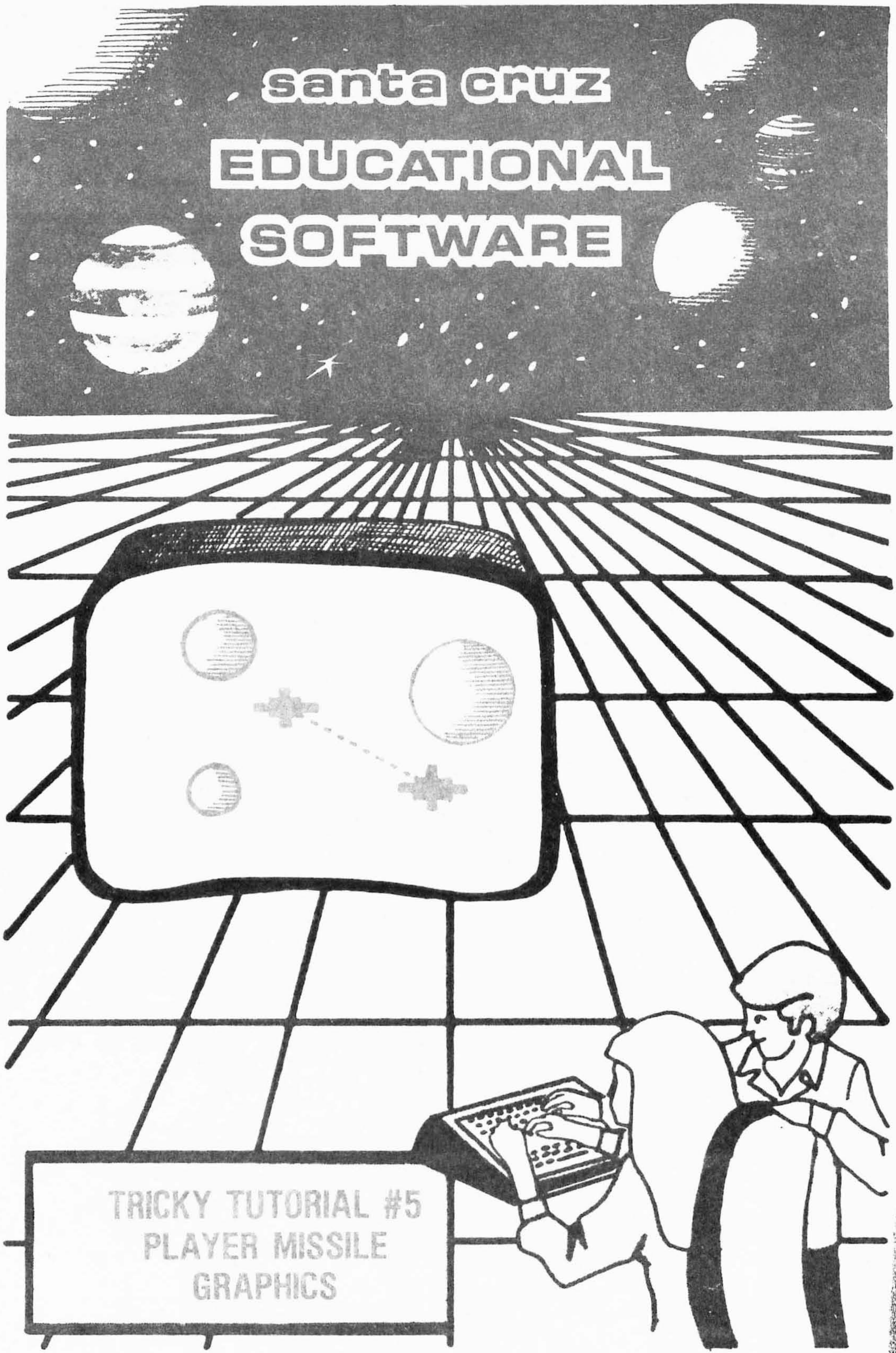
LINES 250-370 : Main loop of program. Location 764 holds the last key pressed. When you run the program, you will note that the missiles don't move the same. One moves in a loop that is fast because it doesn't go back and allow any players to move. The other is slow because it does. The collision registers mentioned in the comments to line 320 "record" when certain things touch eachotheron the screen. You determine what this location looks for by the value you put into 623 (if desired). Lines 350 & 360 POKE the location that controls where ACROSS the screen the Players will appear. If you don't plug in Paddles the values transfered here will be 0, so the players will be off screen. Values of about 40 to 200 will be on the screen.

LINES 390-finish : the rest is really just two subroutines to move the missiles up and down the screen. This is done by erasing the shape at it's current location, and drawing it at a new place in memory placing 0's at the old memory locations, and placing 1's in memory). If you draw it close to the last location, the motion will be slower, but smooth. We also add to the scores here when the collision registers at 53256/7 don't hold a 0. Finally, the routine at line 690 just creates stars by random PLOTs on the screen.



The only way that all of these POKEs will become familiar to you is to get a hold of one of the publications that gives detailed descriptions of each memory location needed for PMG. Our Master Memory Map is good, and ATARI's Operating System manual is better (but very hard to read). We hope you feel that this lesson was worth the cost.

Thanks.....bye!



santa cruz EDUCATIONAL SOFTWARE

TRICKY TUTORIAL #5 PLAYER MISSILE GRAPHICS

EDUCATIONAL SOFTWARE

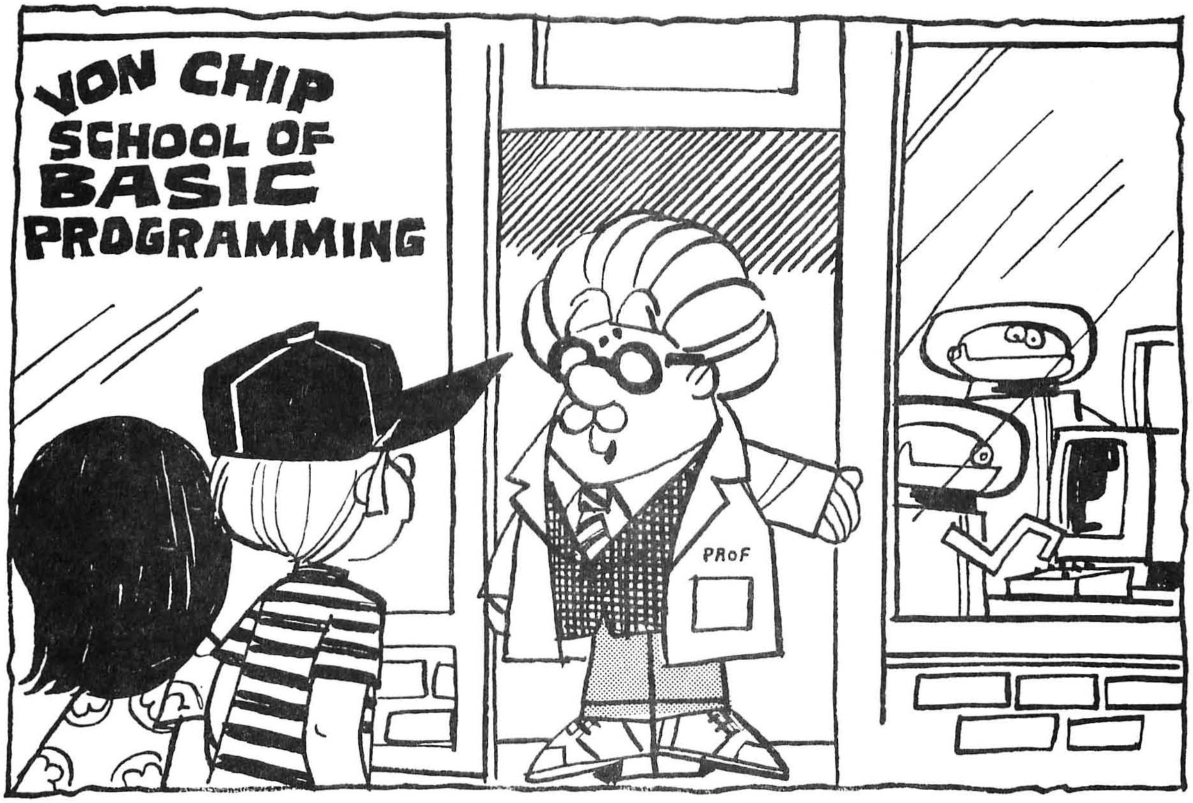
presents

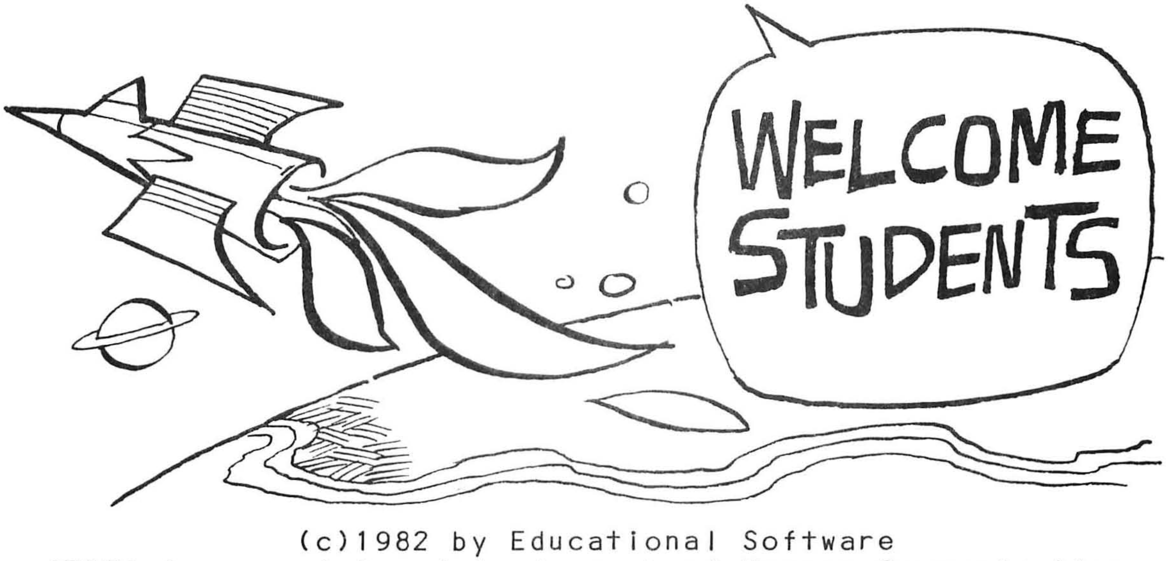
TRICKY TUTORIAL #5

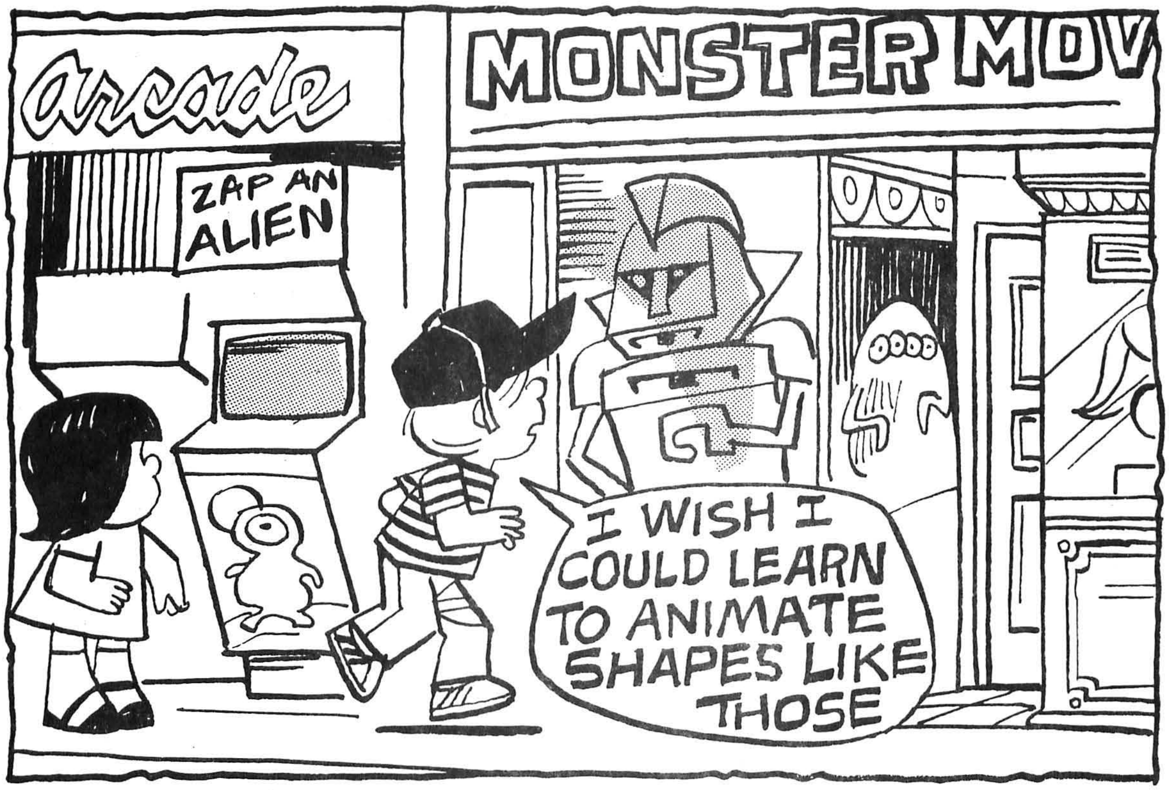
PLAYER MISSILE GRAPHICS

by

ROBIN ALAN SHERER







YOU CAN ANIMATE YOUR OWN CHARACTERS .....
WITH PLAYER MISSILE GRAPHICS (PMG)!

Ok! You spent your hard earned money on this program because everyone says that Player Missile Graphics is the way to go, but you know it is too hard to ever really understand. Besides, you bought $25 worth of magazines for the two page articles on the subject, and they all said the same thing (which made little sense to you). After all that, you got ATARI's book, DE RE ATARI, which was great reading, but Chris Crawford only explains the basic capabilities of PMG. How do you put it all to practical use? Well.......

Were Going to
Write a GAME!

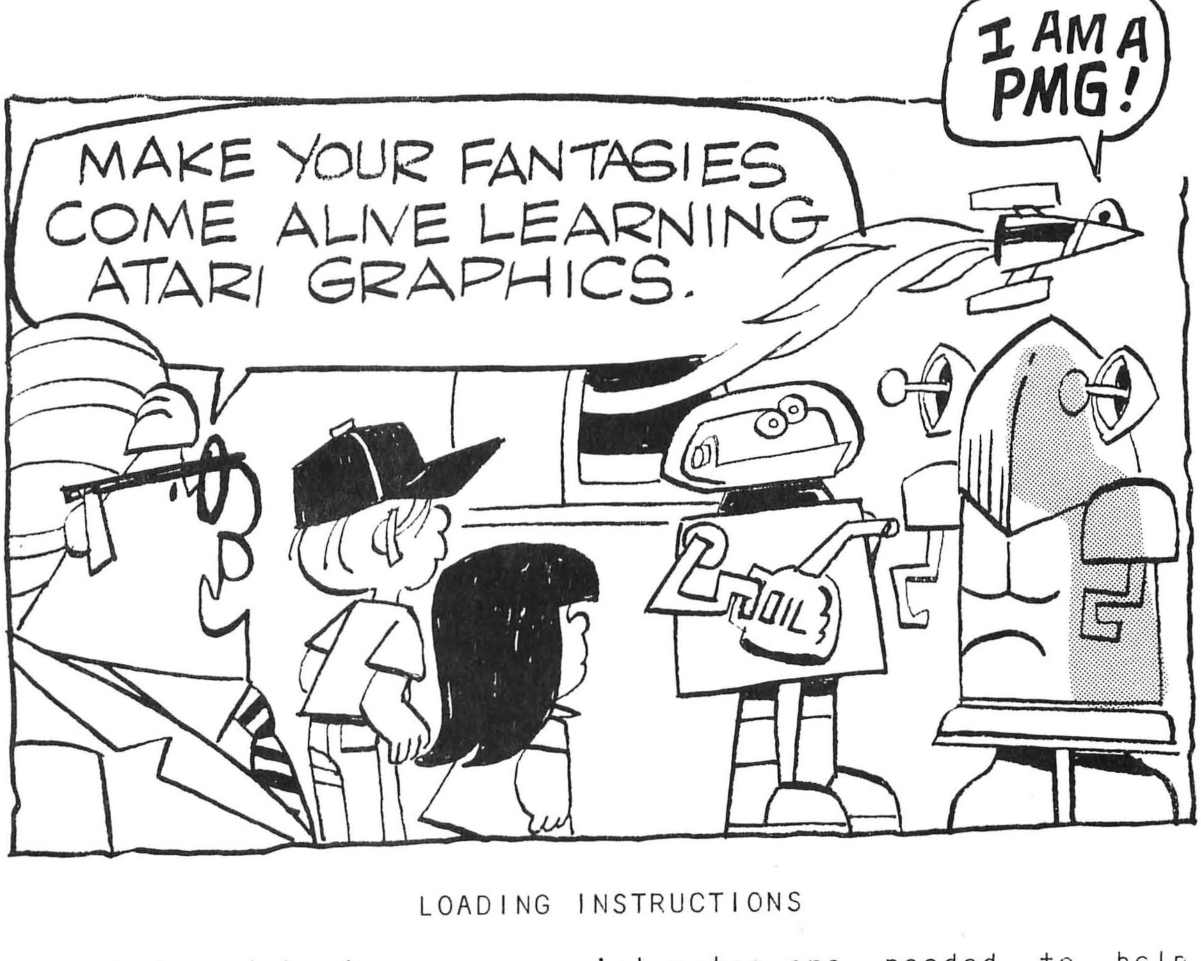
Most of us like to play arcade games, and since most arcade games use cute little figures and a maze, we will design our game that way. One of my favorite games is called PACMAN (tm). We can't actually use the same game as the arcades due to legal problems, but we can take the cute little character and put him into our our own hair raising situation!

THIS IS A
PAC PERSON!

All games take imagination. Let's put ours to work making up all kinds of shapes to animate and "play" with on the screen. Some of these characters will be used in demonstrations within the 14 main programs. You can take over where these lessons leave off and create any kind of application you wish. Player/Missiles is not just for games. Any time you need color or movement in your programs PMG is the best way to get the job done. For example, ATARI's SCRAM program, a Nuclear Plant Simulator uses Players and Missiles for the various tanks and mechanisms on the screen.

WHAT IS A PLAYER/MISSILE?

The names Players and Missiles come from the original use of these objects as guns (for the Player to use) and bullets or Missiles for the "shot" that was fired. Now, of course, many other uses exist, but the names are still used. Let's begin by looking at the original use of PMG, a space game demonstration. Run the first program now and follow along in the discussion.......



LOADING INSTRUCTIONS

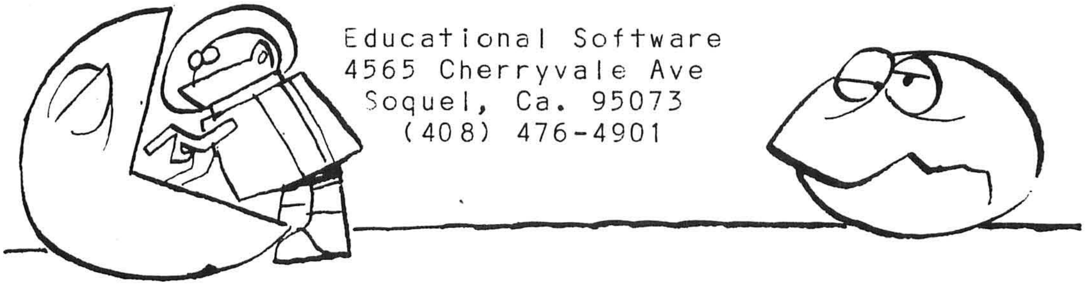
Before I begin, some special notes are needed to help tape users and those with only 16K of memory. When teaching about Player Missile Graphics, I can't help but get carried away. Orbie, Pixel, and I were having so much fun, that we wrote too many examples for this TUTORIAL. Once you finish this lesson, you will find yourself creating all kinds of animated creatures! Fortunately, my Robot helper, Prototype, ate a few of the programs, leaving us with just enough examples to fit a very full tape.

Because of the length of the lesson, some of you may have trouble reading in the programs on your 410 tape player. You will have to get 14 successful LOADS; so, as you LOAD each program , SAVE it on your own backup tape. Before you start, wind the tape forward and backward once. To load, reset your recorder's counter to zero. Now type RUN "C:" and press return twice. The tape should begin to load into memory. There is a backup copy of all examples on the back of your tape, but If loading still produces an error message, you have two options:

1) Rewind the tape to zero, take it out, and listen to it in a regular cassette player. Listen for the beginning of the steady tone that preceeds the digital signal. This tone will continue for about 20 seconds, then the program begins to load. If you are not sure the tape is at the beginning, just rewind or advance it a little bit until you find a place where there is no steady tone, then move back to the start of the tone. Now place it back into the 410 recorder and reset the counter to zero since you now know this is the right place. If the tape loads from here...great. Remember, SAVE each program as you get a successful LOAD on another backup cassette. Prototype sys to remind you to write down the number your Tape player shows where each of these 14 (12 for 16K users) examples start. This would be a good habit to develop when using anyone's cassette based program.

2) You can "guess" at the correct spot by going back and forth until you get lucky, but that is very frustrating!

Don't forget, if any program doesn't load properly, you will find backup copies on the reverse side of the cassette. If the program STILL doesn't seem to load properly, try another recorder (perhaps at a store or a friend's house). If you are so upset at the darn thing you want to buy a disk drive, send it back for a prompt (I hope!) replacement to:



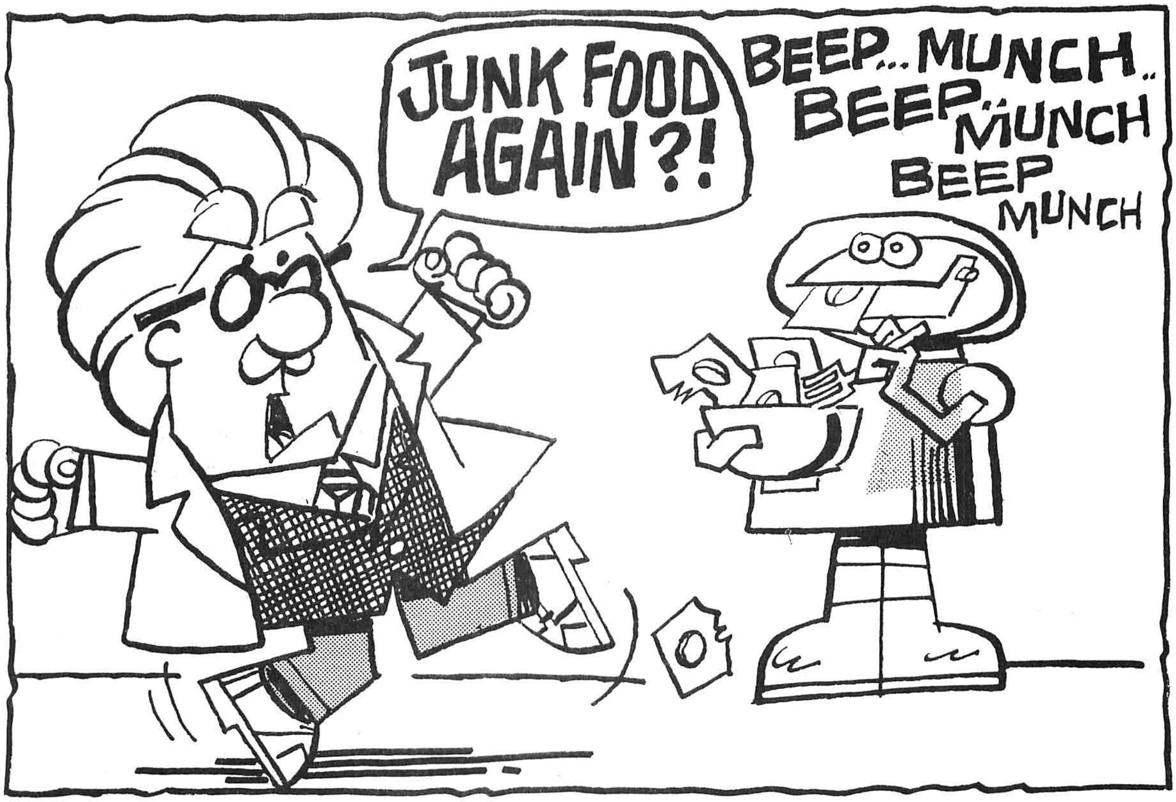
Educational Software
4565 Cherryvale Ave
Soquel, Ca. 95073
(408) 476-4901

- 16k Users -

I haven't forgotten you. All programs on this tape except the two utilities will run on your machines. The utilities are very nice, but not necessary, so just save them until you increase your memory size to 32K. Example 5.4, the Playfield editor, is out of order on your tape. It will be found after example 5.13 (Two players together), which is the last example that runs in 16K. Finally comes example 5.14, the last editor, which also requires 32K.

- HOW TO USE THIS MANUAL -

Without the information in this manual, these programs are rather boring. Only when you understand what is being shown in the examples will Player/Missle Graphics (PMG) begin to make sense to you. In fact, after you have written your first real game or business application you'll feel a wonderful sense of acomplishment!



As you progress you may find yourself lost on a certain subject. IMMEDIATELY refer to the FOLLOWING CHART to see which examples demonstrate the feature in question. Note that I refer to Example 5.1 as 1 since we all know this is Tutorial #5. When the chart says "others" this means that the other examples listed demonstrate the topic also, but not as well as those listed. Here then is :

THE FOLLOWING CHART

TOPIC	SEE EXAMPLES
General Features of PMG	1,2,14
Playfields (backgrounds)	2,4,8
Color Selection & Pokes	5
Memory Area for PMG	7 + others
How to Draw a Player	6,7,14, others
Missiles used as a Player	5,14
Joystick or Paddle?	7
Use of Sound with PMG	1,2,8
Moving Players and Missiles	1,2,6,7,9
Size of Players and Missiles	6,14
Single/Double line Resolution	10
How to Animate a Player	2,11,12
Priority	2,12
Collisions	2,12
Two Players as one	13
Complete Game	2
PMG using Strings	4,14
Poke and Peek Table	Appendix

EXAMPLE 5.1

If you are new to the ATARI computer, watch this example many times and look at it later as each subject is mentioned. Here is what you are seeing. First, the screen on your TV goes black, because I turned off the display processor, called ANTIC, accomplishing two things:

1) Speeding up the machine by 30%, and
2) Making the background of stars suddenly appear, rather than seeing them slowly drawn out.

You can compare this to watching the background drawn out in the next program and choose which method to use. If you decide to turn off the screen display while YOUR backgrounds are drawn, here's how:

At the point you want the screen to "turn off"

```
100 ON=PEEK(559):POKE 559,0

.....(PROGRAM CODE TO DRAW YOUR BACKGROUND)


500 POKE 559,ON
```

Later I'll teach you what the correct number for "ON" is, but you can always just save the old value as I did above. The number in memory location 559 will have to be changed later to allow PMG to work.

Yes, the ATARI does have two microprocessors. The ANTIC is a special one that is used to draw the screen on your TV or Monitor. It does both graphics and text. Whenever it is in use, it "steals cycles" from the main microprocessor, the 6502. This is why the computer is faster when the screen is blank(559=0).... the main processor runs at full speed if it doesn't have to stop all the time to allow the ANTIC time to draw the screen.

Are you still in Program 5.1? Good, don't rush ahead. This is going to take time to explain properly.

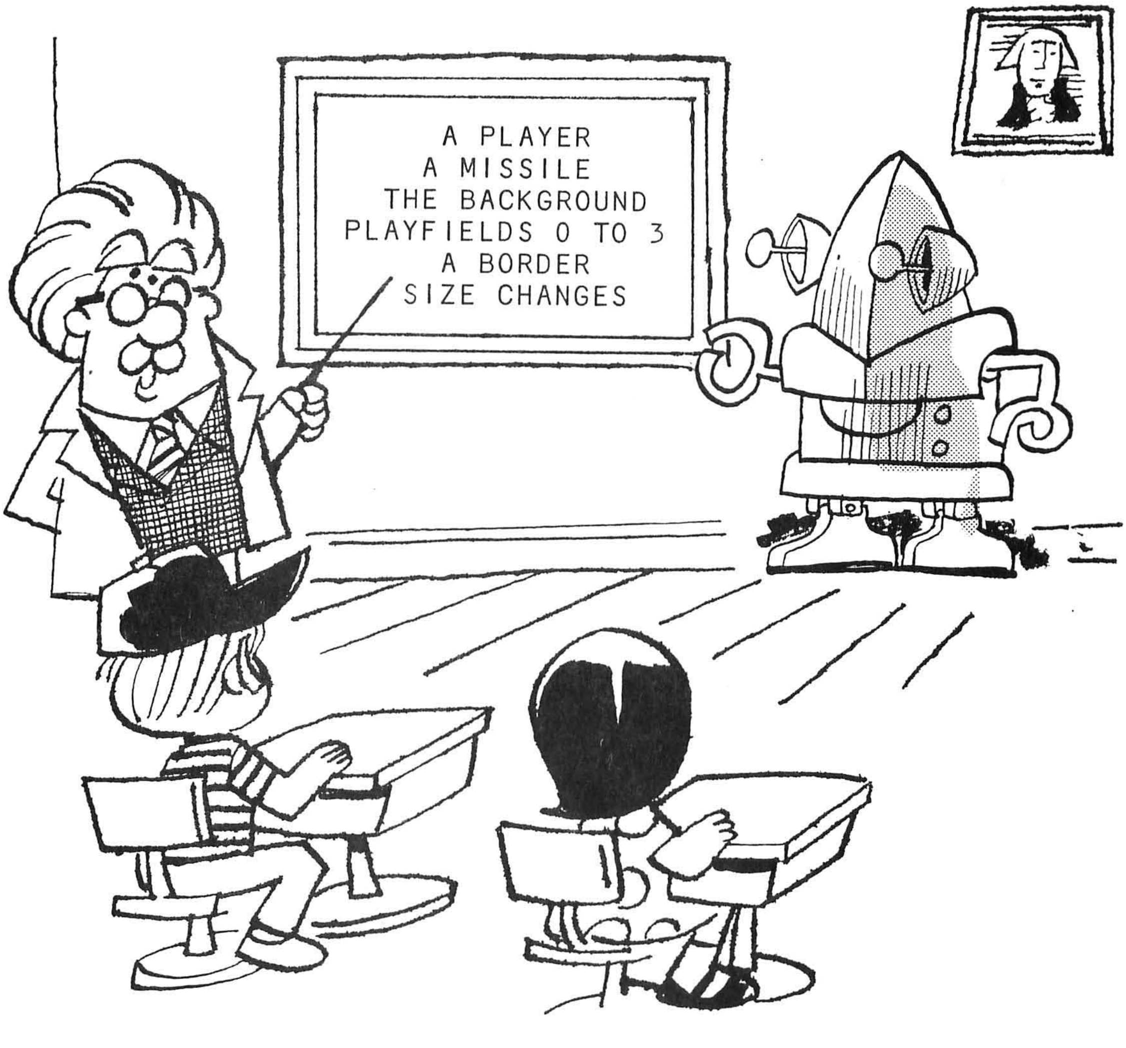
The computer has a nice bunch of "stars" on the screen. Some are red and some blue. This is because when we plot single points in Graphics 8, the TV can only light up either a blue or a red pixel on the screen. If you look real close at your tube you will see these tiny pixels of red and blue. These background stars are called a PLAYFIELD when discussing PMG. If you look at the graphics chart on page 53 in your BASIC manual, you'll see that most graphics modes allow from one to four colors. Modes 9 to 11 have more, but still only four of the colors pertain to the following discussion.

When you enter color 1, and then draw out some shapes on the screen, you are drawing what is called PLAYFIELD 1. Likewise, if color 2 or 3 is allowed in the graphics mode you are using, these would be PLAYFIELD 2 or 3. The fourth playfield is actually color 0, the background color that you get when you start out in a graphics mode. All of these colors can be controlled.

I usually set the border to the background playfield color so the whole screen appears the same. Notice how the stars don't actually go to the edge of the screen, but since the Players can move anywhere, having the border the same color makes the playfield seem larger.

At last we have two Players on the screen. It has been a long winded discussion, but there they are. And, oh my! They move! Oh, you knew they could? Well, I guess there is no suprising you. Notice how easily BASIC can move them HORIZONTALLY across the screen. Now look at the listing for this program. Notice 5 lines (400 to 440) control the sound and movement (You ARE going to study the code, aren't you?). Next, the Players grow in size. This capability also is a simple POKE of a number, and will be explained later.

Finally our space battle concludes with one ship firing missiles at another until they both are destroyed and the example is over. Run the example again until you are sure what the following mean:



OUR COMPLETE GAME
EXAMPLE 5.2

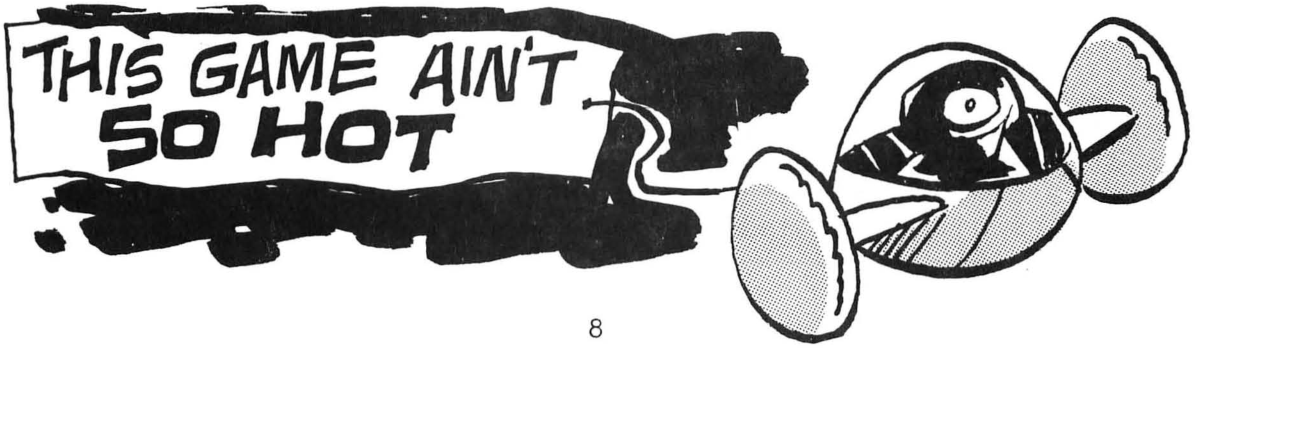
Rather than talk about some strange ideas like collision registers now, I thought you would like to see and play with the goal of this Tricky Tutorial, an actual game using all of the feature of PMG. Now before you begin, please realize that our goal did not include producing a game just like the arcades. Those are written in machine language with special display tubes to allow very fine graphics. We just want to demonstrate all of the topics you will be learning later in one program. For the hot programmers in the audience, the game can be expanded to be quite a lot of fun. As it is now, it just fits in 16k which was required for our friends with ATARI 400's (and there are a lot of you out there).

The "rules" of the game are easily changed. In fact, the first thing you might do when you finish the complete lesson is to change the game to your own specifications. Most changes will be surprisingly simple. I'll offer some hints when the game is explained at the end of the Tutorial. Until then, here are my rules:

HOW TO PLAY THE GAME

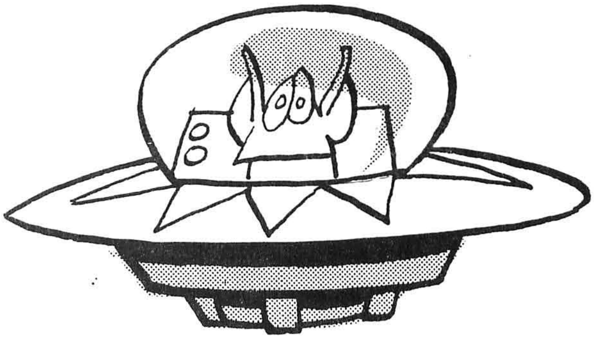
Plug in joysticks into ports 1 and 2. Each Player moves his character and tries to score points. One point is scored each time you "gobble" one of the two "Energizer Pellets" that will appear from time to time. Five points are scored if you gobble your opponent. This is done by first touching the blue recharge zone in the center of the maze. Your character will change color and may score by touching the other Player. If the other Player touches the recharge zone, then he can get you and your color changes back to normal.

I started the scores out at three each just to point out that everything doesn't have to begin at zero. The logic to the game is to balance your moves between going for easy single pointers and the big score of five points!



That's true, but think of this exciting fact. All of the basics techniques of real arcade games are included in this little game. The only improvements needed are your own ideas and SPEED. Arcade games are written in machine language which accounts for the speed. The ideas are up to you!

Notice the way the characters move. They are slow, yet I am moving them with machine language routines (these will be explained later). What slows them down is the Basic language needed to call them. Also, notice what happens when the characters touch the walls of the maze. After you have a real idea of what features are in the game, run example 5.3 and we'll begin the lesson.



EXAMPLE 5.3
PLAYFIELDS

This example is so simple to understand that my daughter even helped me write it. The basic idea I already mentioned above, but I'll repeat myself just this once (or twice). The stuff you see on the screen that is not a Player or Missile is called PLAYFIELD. Playfields are created by plotting points using the PLOT and DRAWTO commands, or by direct POKEing of data into the memory area that is displayed on the screen. You should already be familiar with PLOT and DRAWTO from your BASIC manual. If not, read about them in the manual and then come back.

PRESS 1 on your keyboard and plug in a joystick into port #1. Move the joystick around to draw color 1 on the screen. Now PRESS 2 and draw with color 2. PRESS 3 and do the same. Finally PRESS 0 and draw with color 0. Notice that color 0 seems to erase rather than draw, but I assure you it is drawing. Think about it. Color 0 draws with the background color so when it is plotted any other color, of course, it's changed (I know this was obvious to some of you).

I wanted to show you the playfields. Playfield 0 is the background color; Playfield 1 is whatever is drawn with color 1, etc. Page 53 of your Basic manual shows which Graphics modes allow which playfields. Although very simple, this example is important for you since you must know which playfield you are testing for when you use the collision registers discussed later.

A NOTE ON COLORS -

Throughout all of these examples, if you don't like the colors being used, please change them to match your TV. I have several sets that all show the colors differently, so some of these examples may look strange to you. Here is how to change the colors:

The correct value to POKE is:

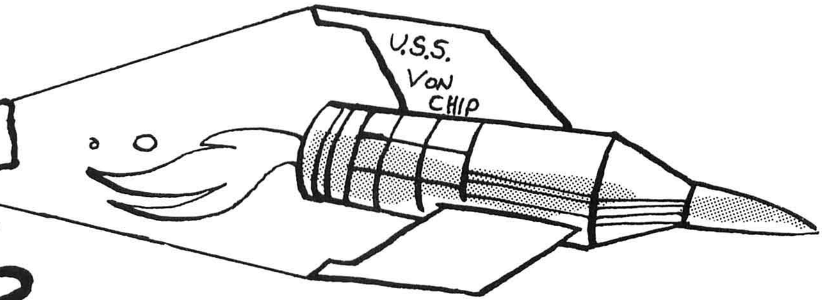
COLOR * 16 + LUMINANCE

where color is a 0 to 15 and luminance is from 0 to 14, in even numbers.

Look within any example you want to change until you find the correct number being POKEd and place your color choice in the POKE statement.

EXAMPLE 5.4

A PLAYFIELD EDITOR



You probably know that any program, whether for a game or a business application, will need a background. The exception is a text-only program which would seldom need the use of PMG.

Let's say you want a space game with mountains, enemy bases, a few ships on the ground to shoot at, and maybe even some trees. These need to be plotted on the screen before the game can begin. The normal method is to use graph paper and hand calculate each point to be plotted. WOW! That is the hard way! Feeling inspired, I thought I would whip up a little editor to help you. Here's how to use it.

NOTE

Because this program requires 32K to run, it is the next to last program on the cassettes. For disk users, it is in the same order we have been discussing.

When the program has been loaded, disk users may press "L" and then "D" to bring in a previously saved drawing as a test. You will notice a flashing dot, or cursor, on the screen. This is the point that is being drawn as you move it. You can choose between playfields 0 , 1, 2 or 3. First, find out how many Playfields you are allowed in the Graphics mode you want to use. For example, if your program uses Graphics 5, you have all four colors to work with; in Graphics 6 you only have two. Press the number of the Playfield color and use the cursor keys(don't press CTRL) to move the cursor. Remember, to erase, you plot in the background, color 0. Draw out a building or spaceship.

When done, the editor allows you save the shape two ways:

To save the shapes for future use and modifications you may press "S" and then "T" or "D" for tape or disk.

When done, the editor allows you save the shape two ways:

To save the shapes for future use and modifications you may press "S" and then "T" or "D" for tape or disk.

Tape users be sure and record the location on the tape where the program starts so that you may find the data again. Disk users will always find the data saved under the same name on the disk, PFDATA, so if you want to keep the shapes use the rename command, "E", FROM DOS, to name the data something more meaningful.

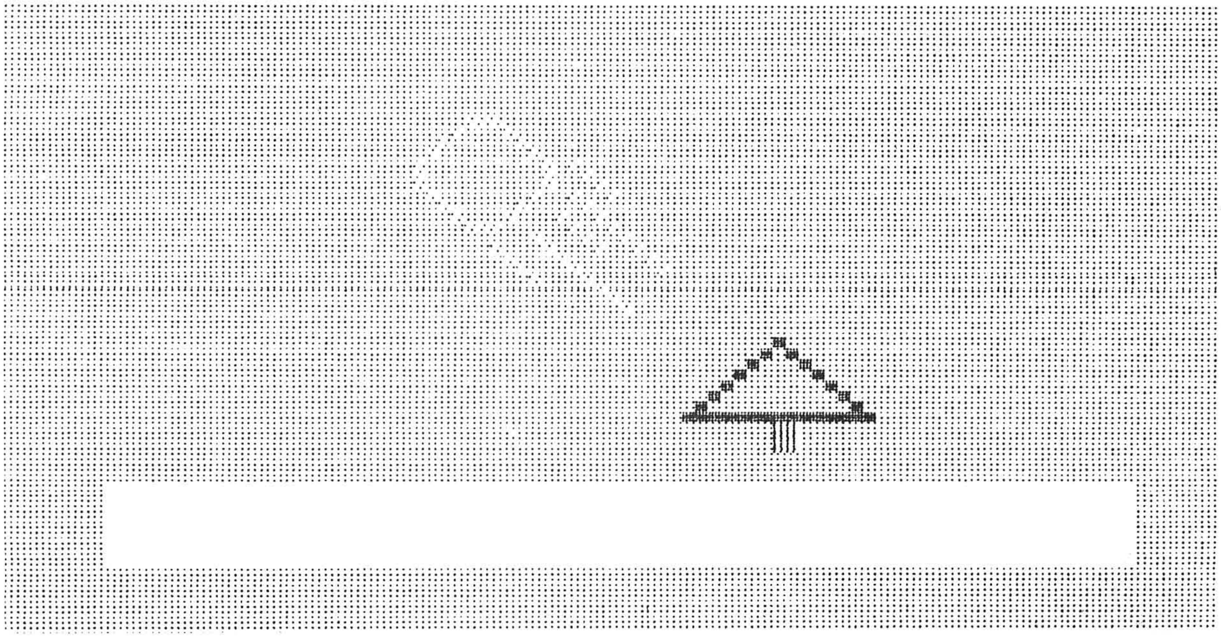
To use the shapes in your program, you will need a simple subroutine. Look in the program code for this example to see a very sophisticated way to handle the data using strings. Go ahead and copy parts of it for your own use.

Before presenting the code, let me remind you what we are doing. Each color of your shape needs to be plotted separately. Also, you will probably want to plot the shape, for example, a tree with some ground around it, at several points in the background. Finally, many other shapes like men, spaceships, and planets, must also be plotted.
To get the data for your shape, just press SELECT and wait for 10 seconds. The text area of the screen will start presenting DATA statements that are the points to plot. The upper left corner of the square that limits your shape, if colored, would be 1,1. These numbers represent first an X value, and then a Y value of a standard grid system the ATARı normally uses for Graphics. To see all of the data for each color, press START each time the data stops. If you have more than 199 points in your shape of any one color, the program will say "TOO MUCH DATA" . I could only fit in that many points, so go back and change a few to a different color. It won't happen for most shapes.

The numbers of the DATA statements should be changed to fit within your program. You may want trees to always start at 5000; buildings at 6000, etc.

Here is a sample program to use with the Playfield Editor.

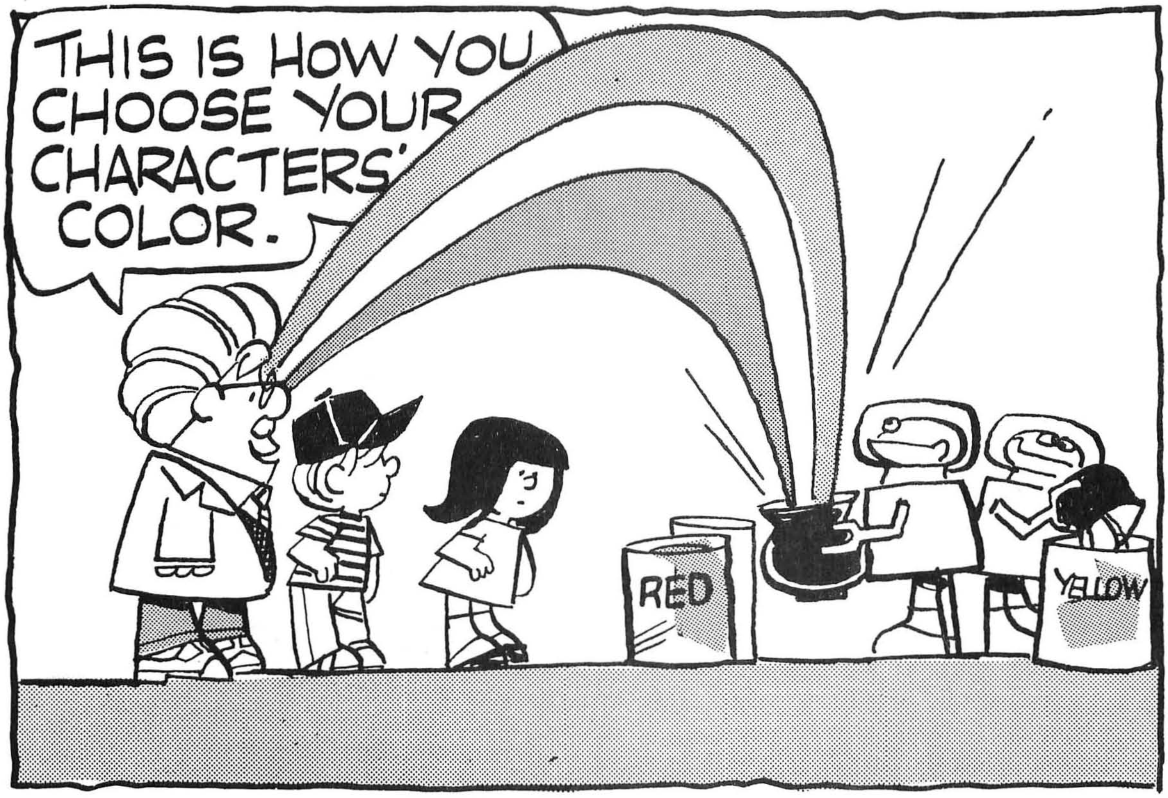
```
 400   GRAPHICS 5
 410   SETCOLOR 0,12,4
 420   SETCOLOR 1,3,2
 430   SETCOLOR 2,13,12
 440   SETCOLOR 3,8,6
 500   REM***DRAW THE TREE
 510   COLOR 1: X=50: Y=24: GOSUB 1000
 600   REM***DRAW THE TRUNK
 610   COLOR 2: X=50: Y=24:GOSUB 1000
 700   REM***DRAW THE SUN AND RAYS
 710   COLOR 3: X=30: Y=15: GOSUB 1000
 800   END
1000   REM***PLOT ROUTINE
1010   FOR I = 1 TO 80
1020   READ A,B: IF A =999 THEN RETURN
1030   PLOT A+X,B+Y: NEXT I
5000   REM***DATA FOR TREE
5010   DATA 2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,8,9,7,9,6,
            9,5,9,4,9,3,9,2,9,1,9,0,9
5020   DATA -1,9,-2,9,-3,9,-4,9,-5,9,-4,8,-3,7,-2,6,-1,
            5,0,4,1,3,999,999
6000   REM***DATA FOR TREE TRUNK
6010   DATA 2,10,3,10,2,11,3,11,2,12,3,12,999,999
7000   REM***DATA FOR SUN AND RAYS
7010   DATA 0,0,1,-1,2,-2,3,-3,4,-4,4,-5,4,-6,3,-7,2,-8,1,-9
7020   DATA 0,-10,-1,-10,-2,-10,-3,-9,-4,-8,-5,-7,-6,-6,
            -5,-6,-4,-5,-3,-4,-2,-3,-1,-2,0,-1,0
7030   DATA 3,1,4,2,5,3,6,4,7,5,8,6,9,7,10,8,5,-1,6,0,7,1,8,2
7040   DATA 6,-3,7,-2,8,-1,9,0,10,1,11,2,12,3,13,4,6,-6,7,-5,
            8,-4,9,-3,0,2,1,3,2,4,3,5,999,999
```



EXAMPLE 5.5
COLORS

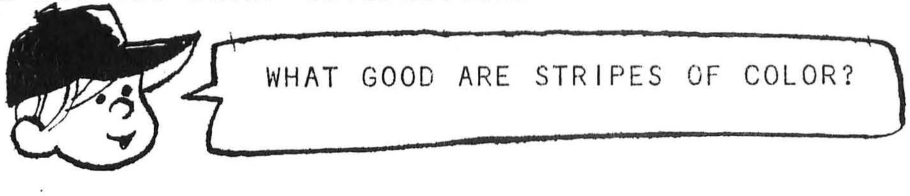
NOTE:

Tape users may have this program cone on the screen looking very strange. Just press reset and type RUN to correct the screen



We talked about how to change the colors of the playfield (708 to 712) a few pages ago. The following utility will allow you to pick the colors directly without calculating the number to POKE. The POKEs listed, however, are for the Player colors which live in different locations (704 to 707 + 711). The color value to place into any of the locations still uses the formula COLOR * 16 + LUMINANCE. Beyond this, the real purpose of this example is to introduce you to the Players and Missiles.

Look at the stripes of color on the screen. Each one is half a Player. The bottom half is "turned off" so that you can see the color information.



Good question! The stripes are only being used to best show the colors of the Players as you change them (I'll tell you how in a moment). You can pick which of the eight pixels (little squares of color) across each stripe will be colored. In the next example we will select certain pixels and create a useful shape. For now the top half of each Player ana Missile is turned on (all 8 pixels colored) and the bottom half is turned off(all eight pixels POKEd with 0's).

Are you wondering which Player is which? Player 0 is on the far left and Player 3 is on the right. On the far right is the corresponding Missiles 0 to 3 that work with each Player. I know that calling the first Player by "#0" is confusing, but it is required by the ATARI hardware and software. Soon you'll see how to place these shapes anywhere on the screen you wish.

How To Use This Example

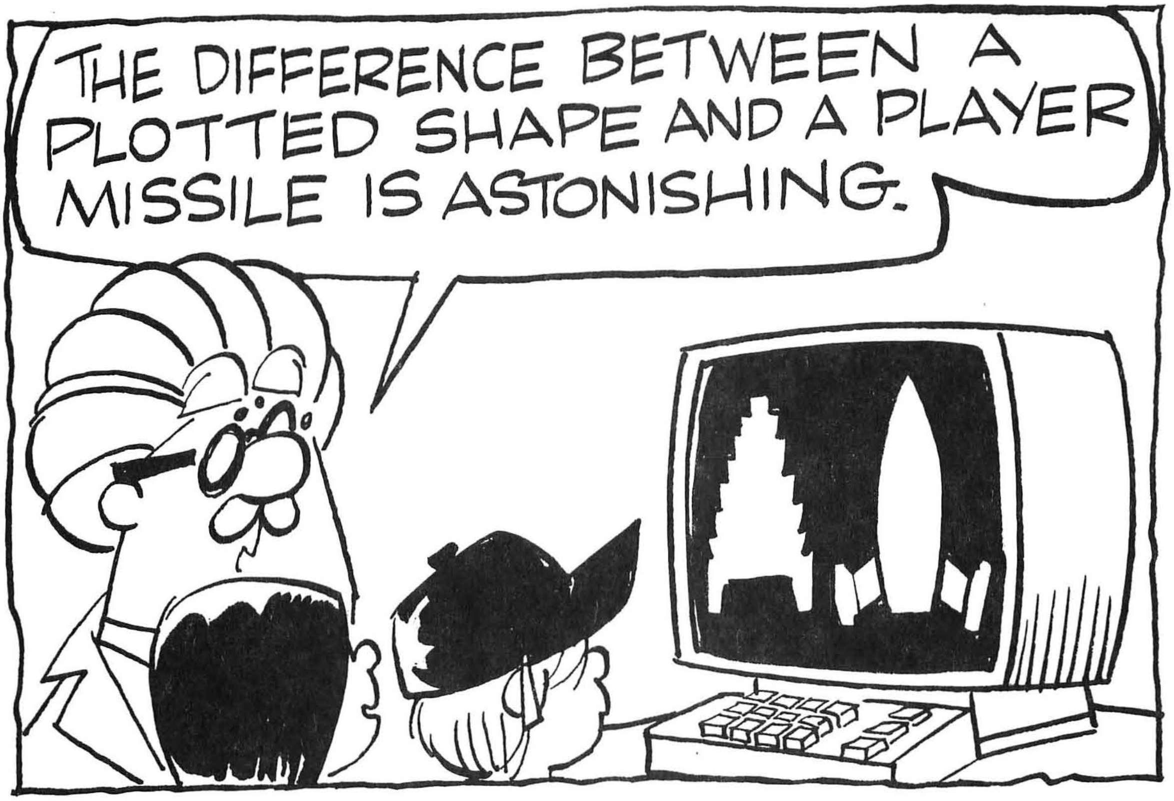
Plug a joystick into port 1. As soon as you move your Joystick an "*" will appear. Place the asterick over any of the Players. Hold the trigger button down and move the joystick up to increase the color value, or down to decrease it. Notice that the numbers you POKE are the same for any given color. This means if you find a red that you like with a value of 45, you can POKE 45 into any of the color registers from 704 to 712 and get red for that Player, Missile, or Playfield.

When you have one Player color just like you want it, move the asterick to another Player and change it's color, too. After all four Players are done, you can set the background color by moving the asterick to the far right side and pressing the trigger. Notice how the background color changes the appearance of the player's colors. Now use these color values in POKE statements within your own programs.

A few of you by now have tried to change the colors of the Missiles. We have not yet mentioned any color registers for the Missiles. That's because there aren't any! Each Missile takes on the color of the Player with the same number. Here is a review of the colors and registers:

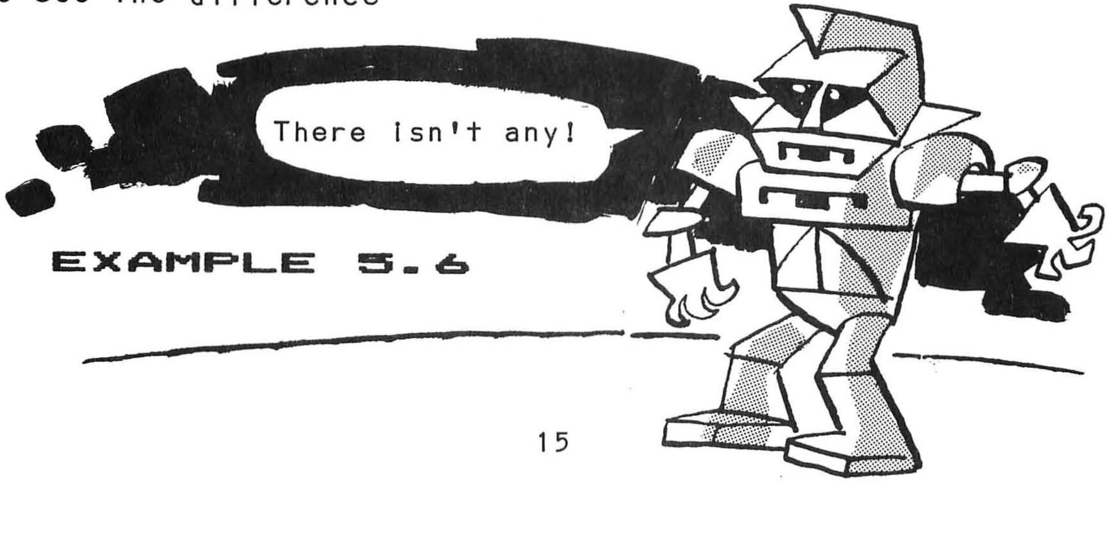
REGISTER	USED FOR
***********	***********
704	Player 0 and Missile 0
705	Player 1 and Missile 1
706	Player 2 and Missile 2
707	Player 3 and Missile 3
711	Player 4 (all Missiles together)

What? Player 4? Yes, you must know by now that the ATARI is full of suprises. It turns out that the Missiles can be combined together into a 5th Player (#4). We will come back to how this is done later. For now, press SELECT. The Missiles will now come together and take on the color POKEd into 711. To use the Color Editor on Player 4, just move the asterick to it and use the trigger. Pressing SELECT again would toggle the 5th Player back to Missiles. This would be a good utility to use whenever you need to choose a color.


THE DIFFERENCE BETWEEN A PLOTTED SHAPE AND A PLAYER MISSILE IS ASTONISHING.

WHICH IS THE PLAYER?

Run the next example, (5.6), and look two of my Robot Prototype's cousins on your screen. Your mission, if you choose to accept it, is to figure out which is a Player and which is plotted, ie., drawn on a Playfield. Really try hard to see the difference



EXAMPLE 5.6

To solve this "Great Mystery" use a joystick in Port 1 to move each shape. The Player moves WITHOUT PRESSING the TRIGGER. The playfield shape moves by PRESSING the TRIGGER.

I have to admit that I cheated in this example. The Player is being moved with a small machine language utility. However, you will soon see that even BASIC can move a Player pretty fast vertically and immediately horizontally.

An interesting experiment to do with this example is to change the color in the program of the Player (find a POKE 704,with a number after it) and then move the Player exactly on top of the plotted shape. If you move two plotted shapes over each other, and then move them away, you will have to redraw them both to remove the holes left in their shapes! The effect is even better if you break the program and change the color of one of the Robots. Aren't Players great?

EXAMPLE 5.7

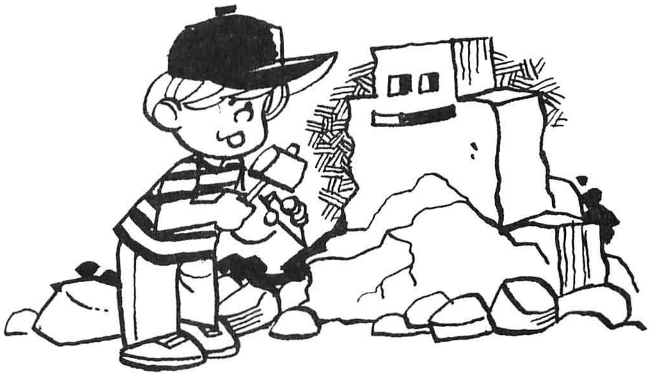
CREATING PLAYERS

Press OPTION to get to the next example, 5.7. This will begin the technical part of this Tutorial.

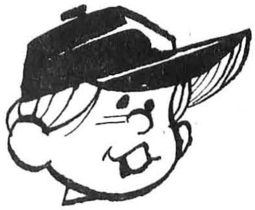
First of all, notice that the last picture is still on the screen. Do you know how this happened? It's very simple. With playfields, when you plot a background on the screen they stay on until erased with either new plotted data, or usually just another Graphics call. You loaded in the next program while a playfield --the Robot-- was still on the screen, so it is still there when this example starts to run. This process is similar to adding 32 to a graphics call as explained in the BASIC manual.

You can create your backgrounds with one program and then have the main loop of your program load in separately. This allows large programs to fit into smaller memory sizes. Players will stay on the screen, too.

Again, plug in Joystick #1. You can move the shape on the screen in all directions, including diagonally. This routine is very important to you since it will teach all of the following basics:



IMPORTANT STUFF TO LEARN FROM THIS EXAMPLE



1) How to create a Player shape

2) How to move that shape up and down (vertically)

3) How to move it sideways (horizontally)

This example can be copied into ANY program you want and just modified in a few places to get a Player moving around on your own design of a game or other application. It should be saved to a separate Tape or Disk and even renamed as a utility for Disk users.

```
10 GOTO 140
20 REM
30 REM ****** MAIN LOOP **********
40 REM
50 ST=STICK(0)
60 X=X+(ST=6)+(ST=7)+(ST=5)-(ST=9)-(ST=10)-(ST=11)
70 Y=Y+(ST=13)+(ST=9)+(ST=5)-(ST=6)-(ST=10)-(ST=14)
80 GOSUB 330
85 IF PEEK(532/9)=3 THEN POKE 764,12:RUN "D:EX5.8"
90 IF PADDLE(0)=228 THEN 110
100 X=PADDLE(0)
110 POKE 53248,X
120 GOTO 50
130 REM
140 REM PMB=PLAYER MISSILE BASE ADDRESS IN PAGES
150 REM (256 BYTES PER PAGE)
160 POKE 704,40
```

The example starts out skipping to line 140 to do some standard setups that will always be required. Line 160 you should recognize as setting the color of Player 0, which we will use for the shape (Spaceship?). Line 170 to 190 look like this:

```
170 PMB=PEEK(106)-16
180 POKE 54279,PMB
190 PMBASE=PMB*256
```

These lines reserve space in memory for Player and Missile shapes to be stored. This topic alone could take pages to explain, but I will give you only enough information to make it work.

Players & Missiles are created on the ATARI by something called DMA. DMA means Direct Memory Access. All the big computers have it. Apples don't! This process simply means that if you have "turned on" the DMA, the computer will take whatever shapes it finds at a place in memory starting with PMBASE and put them on the screen. They will be located horizontally (X direction) by the values stored in their horizontal registers (53248 to 53255). Their size in the X direction is controlled by values stored in size registers (53256 to 53260). Their size in the Y direction is controlled by two factors. First is how many pixels you have turned on for their shape. Second, Players may be created where each number in memory is put on the screen as either one pixel (single line resolution) or two (double line resolution). Their location on the screen in the Y direction is simply controlled by where in the stripe you turn on pixels.

WOW... I'M Tired

That was a lot! There's even more to come, so take a break if you need to......

Back to lines 170 to 190. Line 170 takes the value in location 106, which is the location of the top of user memory (given in number of 256 bytes pages), and subtracts a certain amount of pages. This number is then POKEd into 54279. That will tell the computer where the Player data starts. Finally, line 190 takes PMB in number of pages and converts the value to a regular memory location, PMBASE. We will use this number a lot. The number I subtracted, 16, will change depending on the Graphics mode you are using. Choosing the amount to subtract is explained in EX5.9.

```
210 X=125:Y=100:YSAVE=100
220 POKE 53256,0
```

Next, in line 200, we POKE 53277 with a 3. This memory location is called GRACTL. Poke it with a 1 for Missiles only. Poke with a 3 for Players and Missiles.

In line 210 are the initial X and Y positions where you want the Player on the screen.In 220, the size of Player 0 is set to normal, which is 0. If you had just turned on the machine, 53256 would have contained 0 anyway, but since we are running many programs one after another, this made sure the value was correct. The choices you have for the size are:

Poke With ***********	For *******
0	Normal size
1	Double size
3	Quadruple size

Here are the registers available for controlling Player/Missile size. POKE wth the appropriate values from the previous table.

RESISTER ********	PARAMETER *********
53256	size, Player 0
53257	size, Player 1
53258	size, Player 2
53259	size, Player 3
53260	size of all missles

To determine Missile size, you'll have to do a little calculating, but nothing too complicated. Just pick the desired size for each Missile from the chart below, then add up the numbers for all four (or however many you're using) and POKE the total into memory location 53260.

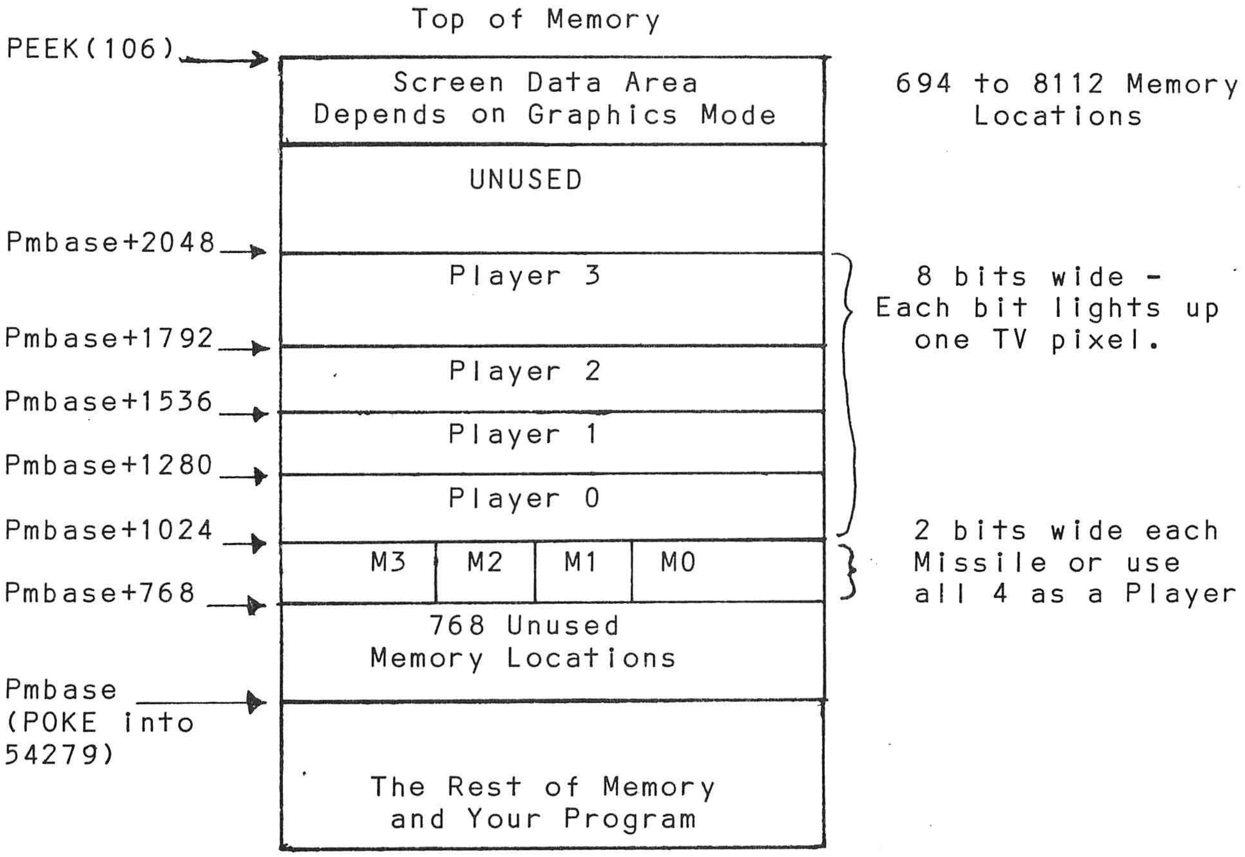
For example, if you want a normal Missile 0, a double Missile 1, a quadruple Missile 2, and a double Missile 3, you would POKE 0+4+48+64=116.

MISSILE # ********	NORMAL ******	DOUBLE ******	QUAD ****
0	0	1	3
1	0	4	12
2	0	16	48
3	0	64	192

```
230 FOR I=PMBASE+1024 TO PMBASE+1280:POKE I,0:NEXT I
```

Line 230 POKEs 0's into the area where the data for the Player must go. This will act to clear out the Players stripe first. This may not seem necessary to you. For this reason, some of the later examples will actually erase their Player areas while you are watching. You will see the junk being erased. This statement takes a few seconds to do, and thus delays the beginning of any program that uses it. The best way to speed it up is to put it at the very front of your programs. This is demonstrated in later examples. For this example we only had to clear out memory between Pmbase+1024 and Pmbase+1280. Why? To explain we need a chart of what memory looks like:

SINGLE LINE RESOLUTION



Look at the chart above marked Single Line Resolution. Starting at the place marked top of memory, we move down by a certain number of pages subtracted from the value in 106. This places us at the BOTTOM of the chart. This calculation was done in lines 170 to 190. Again, the amount to subtract will be explained in EX5.9.

Now that we have placed Pmbase, go back up in memory to Pmbase+1024. The next 256 bytes of memory are the stripe that holds Player 0. If any of these bytes of memory are not 0, and if the setup POKEs have been made to turn on DMA, then those non 0 bytes will light up on the screen with the color values stored in 704.

To move the shape up or down the screen, you just POKE non 0 bytes up or down the stripe of memory. Let's say that you place Pmbase at 30000 in your memory. Any non 0 bytes at Pmbase+1030(30000+1030, or 31030) will appear near the top of the screen and if those same numbers are POKEd into Pmbase+ 1200, they will appear near the bottom of the screen.

Continuing with this logic should make it clear what happens if you POKE numbers into Pmbase+ 1400. You would now be turning on a shape in Player 1 that would appear on the screen if the "setup POKEs" were done. Let's finish discussing the setup procedure.

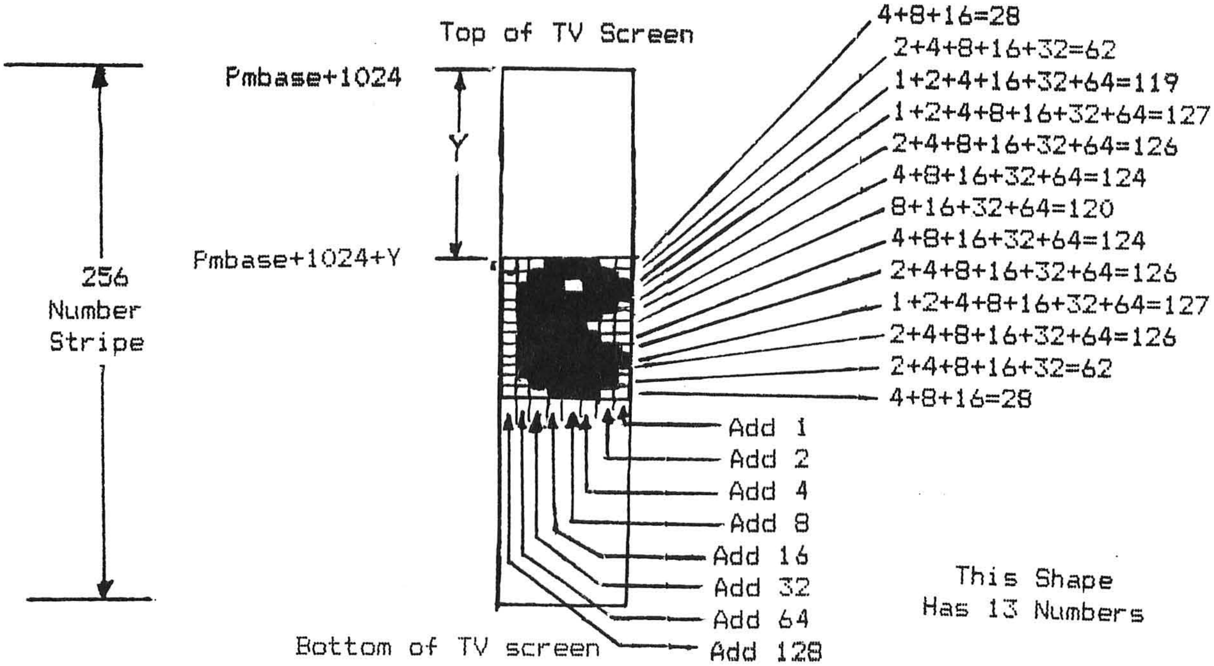
Lines 240 to 300 POKE the shape into memory, just as we were talking about above. Understanding lines 250 to 300 will be a real key for you. Here is what they look like:

```
240 RESTORE 210:CNT=0
250 FOR I=PMBASE+1024+Y TO PMBASE+1280
260 READ B:IF B=0 THEN 310
270 CNT=CNT+1
280 POKE I,B
290 NEXT I
300 DATA 8,60,126,195.126,60,24,126,165,0,0
```

The shape is poked into memory with an offset of "Y". This says to place the shape Y units down the screen. If Y=0 then the shape will be at the top of the screen. A value of Y=252 puts the shape at the very bottom since the shape has eight numbers that must fit into the 256 byte long stripe of memory. In a few lines we will simply calculate where we want the shape on the screen and use this routine to POKE it into the right place.

The last major hurdle is the way the data is calculated. Here is another figure to help you visualize the shape:

PLAYER 0



To create your own shape, you just fill in the boxes you want for the shape. Then to get the "TOTAL", you just go across each row and add up the value for any box that is filled in. When you poke these numbers into memory, your shape appears on the screen!

Back to the program code. Line 310 is what finally turns on the player. The correct value here is obtained by adding up the options you want from the following:

	FOR	ADD	
DMACTL (559)	Wide playfield	3	
	Standard Playfield	2	Choose 1 Only
	Narrow Playfield	1	
	Turn on Missile DMA	4	
	Turn on Player DMA	8	
	Double Line Resolution	0	
	Single Line Resolution	16	Choose 1 Only
	Turn on Main DMA	32	

In this case we want Standard playfield (2) + Missile (4) and Player (8) DMA plus the Main DMA (32) and Single line Resolution (16) for a total of 2+4+8+32+16=62. This is why we poke 559,62.

That ends the Setup

You should be able to use this same setup code for most PMG programs you do. Just make the few changes required by things like resolution and playfield size. Note - single and double line resolution will be discussed in example 5.10 coming up soon.

The program now branches to the main program loop at line 50, which reads joystick 0. Lines 60 and 70 increment X & Y from their initial positions set in line 210. The only thing left for the program to do is move in the X and Y directions. The X direction is easy. The ATARI has built in position registers for the X direction. In line 110 we just poke the new X value into that memory location, 53248.

Here are the X direction registers

REGISTER #	MISSILE/PLAYER
**********	**************
53248	P0
53249	P1
53250	P2
53251	P3
53252	M0
53253	M1
53254	M2
53255	M3

To move in the Y direction we have to do something similar to the way the playfield Robot was moved in EX5.6. Fortunately, we don't have to move as many bytes of memory around. That robot had almost 200 data points! Moving our player is as simple as poking in eight numbers into memory.

```
310 POKE 559,62
320 GOTO 50
330 REM
340 REM ***** UP/DOWN MOVE *******
350 IF YSAVE=Y THEN RETURN
360 IF YSAVE<Y THEN 430
370 RESTORE 210
380 FOR I=1 TO CNT
390 READ B
400 POKE PMBASE+1024+Y+I,B
410 NEXT I
420 YSAVE=Y:POKE PMBASE+1024+Y+CNT+1,0:RETURN
430 RESTORE 210
```

```
440 FOR I=1 TO CNT
450 READ B
460 POKE PMBASE+1024+Y+I-1,B
470 NEXT I
480 YSAVE=Y:POKE PMBASE+1024+Y-1,0:RETURN
```

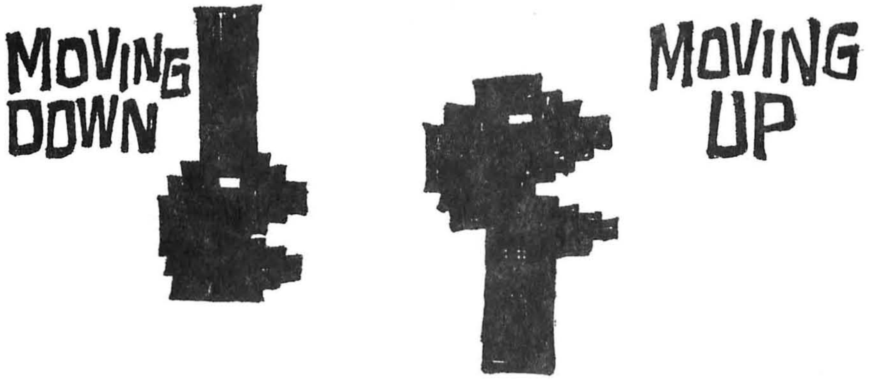
Look at lines 340 to 480. First, in 350, I compare the current value of Y to the last. If it is the same, we don't waste our time and just return. Line 360 says, "If Y is now bigger than last time, go to the move up in memory routine at 430, otherwise move down". Remember, as Y increases in memory, the shape appears to go down the screen. This comes from the way the ATARI does it's coordinate system and is explained on page 47 of your Basic manual.

Both the routine to move up, and the one to move down, are similar to the original one we used to POKE the shape into memory. All we are doing is POKEing the same shape down or up by one byte in memory.

NOW WE ARE GETTING HOT!

IMPORTANT POINT!

When you move the shape you repoke 7 of the 8 numbers that are currently in memory and poke in one number in a new location. That last number is not changed because you have moved by one byte. That is why the 0 is poked into that remaining location in line 420 or 480. Were the 0 omitted, a trail of unchanged bytes would be left on the screen, leaving something like this:



If you are lost by all this there is only one way to figure it out. Stop the lesson now, break the program, and start making small changes to where and what numbers are poked into memory. Soon you will say to yourself, "That's what The (old/not so old) PROFESSOR was trying to say!

THAT'S IT?

You now have most ot the basics to place a player on the screen and move it around. The earlier playfield editor discussed how to create the background for your program. Now all we need are the small details that tie all of this together.

HEY! WHAT ABOUT THE PADDLES?

Oh yes, I forgot to tell you why the move in the X direction was so slow. The joysticks we are using, combined with the program code, only allow X to change by one value at a time. Paddles, on the other hand, can instantly change the value they return from 0 to 228. Plug in a paddle controller, if you have one, and now try moving the shape across the screen. It will be instantaneous!

EXAMPLE 5.8

This simple example takes the same basic routine as in 5.7, but starts to make it look like a game. Finish it if you'd like, it's good practice. There are three things you should learn from this example.

First, notice the stripe of randomly flickering "junk" on the screen when the program first starts. This is caused by only partly turning on the DMA as discussed above. As soon as all the setup is complete, the exact shape appears.

Second, look at the program list and see the way the program is structured. When you write your own codes, I suggest you do something like this. The program just goes to several subroutines that :

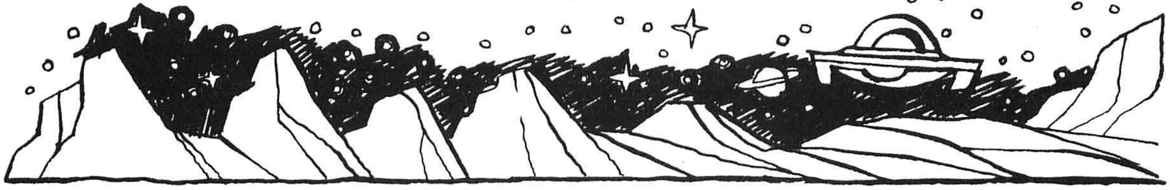
1) Draw the mountains
2) Draw the stars
3) Set up Player 0 as shown in the utility above
4) Main loop routine for the movement and/or scoring

Notice that I use sound in the loop. It's easy to do if you spend a little time in practice. This simple structuring is used in many arcade-style games.

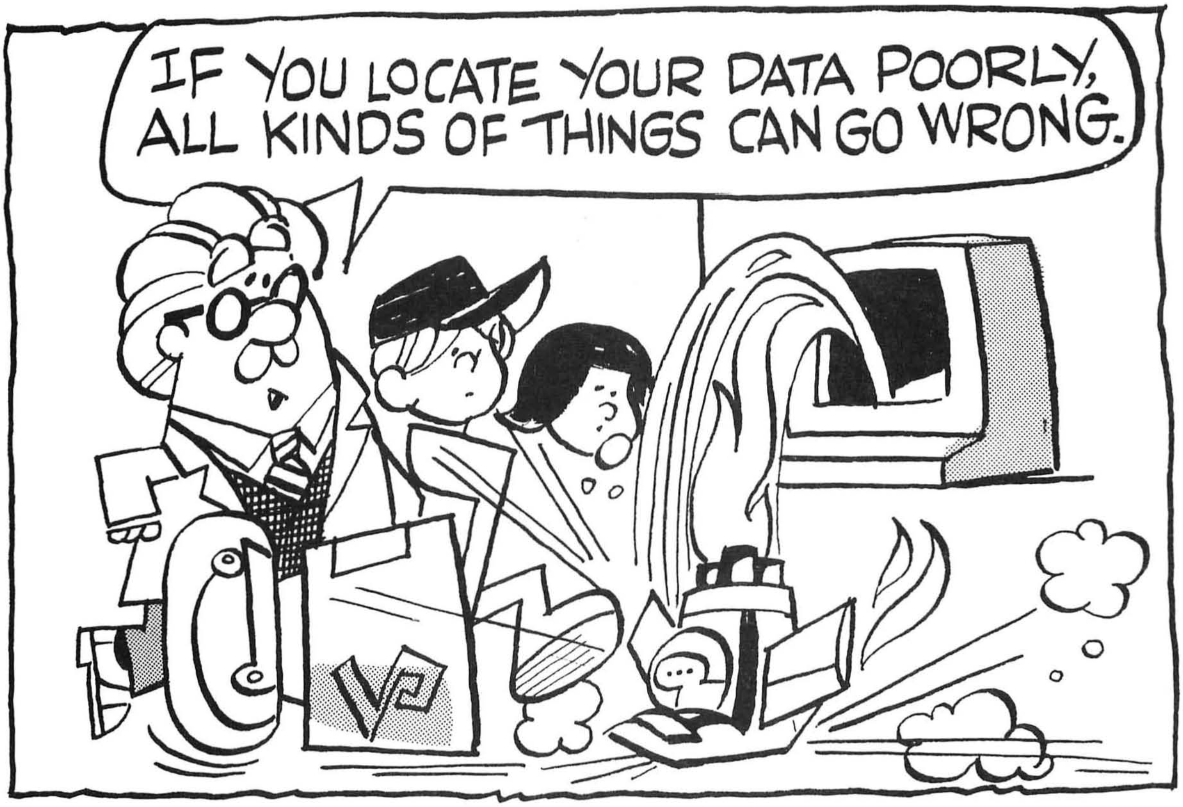
Third, I'll show you one of the two machine language utilities within the program. Both are short and can easily be transferred to you own programs. This one is used to move only Player 0. It will read Joystick 0 and move the data in the Player 0 stripe accordingly. You must use single line resolution Players, as we have been doing so far. (That means the shape is at PMBASE +1024 to PMBASE +1279) Copy the needed lines to a separate file using the LIST command. Then you could enter it as needed. Don't forget to delete the other lines of the program first. The lines you need to keep are just 40,90, and 130.

Heres the routine:

```
40 DIM E$(60)
90 E$(1,60)="hhhJH[illegible]
H[illegible]"
160 A=USR(ADR(E$),STICK(0))
```



EXAMPLE 5.9
A GOOF-UP!


IF YOU LOCATE YOUR DATA POORLY, ALL KINDS OF THINGS CAN GO WRONG.

When you run this example notice our old friend the Robot. He's back! However, as you move him around (using the same machine language routine as the last example), there's some 'junk' on the screen.

The time has come, my friend, to explain how to position the Player/Missile data area within memory. Look at the program listing for example 5.9. Do you remember how we obtained the value for PMBASE? It worked like this:

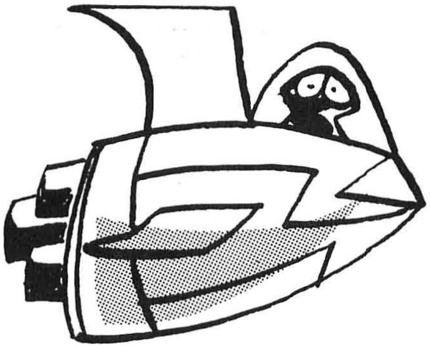
```
100  PMBASE = (PEEK(106)-16)*256
```

On page 45 of the BASIC manual, you'll see a chart showing the amount of RAM (memory) needed for each graphics mode. As the next figure shows, you don't want the data for the Players and Missiles overlapping the data placed on the screen. Even worse would be to write over the DISPLAY LIST. That is what happened to tape users with example 5.5.

When they reran the program by pressing RESET, the Operating System fixed the display list. In this example, look at the Player data being moved by the machine language program within the Player stripe. See if you can understand the term 'wraparound' by moving the Player up until it comes back from the bottom. This will also work sideways. The data that is being written to the screen area will also 'wraparound'.

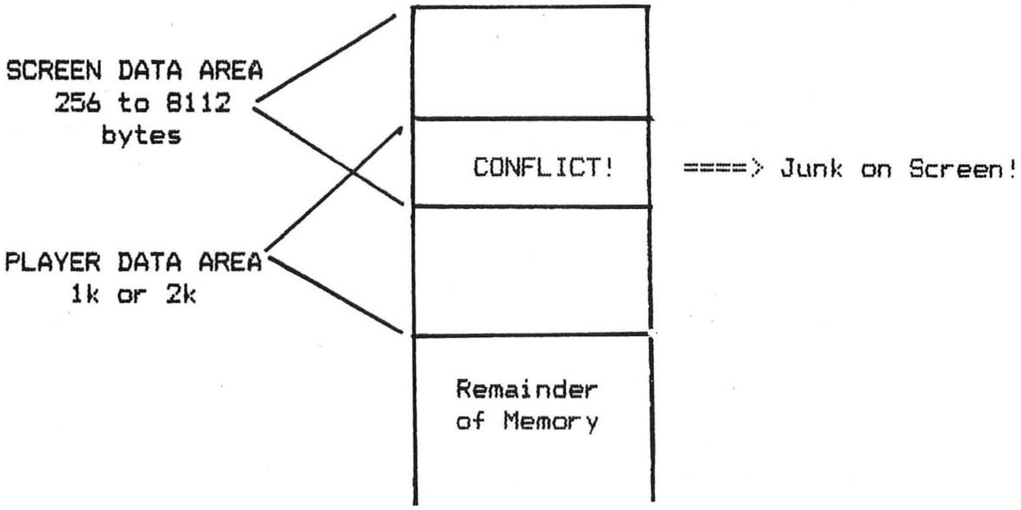
In example 5.5, when I explained the method, I said that the number 16 changes depending on your program. Let's discuss why. The listing you are looking at will only use 16 as the memory amount to subtract because the Graphics mode being used requires 1K or less of memory. As another example, 5.8 POKEs memory back 40 pages because Graphics 8 requires more memory.

Here's a drawing to help our discussion:



OOP'S WRONG DRAWING ....

CONFLICTS
Between
PLAYER MISSILE DATA AREA
and
SCREEN DATA AREA??



The easy way to avoid putting Player data where it doesn't belong is to use the following chart. Single line resolution players always must start on a "1K boundary." This simply means subtract from the value in 106 either 8 pages, or 16, 24, 32, etc. Remember, a PAGE is 256 bytes (1/4K). When we discuss double line resolution players (soon, soon), you must start the data on a 1K boundary, so subtract 4, 8, 12, 16, etc.

GRAPHICS MODE	APPROX. # MEMORY LOCATIONS USED	SUBTRACT THIS # FROM LOCATION 106
0	992	16 (8)
1	674	16 (8)
2	424	16 (8)
3	434	16 (8)
4	694	16 (8)
5	1174	16 (8)
6	2174	16 (12)
7	4190	24 (20)
8	8112	40 (36)
9	8112	40 (36)
10	8112	40 (36)
11	8112	40 (36)

IT'S A BIRD
IT'S A PLANE
IT'S...DOUBLE LINE RESOLUTION

EXAMPLE 5.10

This simple example introduces you to the SQUIGILY MONSTER. It will show itself in two sizes. The smaller size is called single line resolution. This is the way we have previously set up our Players. Simply put, each number in the data area for the Player is placed on the screen as one line also. The data area has 256 bytes available, which as you saw in the Color Changing example, more than covers the screen. If you look closely at the screen you can see the individual pixels.

The other way you can set up your Players and Missiles is called double line resolution. This uses only 128 bytes per Player; meaning, to fill the screen, the computer will place TWO lines on the screen for each number in its data area. This makes the shape appear thicker.

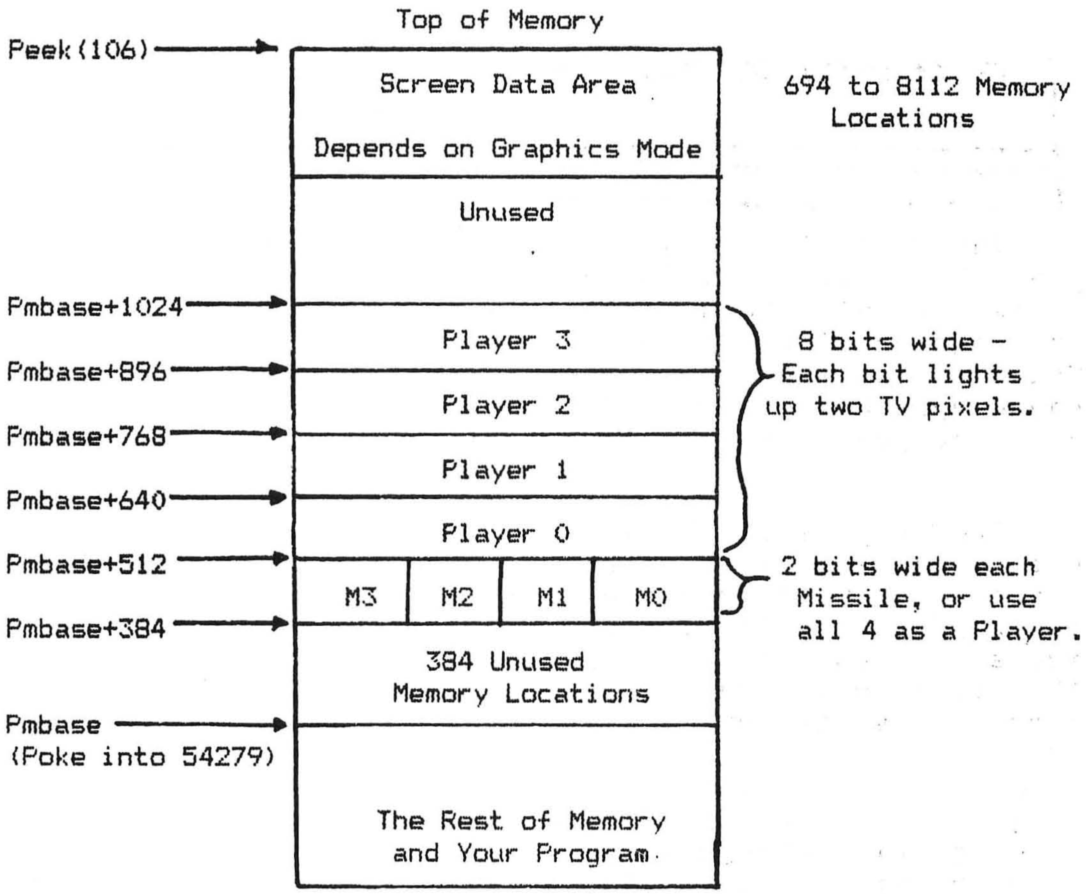
Press the SELECT button for the single line Player and the START button for double line.

Double line Players are set up similar to single line, but with these differences:

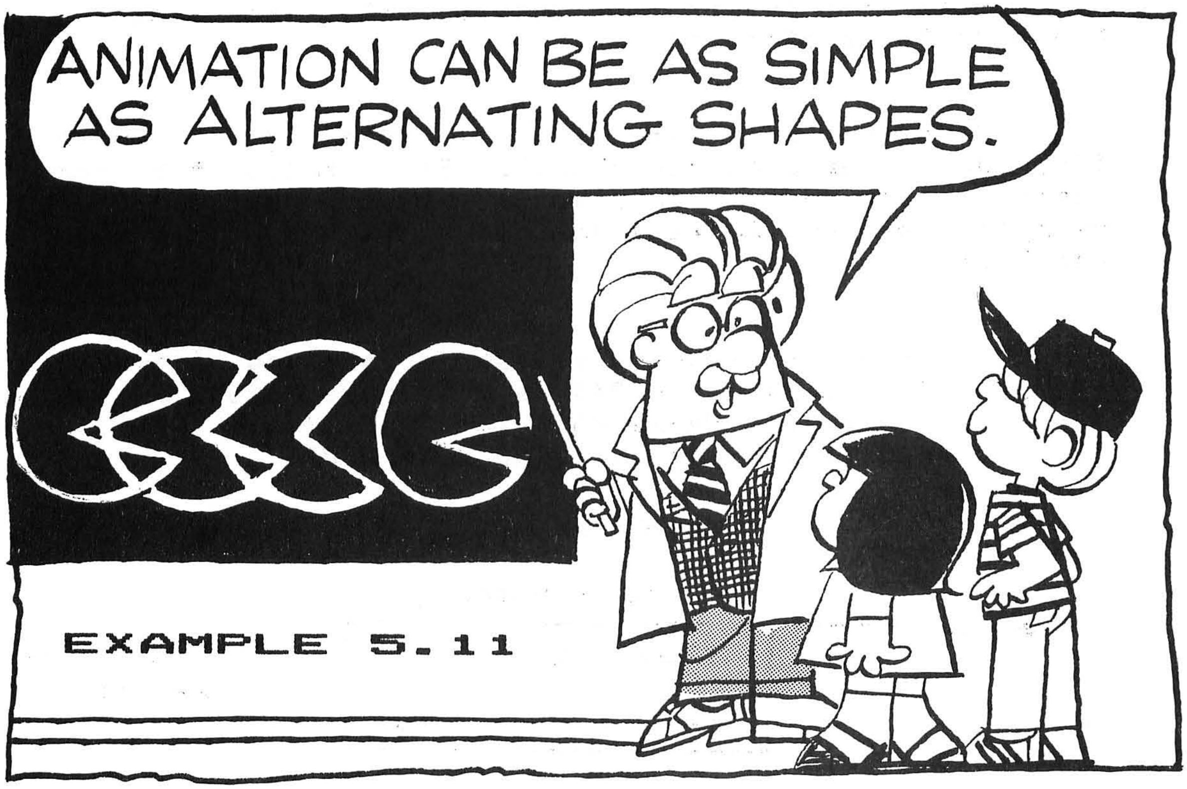
1) Instead of adding 16 to the total value to POKE into 559, add 0 (see previous chart). This tells the computer where to look for the data for each Player and Missile. This example switches between the two resolutions with just this one POKE.

2) Instead of POKEing the shape of the Players into PMBASE+768 to PMBASE +2048 as the above figure showed, you place the data into PMBASE+384 to PMBASE+1024. This means that we have to look at .... ANOTHER FIGURE:

DOUBLE LINE RESOLUTION



The decision on which resolution to use depends on how small you want the pixels to be in your figure and how much memory you can afford to use for the shape data. I have given you several of each type in these examples, so it's up to you. Have fun!



There isn't too much more to learn before we can discuss the game. Next, we'll animate the little Pacman(tm) type shape (note: we refer to a general type of shape and are not trying to infringe on others rights, end of legal talk). Run example 5.11. You'll see the memory area for the shape cleared out, then the shape appears. It doesn't look much like a Pacman(tm)? No mouth? Push the joystick!

The shape looks better now that it is moving. I could have used three or four shapes to open and close the mouth, but two shapes is simple to explain. If you add another shape with the mouth just slightly open, you'll see smoother action. To see the two shapes as they change, just give a series of quick pushes on the stick. What you are seeing is two Players, but only one at a time. Let's see how this is done.

```
10 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 240
20 FOR I=PMBASE+1024 TO PMBASE+1792:POKE I,0:NEXT I:RETURN
30 REM *****PLAYER SETUP *****
40 GRAPHICS 7:D=20:E=50:POKE 710,0:POKE 712,0:POKE 704,41:
   POKE 706,41:POKE 559,62
45 COLOR 2:PLOT 1,57:DRAWTO 159,57:PLOT 1,65:DRAWTO 159,65
50 A=PEEK(106)-32:POKE 54279,A:POKE 204,A+4:POKE 203,0:
   PMBASE=A*256
60 POKE 53277,3:X=150:POKE 205,120:POKE 53256,0:POKE 53250,150:
   POKE 53258,0
70 GOSUB 20
80 RESTORE 110:Y=150
90 FOR I=PMBASE+1024+Y TO PMBASE+1209:READ B:IF B<>0 THEN
   POKE I,B:NEXT I
```

```
100 REM ***** DATA FOR PACMAN1 *****
110 DATA 28,62,119,127,254,252,248,252,254,127,126,62,28,0,0
120 RESTORE 150
130 FOR I=PMBASE+1536+Y TO PMBASE+1792:READ B:IF B<>0 THEN
    POKE I,B:NEXT I
140 REM ***** DATA FOR PACMAN2 ******
150 DATA 28,62,118,255,255,255,255,255,126,126,62,28,0,0,0,0
160 POKE 559,62
170 REM
180 REM ******MAIN LOOP ******
190 REM
210 ST=STICK(0):S=PEEK(53279)
220 IF S=3 THEN POKE 764,12:RUN "D:EX5.12"
222 IF ST=15 THEN 210
225 X=X+(ST=7)-(ST=11):POKE 53248,X:POKE 53250,0:FOR J=1 TO 50:
    NEXT J
226 POKE 53250,X:POKE 53248,0
230 GOTO 210
240 GRAPHICS 17:POKE 712,69:POKE 710,19
250 ? #6;"ANIMATION OF SHAPES "
260 ? #6;"  IS AS SIMPLE AS   "
270 ? #6;"SWITCHING BETWEEN   "
280 ? #6;" TWO PLAYERS AS YOU "
290 ? #6;"MOVE AROUND.....use "
300 ? #6;"YOUR JOYSTICK TO SEE"
310 ? #6;"THE pacman(TM) MOVE!"
320 FOR I=1 TO 3000:NEXT I:GOTO 40
330 POKE 764,255
```

Look at the program code for example 5.11. Notice in line 50 the memory setback is 32, not 16. This is the amount required when using Graphics mode 7. In line 60 POKEs, are made to both the horizontal location and size of Player 2. Yes, Player 2! You don't have to use the Players in any particular order. Lines 120 to 150 just POKE in the data for a second shape into the correct area for Player 2. In line 225, POKEs are made to place Player 0 at X and Player 2 at 0, which is off screen. Finally, at line 226 Player 0 is moved off screen and Player 2 is placed at X. There is a delay on line 225 to allow you to see the "switching" of shapes.

Does that seem hard? All you have to do to create additional Players is:

1) Poke into the correct memory area the data for their shape

2) Poke the size you want into the size location in memory (optional,since 0 is a default if you forget)

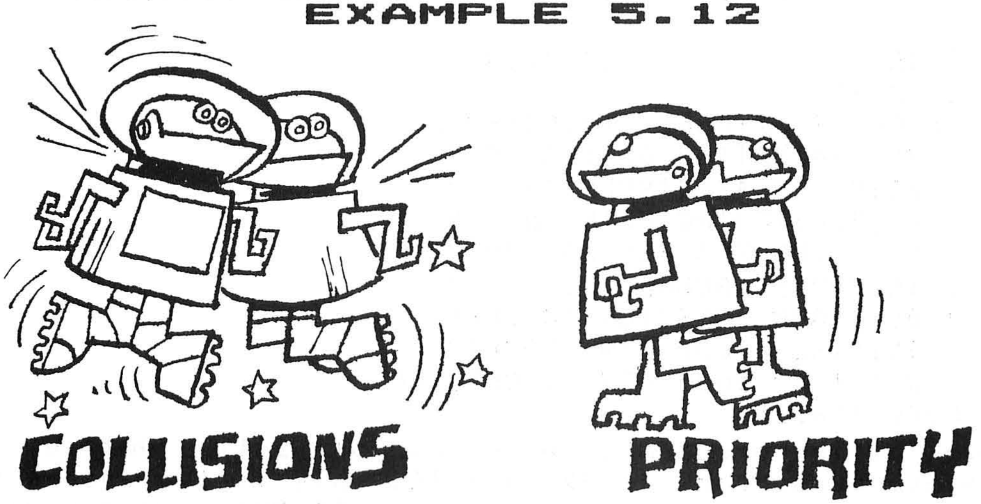
3) Poke the horizontal location (X) into the horizontal memory location.

4) Have fun!

To animate a shape, just alternate two or more shapes at the same spot on the screen. The different shapes will seem to blend into a single, animated shape!

A note on animation is in order. Remember the Robot example where I asked "Which is the Player?". I hope you can now see the power of using Players for your shapes. To plot and erase this shape from BASIC would be very,very slow!

PRIORITY AND COLLISIONS
EXAMPLE 5.12



These two topics were saved for last. They're not difficult subjects, I just wanted you to get a lot of experience first with the basics. Example 5.12 takes the beginnings of a game that we did in example 5.11, and adds collision with some energy pellets in order to allow the creature to "eat" the pellets. When it crosses over the "walls" in the tunnel, the shape will seem to be in front of the walls. At the end of the tunnel, the shape will be inside, or behind the wall because I set the priorities so some colors act differently than others. Here's how I did it!

Most of this program is the same as the other examples. One of the nicest things about using Players and Missiles is writing a program, and then making only slight changes to use it for a completely different purpose.

Line 10 POKEs all of the horizontal location registers to 0. This is not used in this program, but makes sure that all previous Players are off screen. Remember, just because we loaded in a new program, doesn't mean that old Players will disappear. They stay there until removed! This can be useful in programs that have multiple parts to them.

```
10 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 270
20 FOR I=PMBASE+1024 TO PMBASE+1536:POKE I,0:NEXT I:RETURN
30 REM *****PLAYER SETUP *****
40 GRAPHICS 7:POKE 710,0:POKE 712,69:POKE 704,41:POKE 706,41:
   POKE 559,62
45 POKE 708,103:POKE 752,1:POKE 709,139:POKE 710,100
47 COLOR 1:FOR X=20 TO 150 STEP 30:PLOT X,61:PLOT X+1,61:NEXT X
50 COLOR 2:PLOT 1,57:DRAWTO 159,57:DRAWTO 159,66:DRAWTO 1,66:
   DRAWTO 1,57
52 PLOT 157,57:DRAWTO 157,66:PLOT 158,57:DRAWTO 158,66
55 COLOR 3:FOR J=1 TO 4:FOR K=1 TO 5
56 T=K+J*30
57 PLOT T,57:DRAWTO T,66:NEXT K:NEXT J
```

Lines 47 to 57 plot three different parts to the background. The first part is the "energy pellets",ie. small dots, that the Pacman(tm) will "eat". These are plotted in Color 1 (location 708). Next, lines 50 & 52 draw out the rectangular lines that our imagination will call walls of a Pacman(tm) tunnel. The walls are drawn in Color 2. Lines 55 to 57 draw the walls that impede the character's progress. These walls are plotted in... You guessed it.... Color 3. The fact that different things are drawn in different colors is critical to the game. These locations, called COLLISION REGISTERS, are checked by the ATARI to see if something (like a Player) has touched something else (like a Color 1 wall). Notice that Colors 1 to 3 may all be the same, i.e., red, but the different walls must be drawn with different COLOR statements for collisions to work.

```
60 A=PEEK(106)-32:POKE 54279,A:POKE 204,A+4:POKE 203,0:PMBASE=A*256
70 POKE 53277,3:X=50:POKE 205,120:POKE 53256,0:POKE 53250,150
80 GOSUB 20
90 RESTORE 120:Y=150:ERAX=-10
100 FOR I=PMBASE+1024+Y TO PMBASE+1280:READ B:IF B=999 THEN 110
105 POKE I,B:NEXT I
110 REM ***** DATA FOR PACMAN1 *****
120 DATA 28,62,119,127,254,252,248,252,254,127,126,62,28,999
130 RESTORE 160
140 FOR I=PMBASL+1536+Y TO PMBASE+1792:READ B:IF B=999 THEN 150
145 POKE I,B:NEXT I
150 REM ***** DATA FOR PACMAN2 ******
160 DATA 28,62,118,255,255,255,255,255,126,126,62,28,999
17u POKE 559,62
180 REM
19u REM ******MAIN LOOP ******
200 REM
205 POKE 532/8,1:REM CLEAR ALL COLLISION LOCATIONS(SO THEY HOLD 0'S)
220 ST=STICK(0):S=PEEK(53279)
230 IF S=3 THEN POKE 764,12:RUN "D:EX5.13"
240 X=X+(ST=7)-(ST=11):POKE 53248,X:POKE 53250,0:FOR J=1 TO 50:
    NEXT J
242 IF PEEK(53252)<>0 THEN GOSUB 1000:GOTO 220
245 IF X>220 THEN X=40
250 POKE 53250,X:POKE 53248,0
260 GOTO 220
```

In line 205, we learn a new trick. To go with those Collision Registers, there is this location, called HITCLR, at 53278. It does just what it's name implies... When you POKE it with a 1, it will clear the other Collision locations so that they read 0. If this isn't done, then after a collision occurred, the appropiate collision register would be set and stay that way. You'll see all this in action in a moment. Here's a list of all the collision registers:

Memory Location	Type of collision
*******************	*************
53248	Missile 0 to Playfield
53249	Missile 1 to Playfield
53250	Missile 2 to Playfield
53251	Missile 3 to Playfield
53252	Player 0 to Playfield
53253	Player 1 to Playfield
53254	Player 2 to Playfield
53255	Player 3 to Playfield
53256	Missile 0 to Player
53257	Missile 1 to Player
53258	Missile 2 to Player
53259	Missile 3 to Player
53260	Player 0 to Player
53261	Player 1 to Player
53262	Player 2 to Player
53263	Player 3 to Player
53278	Clear all collision registers

That's a lot! They are really easy to use, however, so stay with me. You simply PEEK at the location that starts with the object you are concerned with. In this example, we want to know if the shape, Player 0, has collided with the Playfield. That means we will look at the value in 53252. This is done in line 242. If it isn't 0 then some kind of a collision has occurred and we branch to the subroutine at line 1000.

```
270 GRAPHICS 17:POKE 712,245
280 ? #6;"NOW WE ADD CHECKING "
290 ? #6;"FOR COLLISIONS WITH "
300 ? #6;" THE WALLS AND ALSO "
310 ? #6;" PRIORITY WITH THE  "
320 ? #6;"OTHER SHAPES....use "
330 ? #6;"YOUR JOYSTICK TO SEE"
340 ? #6;"THE pacman(TM) MOVE "
350 ? #6;"  IN AND OUT OF THE "
360 ? #6;"   TUNNEL SHAPES    "
365 FOR I=1 TO 2000:NEXT I
370 GOTO 30
```

```
1000 REM **** COLLISION SOUND ****
1002 COL=PEEK(53252)
1004 IF COL=2 THEN 1015
1005 IF COL<>1 THEN 1030
1006 ? "YUMMY!!":? :?
1007 FOR I=200 TO 0 STEP -1:SOUND 0,I,10,8:NEXT I:? :? :?
1008 COLOR 0:ERAX=ERAX+30:PLOT ERAX,61:PLOT ERAX+1,61
1009 X=X+8:POKE 53248,X:POKE 53250,X
1010 GOTO 1030
1015 SOUND 0,50,8,8
1020 FOR I=1 TO 30:NEXT I:SOUND 0,0,0,0
1030 POKE 53278,1:RETURN
```

From 1000 to 1005, we test to see what is in the Collision register. We already know it isn't a zero -- that's how we got this far. If it's a 2 then we know the Player "hit" (overlapped) Playfield 2, which is the walls of the tunnel. We then make a collision sound and go to line 1030 which is ALWAYS REQUIRED to reset all collisions registers to hold 0's. If it is a 1, we ran into Playfield 1, which in our program is the "energy pellets", so the message "Yummy" is printed and an appropiate sound is made. These lines also POKE HITCLR (53278) and return.

Does all that make sense to you?

Let me repeat the basic way to use collisions: Find the Collision register that deals with the two items you are concerned with, either a Missile #_ or a Player #_ on the left side and a Player, Missile, or Playfield number on the right. Read the value in that register by using the PEEK command. The number that results will be either 0, 1, 2, or 3. If it is 0, it means you collided with yourself which is meaningless. If it is a 1, you collided with Playfield 1, Player 1 , or Missile 1 depending on the register you are using. If a 2, it means you collided with PF 2, PLAYER 2, or Missile 2. A 3 means a collision with PF3, Player 3, or Missile 3. You can then branch to do whatever kind of noise or graphics you want.

I have one last trick in this example. Line 245 allows the character to wraparound in the X direction. It just tests for a value of X greater than 200, which is off the right side of the screen. By setting X equal to 40, the character will suddenly appear on the left!

One part of this example doesn't show up in the program listing. That's priority, meaning "What will appear in front of what". In other words, will a character appear to go inside a House drawn on the screen, or pass in front? This example didn't POKE anything into the Priority register, so the defaults were used. Here are the options for the priority register:

PRIOR (623)

	ADD
For overlapping areas of Players to have a third color	32
For all 4 Missiles to have the color in location 711(Playfield 3)	16

ADD ONLY ONE FROM THE FOLLOWING LIST

PF0, PF1, P0 to P3, PF2, PF3, BACKGROUND	8
PF0 to PF3, P0 to P3, BACKGROUND	4
P0 & P1, PF0 to PF3, P2 & P3, BACKGROUND	2
P0 to P3, PF0 to PF3, BACKGROUND	1

To use this location, just add up the options and POKE the TOTAL NUMBER into 623. You can only choose one set of priorities, so either choose 8, 4, 2, or 1.

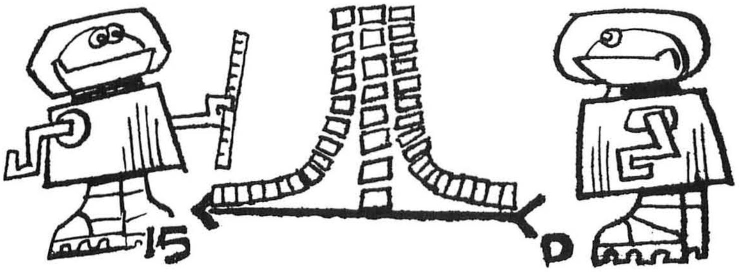
A 5TH PLAYER?

Remember in example 5.5, the Color Utility, how you could press SELECT and have all the Missiles come together into a single Player? This is how it was done. I POKEd 623 witn 17 for both the priority I wanted and to change the colors (lines 1000 to 1050). Then I simply moved their X locations next to each other. Remember that Missiles are only 2 pixels wide each. You can use them together just like a regular Player. The mathematics to keep track of them will be a little harder, that's all.

16 PIXEL WIDE PLAYERS

EXAMPLE 5.13

You now know that you have at least 128 pixels in the Y direction to work with. For Single line resolution you have 256. But so far we have been hindered by only 8 pixels of width. The answer is obvious, but I'll mention it anyway. Just position two Players near each other so they match up! You can write some simple math to always keep them 8 apart in the X direction. Look at example 5.13 to see an ATARI logo made this way.



EXAMPLE 5.14

(32K required)

Just in case you have gotten to this point in the Tutorial, and feel you haven't gotten your money's worth, example 5.14, the last program, is an Editor to create and change your Player shapes. I hope you like it! Here is how to use it:

1) When the program starts, it will ask if you want to load in old data, or create new shapes. Choose NEW.
2) Then Press the SELECT button. The message "Player # to edit" will appear. Pick any number from 0 to 4. Player 4 is made up of the Missiles.
3) A flashing cursor will appear in the editing box on the left. Press D to draw and move the cursor with the four cursor control keys (the CTRL is not needed).
4) To stop drawing, press E (erase). Move the cursor to the next spot you want to draw.
5) While continuing to edit the shape, press START to see the Player's shape updated. The Player will appear next to it's number.
6) You may also move the Player you are currently editing anywhere on the screen by using a Joystick in Port 1.
7) You can change the size of the Player being edited (except #4), by pressing SELECT and choosing 0, 1, or 3 for normal, double, or quadruple size.
8) Press OPTION to edit a different Player.
9) Press S to save the shapes you have made to tape or disk.
10) Press L to load any saved Players back in. Disk users will find a set of Players on the disk already. Remember when you load in a set of Players you won't see them until you press OPTION and request each one!

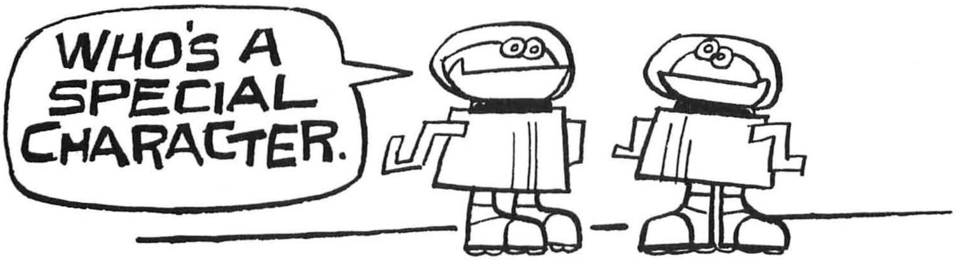
This Editor should allow you to quickly create Players for any purpose. Write to us with suggestions on improvements and I will incorporate your ideas in later versions of this program. The code for this also is a great Tutorial on handling Players by using strings to store them in. Advanced users should study it.

****

At this point let me see if there are any loose ideas that need to be covered. First, let's go way back to the game in example 5.2. You have learned all of the technical aspects of writing a game using PMG. I want to discuss the structuring of a game. This means what goes where in the program coae. You should be able to figure out the rest, but if I've missed anything, please write.

NOTE:

Because of the special characters in this program, the listing beingreferred to will be found a few pages on in the listings section of the manual.



Lines 40 to 60 set the colors. Line 70 points to a subroutine to draw the Playfields. Always do this toward the end of your program so that the main loop will run more quickly. The Playfield in this case consists of just plotting a lot of data. This is similar to the earlier discussion on Playfields.

Line 80 tells you that the program hasn't stopped, but is just setting up itself. This is always a nice touch.

Line 110 is tricky. It's another machine language subroutine! But this one is far more useful than the earlier one. Here I set up four starting locations that are in a safe area of memory, right between PMBASE and PMBASE+768 where the Missile data starts. Since line 20 clears this whole area out, we know it is safe and available. You'll see what goes there in a moment.

Now, in lines 130 to 310, I POKE in the data for two shapes for the Pacman(tm) and two for the Squigily Monster. These start at convenient increments so I can find the shapes later.

Line 330 is the machine code which is stored in a string. You may copy this line out for other programs along with it's dimension statement in line 50 and it's USR call.

Lines 380 to 440 do the normal math that is needed to keep track of X and Y for TWO joysticks.

Lines 450 to 480 are the neat part. First let me tell you what the machine language can do for you. The routine is a memory move routine. It takes the values in the memory locations you tell it, and moves them to any other location. You can use it to move and entire screen of data instantly! I use it to change the shape that is in the area where the data for the two Players is stored. These lines simply tell the computer...

Line 450: "Move the Pacman(tm) data from the location called shape2 to the Player 0 area using 20 bytes of data"

Line 460:"Same thing, but move Squiggly data to Player 1 area"

Line 470:"Move a different Pacman(tm) shape to the Player 0 area"

Line 480:"Move a different Squiggly shape to the Player 1 area.

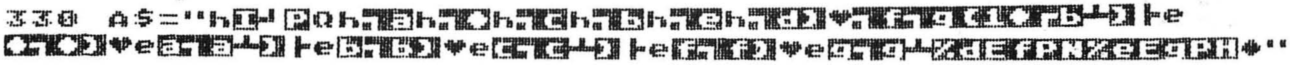
All of the moves above also take care of movement up or down the screen at the same time with the addition of the Y coordinate you want in the "move data to" location. This routine will move all 5 Players and/or the Missiles very quickly. The earlier machine language routine only moved Player 0.

Lines 490 to 570 take care of all the Collisions that we might be interested in, and each branches to a subroutine or makes some change on the same line.

The scoring and sounds are straight forward.... Make a sound... increment a score ...change a color. ALL of these depend on your imagination, i.e., what do you want to happen in the game?

Here are the lines to use for the second machine language routine.

```
50  DIM A$(100)
```



```
330  A$="[illegible]"
450  D=USR (ADR (A$), From Address, To Address, # of bytes to move)
```

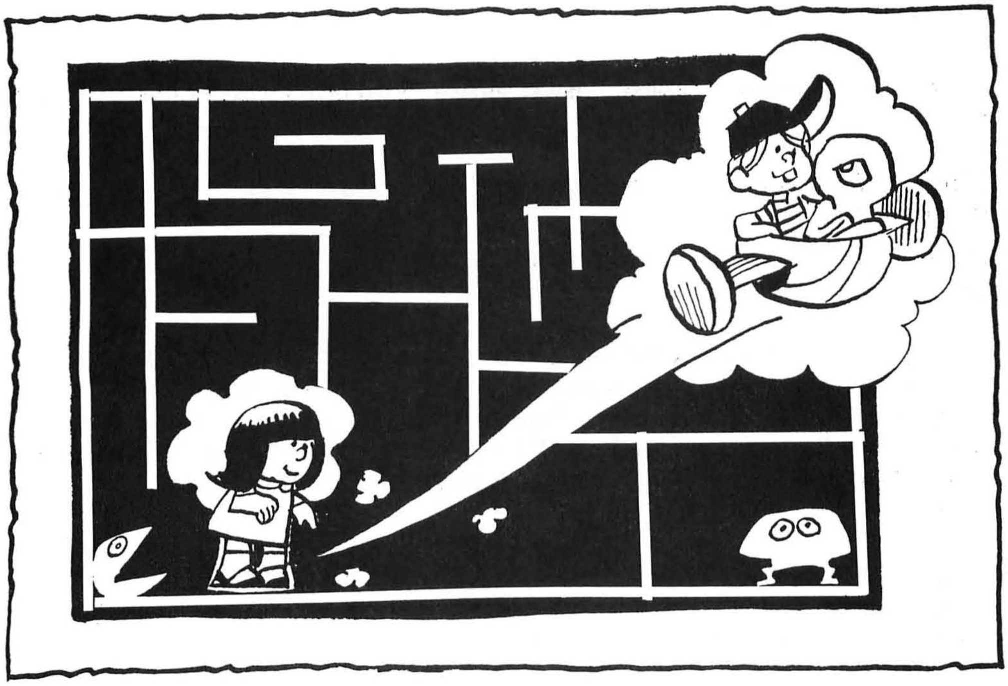
Even though I did not emphasize the use of Missiles very much, I hope you realize that they are handled just like Players except they are two bits wide. A good example of moving them is in example 5.1, lines 640 to 710.

Your main limits are having only two pixels of width, and the Missile data areas all in the same stripe. This means you must POKE 0's into the data area for Missiles you don't want to show on the screen, or just move them off screen with the horizontal position registers.

I left a few strange things in the programs in terms of colors, sizes, and moving shapes off screen when not in use. If you are going to understand PMG, I suggest you now take any of my examples and start to clean up little details to be more to your liking. That will be your final exam...

WILL YOU PASS?

Scores will be mailed out on Friday. BYE!!!



SELECTED PROGRAM LISTINGS

```
5 REM   EX5.2 (c)1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE. PAC
MAN CHARACTER COPYRIGHT ATARI INC.
10 GOTO 580
20 FOR I=PMBASE TO PMBASE+2048:POKE I,0:NEXT I:RETURN
30 REM *****PLAYER SETUP *****
40 GRAPHICS 7:POKE 710,121:POKE 712,0:POKE 704,41:POKE 705,4
1:POKE 559,62
50 DIM A$(100):POKE 623,1
60 POKE 708,103:POKE 752,1:POKE 709,148
70 GOSUB 690:REM DRAW BACKGROUND
80 ? " *** WAIT A  MOMENT  PLEASE ***"
90 A=PEEK(106)-32:POKE 54279,A:PMBASE=A*256
100 POKE 53277,3:POKE 53256,0:POKE 53250,150
110 GOSUB 20:SHAPE1=PMBASE:SHAPE2=PMBASE+50:SHAPE3=PMBASE+10
0:SHAPE4=PMBASE+150
120 RESTORE 160:Y1=172:Y2=172:X1=60:X2=190
130 FOR I=PMBASE TO PMBASE+20:READ B
140 POKE I,B:NEXT I
150 REM ***** DATA FOR PACMAN1 *****
160 DATA 0,0,0,28,62,119,127,126,124,120,124,126,127,126,62,
28,0,0,0,0,0
170 RESTORE 210
180 FOR I=PMBASE+50 TO PMBASE+70:READ B
190 POKE I,B:NEXT I
200 REM ***** DATA FOR PACMAN2 ******
210 DATA 0,0,0,28,62,118,126,126,126,126,127,125,126,62,20,0
,0,0,0,0
220 POKE 559,62
230 RESTORE 260:FOR I=PMBASE+100 TO PMBASE+120:READ B
240 POKE I,B:NEXT I
250 REM ***** DATA FOR MONSTER1 *****
260 DATA 0,0,0,0,124,214,214,254,124,68,68,204,0,0,0,0,0,0,0
,0,0
270 RESTORE 310
280 FOR I=PMBASE+150 TO PMBASE+170:READ B
290 POKE I,B:NEXT I
300 REM ***** DATA FOR MONSTER2 *********
310 DATA 0,0,0,0,124,254,214,214,124,68,68,102,0,0,0,0,0,0,0
,0,0,0
```

```
320 SCOREMON=3:SCOREPAC=3
330 A$="h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]h[illegible]♥[illegible]├e[illegible]♥e[illegible]├e[illegible]♥e[illegible]├e[illegible]♥e[illegible]ZdEFPNZeEGPH♦"
340 ? :? :? :? "STICK1                              STICK2":FOR I=1 TO 30
0:NEXT I
350 REM ******MAIN LOOP ******
360 GOSUB 1030
370 POKE 53278,1:REM CLEAR ALL COLISION LOCATIONS (SO THEY H
OLD 0'S)
380 ST1=STICK(0):S=PEEK(53279)
390 ST2=STICK(1)
400 IF S=3 THEN POKE 764,12:RUN "D:EX5.3"
410 Y1=Y1+(ST1=13)*2-(ST1=14)*2
420 Y2=Y2+(ST2=13)*2-(ST2=14)*2
430 X1=X1+(ST1=7)-(ST1=11):POKE 53248,X1
440 X2=X2+(ST2=7)-(ST2=11):POKE 53249,X2
450 D=USR(ADR(A$),SHAPE2,PMBASE+1024+Y1,20)
460 D=USR(ADR(A$),SHAPE4,PMBASE+1280+Y2,20)
470 D=USR(ADR(A$),SHAPE1,PMBASE+1024+Y1,20)
480 D=USR(ADR(A$),SHAPE3,PMBASE+1280+Y2,20)
490 IF PEEK(53252)=2 THEN POKE 704,200:POKE 705,41:SOUND 0,2
00,10,8:FOR I=1 TO 30:NEXT I:SOUND 0,0,0,0
500 IF PEEK(53252)=1 THEN X1=X1-(ST1=7)+(ST1=11):POKE 53248,
X1:Y1=Y1-(ST1=13)*3+(ST1=14)*3
510 IF PEEK(53253)=2 THEN POKE 705,200:POKE 704,41:SOUND 0,2
00,10,8:FOR I=1 TO 30:NEXT I:SOUND 0,0,0,0
520 IF PEEK(53253)=1 THEN X2=X2-(ST2=7)+(ST2=11):POKE 53249,
X2:Y2=Y2-(ST2=13)*3+(ST2=14)*3
530 IF PEEK(53260)<>0 THEN GOSUB 920:GOTO 570
540 IF PEEK(53253)=4 THEN GOSUB 890
550 IF RND(0)*100>99 THEN COLOR 3:PLOT 35,55:PLOT 135,65
560 IF PEEK(53252)=4 THEN GOSUB 860
570 POKE 53278,0:GOTO 380
580 GRAPHICS 17
590 ? #6;"WE ARE GOING TO PLAY"
600 ? #6;"WITH A COMPLETE GAME"
610 ? #6;" FIRST TO STUDY THE "
620 ? #6;"  FEATURES THAT THE "
630 ? #6;"ATARI(TM) OFFERS US."
640 ? #6;"THEN WE WILL TEACH  "
650 ? #6;"YOU HOW EACH PART OF"
660 ? #6;"THE GAME WAS DONE   "
670 ? #6;"PRESENTING   PACMAN2"
680 FOR I=1 TO 3000:NEXT I:GOTO 30
690 REM **** BACKGROUND ************************
700 RESTORE 740:COLOR 1
710 READ X1,Y1,X2,Y2:IF X1=999 THEN 820
720 PLOT X1,Y1:DRAWTO X2,Y2
730 GOTO 710
740 DATA 20,20,20,0,20,0,0,0,0,0,0,79,0,79,70,79,70,79,70,70
,0,50,10,50,0,40,10,40,10,10,10,30,10,30,20,30
750 DATA 20,30,20,40,20,40,40,40,40,40,40,10,20,50,20,60,20,
60,10,60,10,60,10,70,10,70,50,70,50,70,50,40
760 DATA 30,30,30,0,30,0,70,0,70,0,70,10,50,0,50,30,50,40,60
,40,60,10,60,50,30,50,40,50,30,60,40,60
770 DATA 70,30,90,30,90,30,90,50,70,50,70,30,70,20,100,20,10
0,20,100,60,90,10,90,20,80,10,80,0
780 DATA 80,0,159,0,159,0,159,70,159,79,80,79,80,79,80,60,80
,60,60,60,60,60,60,70
790 DATA 90,60,90,70,90,70,150,70,150,70,150,40,140,60,130,6
0,130,60,130,70,130,40,110,40,110,40,110,70
800 DATA 100,30,130,30,110,10,110,20,110,20,150,20,140,20,14
0,50,140,50,120,50,120,50,120,60
810 DATA 150,10,150,30,140,0,140,10,130,10,130,20,120,0,120,
10,100,0,100,10,999,99,9,9
```

```
820 REM ******** FILL BOXES ********
830 COLOR 2:FOR I=1 TO 20:PLOT 70,30+I:DRAWTO 90,30+I:NEXT I

840 COLOR 3
850 RETURN
860 REM ***** SCORE FOR PAC ************
870 COLOR 0:PLOT 35,55:PLOT 135,65:SCOREPAC=SCOREPAC+1:FOR I
=50 TO 1 STEP -1:SOUND 0,I,10,8:NEXT I
880 GOSUB 1030:RETURN
890 REM *** SCORE MONSTER *******
900 COLOR 0:PLOT 35,55:PLOT 135,65:SCOREMON=SCOREMON+1:FOR I
=50 TO 1 STEP -1:SOUND 0,I,10,8:NEXT I
910 GOSUB 1030:RETURN
920 REM **** PAC HITS MONSTER ***********
930 IF PEEK(53260)=2 AND PEEK(704)=200 THEN SCOREPAC=SCOREPA
C+5:GOSUB 1030:GOTO 950
940 GOTO 960
950 FOR I=200 TO 0 STEP -1:SOUND 0,I,10,8:NEXT I
960 REM **** MON HITS PACMAN ******
970 IF PEEK(53261)=1 AND PEEK(705)=200 THEN SCOREMON=SCOREMO
N+5:GOSUB 1030:GOTO 990
980 GOTO 1000
990 FOR I=0 TO 200 STEP 1:SOUND 0,I,10,8:NEXT I:SOUND 0,0,0,
0
1000 X1=60:X2=190:Y1=172:Y2=172
1010 DUM=USR(ADR(A$),PMBASE+200,PMBASE+1024+Y,240)
1020 DUM=USR(ADR(A$),PMBASE+200,PMBASE+1024,510)
1030 POKE 656,1:POKE 657,2:? "PACMAN= ";SCOREPAC;"
 SQUIGILY= ";SCOREMON:RETURN
```

```
4 REM EX 5.6
5 REM (C) 1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE. WRITTEN B
Y ROBIN SHERER
10 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 410
20 REM ***** PLOT SHAPE ******
30 COLOR 0
40 D=D+(ST=6)+(ST=7)+(ST=5)-(ST=9)-(ST=10)-(ST=11)
50 E=E+(ST=13)+(ST=9)+(ST=5)-(ST=6)-(ST=10)-(ST=14)
60 FOR X=0 TO 9:FOR Y=0 TO 25:PLOT X+D,Y+E:NEXT Y:NEXT X
70 RESTORE 150:COLOR 1
80 FOR N=1 TO 100:READ X:READ Y
90 IF X=0 AND Y=0 THEN RETURN
100 PLOT X+D,Y+E:NEXT N
110 E$(1,60)="hhhJH▣ ▀█ 1K◤ ▛KHHPwhJH▣ ▀█K1KH▛K◤▛PwhJH▣◥FMZM▄♥P
hJ▣◥FMZM▄♥P♦♥♥"
120 REM POKE 53248,30
140 SOUND 2,0,8,2:FOR VOL=1 TO 15 STEP 0.1:SOUND 0,25,4,VOL:
SOUND 1,13,4,VOL
150 DATA 5,0,3,1,4,1,5,1,6,1,2,2,3,2,4,2,5,2,6,2,7,2,1,3,2,3
,7,3,8,3,2,4,3,4,4,4,5,4,6,4,7,4,3,5,4,5,5,5,6,5
160 DATA 4,6,5,6,2,7,3,7,4,7,5,7,6,7,7,7,1,8,3,8,6,8,8,8,1,9
,8,9,2,10,4,10,5,10,7,10,2,11,4,11,5,11,7,11
170 DATA 2,12,7,12,3,13,6,13,3,14,6,14,4,15,5,15,4,16,5,16,4
,17,5,17,4,18,5,18,3,19,4,19,5,19,6,19
180 DATA 2,20,3,20,4,20,5,20,6,20,7,20,1,21,2,21,3,21,4,21,5
,21,6,21,7,21,8,21,2,22,3,22,4,22,5,22,6,22,7,22
190 DATA 3,23,4,23,5,23,6,23,4,24,5,24,0,0,0,0
200 REM *****PLAYER SETUP *****
210 DIM E$(60):GRAPHICS 6:D=20:E=50:POKE 559,0:GOSUB 20:? "
   WHICH IS THE PLAYER?":POKE 710,148:POKE 712,148
220 ? "SELECT = SIZES  OPTION = NEXT EX."
230 E$(1,60)="hhhJH▣ ▀█♥1K◤▛KHHPwhJH▣ ▀█♥1KH▛K◤▛PwhJH▣◥FMZM▄♥P
hJ▣◥FMZM▄♥P♦♥♥"
```

```
4 REM EX5.8
5 REM (C) 1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE(ROBIN SHER
ER)
10 GOTO 30
20 FOR I=PMBASE+1024 TO PMBASE+1536:POKE I,0:NEXT I:RETURN
30 GOSUB 200:REM DRAW MOUNTAINS
40 DIM E$(60)
50 GOSUB 300:REM DRAW STARS
60 POKE 710,0:POKE 712,0:POKE 704,123
70 GOSUB 380:REM SETUP PLAYERS
80 POKE 204,A+4:POKE 203,0:POKE 205,120
90 E$(1,60)="hhhJH⊠[illegible]hJH⊠[illegible]hJH⊠◥[illegible]▬♥⊠h
J⊠◥[illegible]▬♥⊠♦♥♥"
100 REM POKE 53248,30
110 REM ***MAIN LOOP *********
120 SOUND 2,0,8,2:FOR VOL=1 TO 15 STEP 0.1:SOUND 0,25,4,VOL:
SOUND 1,13,4,VOL
130 A=USR(ADR(E$),STICK(0))
140 NEXT VOL
150 FOR VOL=14 TO 0 STEP -0.1:SOUND 0,25,4,VOL:SOUND 1,13,4,
VOL
160 A=USR(ADR(E$),STICK(0))
170 NEXT VOL
180 IF PEEK(53279)=3 THEN POKE 764,12:RUN "D:EX5.9"
190 GOTO 110
200 REM **** PLOT BACKGROUND *****
210 GRAPHICS 8+16:RESTORE 260
220 COLOR 1
230 READ X,Y:PLOT X,Y
240 READ X,Y:IF X>900 THEN RETURN
250 DRAWTO X,Y+10:GOTO 240
260 DATA 0,157,27,166,39,166,49,166,57,157,57,158,69,158,74,
164,87,170,87,171,90,169,95,140
270 DATA 110,70,112,65,114,50,116,63,127,93,130,97,140,97,15
3,63,159,57,165,84,168,91,175,103,181,123
280 DATA 190,174,199,174,205,165,207,149,229,133,245,156,256
,133,271,120,283,165,287,172,291,164,300,89,305,76
290 DATA 310,63,999,999
300 REM **** STARS *****
310 TRAP 370
320 RESTORE 340
330 READ X,Y:PLOT X,Y:GOTO 330
340 DATA 2,4,5,8,9,10,13,43,34,65,7,3,23,66,123,45,300,67,30
1,65,246,34,234,49,261,68,241,133,123,55,67,98
350 DATA 12,64,75,58,119,10,133,43,4,65,7,93,203,66,12,45,17
8,67,145,65,46,34,187,49,123,68,21,133,243,55,116,74
360 DATA 123,5,156,7,213,8,295,12,296,34,294,56,230,56,230,6
,78,9
370 RETURN
380 REM **** SETUP PLAYERS ****
390 A=PEEK(106)-40:POKE 54279,A:PMBASE=A*256
400 GOSUB 20:REM CLEAR PM AREA
410 POKE 53277,3:REM TURN ON PM DMA
420 Z=20:REM INITIAL Y POSITION OF SHIP
430 X=150:REM INITIAL X POSITION OF SHIP
440 POKE 53248,X:POKE 53249,X:REM POSITION SHIP (PLAYER0) AN
D FLAME (PLAYER1)
450 RESTORE 480
460 FOR I=PMBASE+1024+Z TO PMBASE+1280:READ SHAPE:IF SHAPE<>
0 THEN POKE I,SHAPE:NEXT I
470 POKE 623,1:REM SET PRIORITY
480 DATA 8,60,126,195,126,60,24,126,165,129,90,0,0,0
490 POKE 559,62
500 RETURN
```

```
240 A=PEEK(106)-16:POKE 54279,A:POKE 204,A+4:POKE 203,0:PMBA
SE=A*256
250 POKE 53277,3:POKE 704,40:POKE 53248,150:POKE 205,120:POK
E 53256,0
260 FOR I=PMBASE+1024 TO PMBASE+1280:POKE I,0:NEXT I
270 RESTORE 290
280 FOR I=PMBASE+1155 TO PMBASE+1209:READ B:POKE I,B:NEXT I
290 DATA 8,8,60,60,126,126,195,195,126,126,60,60,24,24,126,1
26,165,165
300 DATA 129,129,90,90,90,90,66,66,36,36,36,36,24,24,24,24,2
4,24,24,24
310 DATA 60,60,126,126,255,255,126,126,60,60,24,24,0,0,0,0,0

320 POKE 559,62
330 REM
340 REM *******MAIN LOOP ******
350 REM
360 IF STRIG(0)=0 THEN GOSUB 20:FLAG=0
370 ST=STICK(0):S=PEEK(53279)
380 IF S=5 THEN GOSUB 670
390 IF S=3 THEN POKE 764,12:POKE 53256,0:RUN "D:EX5.7"
400 A=USR(ADR(E$),STICK(0)):GOTO 360
410 GRAPHICS 17:POKE 712,148
420 ? #6;" THIS NEXT EXAMPLE  "
430 ? #6;"  SHOWS YOU THE     "
440 ? #6;"DIFFERENCE BETWEEN A"
450 ? #6;"NORMALLY DRAWN SHAPE"
460 ? #6;"AND A PLAYER....use "
470 ? #6;"YOUR joystick TO SEE"
480 ? #6;"WHICH IS THE PLAYER!"
490 FOR I=1 TO 3000:NEXT I:GOTO 210
500 REM ***** PLOT SHAPE ******
510 COLOR 1
520 D=D+(ST=6)+(ST=7)+(ST=5)-(ST=9)-(ST=10)-(ST=11)
530 E=E+(ST=13)+(ST=9)+(ST=5)-(ST=6)-(ST=10)-(ST=14)
540 RESTORE 370
550 FOR N=1 TO 100:READ X:READ Y
560 IF X=0 AND Y=0 THEN 110
570 PLOT X+D,Y+E:NEXT N
580 IF FLAG=O THEN COLOR 0:GOTO 130
590 RETURN
600 FOR X=1 TO 8:FOR Y=0 TO 25:PLOT X+D,Y+E:NEXT Y:NEXT X
610 RETURN
620 DATA 5,0,3,1,4,1,5,1,6,1,2,2,3,2,4,2,5,2,6,2,7,2,1,3,2,3
,7,3,8,3,2,4,3,4,4,4,5,4,6,4,7,4,3,5,4,5,5,5,6,5
630 DATA 4,6,5,6,2,7,3,7,4,7,5,7,6,7,7,7,1,8,3,8,6,8,8,8,1,9
,8,9,2,10,4,10,5,10,7,10,2,11,4,11,5,11,7,11
640 DATA 2,12,7,12,3,13,6,13,3,14,6,14,4,15,5,15,4,16,5,16,4
,17,5,17,4,18,5,18,3,19,4,19,5,19,6,19
650 DATA 2,20,3,20,4,20,5,20,6,20,7,20,1,21,2,21,3,21,4,21,5
,21,6,21,7,21,8,21,2,22,3,22,4,22,5,22,6,22,7,22
660 DATA 3,23,4,23,5,23,6,23,4,24,5,24,0,0,0,0
670 REM ***** SIZES *****
680 ? :? :? "PRESS 0, 1, OR 3 FOR SIZE":? "SELECT = SIZES
OPTION = NEXT EX."
690 POKE 764,255
700 PK=PEEK(764)
710 IF PK=50 THEN POKE 53256,0:RETURN
720 IF PK=31 THEN POKE 53256,1:RETURN
730 IF PK=26 THEN POKE 53256,3:RETURN
740 GOTO 700
```

```
3 REM (C) 1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE(ROBIN SHER
ER)
4 REM EX 5.9
5 FOR I=1 TO 8:POKE 53247+I,0:NEXT I
8 GOTO 100
10 DIM E$(60)
20 E$(1,60)="hhhJH8 ▀█♥IK◤FKHHPWhJH8 ▀█♥IKHFK◤◤PWhJH8╲FMZM▄♥Ph
J8╲FMZM▄♥P♦♥♥"
30 A=PEEK(106)-16:POKE 54279,A:POKE 204,A+4:POKE 203,0:PMBAS
E=A*256
40 POKE 559,62:POKE 53277,3:POKE 704,40:POKE 205,120:POKE 53
256,1
50 FOR I=PMBASE+1024 TO PMBASE+1280:POKE I,0:NEXT I
60 FOR I=PMBASE+1100 TO PMBASE+1127:READ B:POKE I,B:NEXT I
70 DATA 8,60,126,195,126,60,24,126,165,129,0,90,90,66,36,36,
24,24,24,0,24,60,126,255,126,60,24,0
75 POKE 53248,150
80 A=USR(ADR(E$),STICK(0)):IF PEEK(53279)=3 THEN POKE 764,12
:POKE 53256,0:RUN "D:EX5.10"
90 GOTO 80
100 GRAPHICS 17
102 ? #6;"WE MENTIONED THAT    "
104 ? #6;"YOU MUST BE CAREFUL "
106 ? #6;"WHERE YOU PLACE THE "
110 ? #6;"PLAYER/MISSILE DATA."
120 ? #6;"HERE IS AN EXAMPLE  "
125 ? #6;"OF WHAT CAN GO WRONG"
130 ? #6;"IF YOU LOCATE THE   "
140 ? #6;"DATA POORLY.        "
150 ? #6;"                    "
160 ? #6;"                    "
162 FOR I=1 TO 3000:NEXT I:POKE 752,1
165 GRAPHICS 7:POKE 712,148
180 ? "MOVE PLAYER BOTH DIRECTIONS":? "PRESS OPTION TO GO ON
"
190 GOTO 10
2000 REM ****** PLOT SHAPE ******
2010 COLOR 1
2015 RESTORE 2100
2020 FOR N=1 TO 100:READ X:READ Y
```

```
3 REM (C) 1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE(ROBIN SHER
ER)
5 REM EX5.10
10 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:POKE 559,0:GOTO 30
20 FOR I=PMBASE+512 TO PMBASE+1280:POKE I,0:NEXT I:RETURN
30 GRAPHICS 0:? :? :? " PRESS OPTION TO GO ON":POKE 752,1
40 ? :? " PRESS SELECT FOR SINGLE LINE          RESOLUTION P
LAYER"
50 ? :? " PRESS START FOR DOUBLE LINE           RESOLUTION P
LAYER"
70 POKE 704,40
80 PMB=PEEK(106)-16
90 POKE 54279,PMB
100 PMBASE=PMB*256
110 POKE 53277,3:GOSUB 20:POKE 559,34
120 X=125:Y=110:YSAVE=100
130 POKE 53256,0:GOTO 200
140 REM ****** MAIN LOOP *******
150 P=PEEK(53279)
160 IF P=5 THEN POKE 559,62:POKE 53248,100
170 IF P=6 THEN POKE 559,46:POKE 53248,100
180 IF P=3 THEN POKE 764,12:RUN "D:EX5.11"
190 GOTO 150
```

```
200 REM
210 REM SINGLE LINE RESOLUTION ********
220 REM
230 RESTORE 290:CNT=0
240 FOR I=PMBASE+1024+Y TO PMBASE+1280
250 READ B:IF B=0 THEN 300
260 CNT=CNT+1
270 POKE I,B
280 NEXT I
290 DATA 124,214,214,254,124,68,68,204,0,0,0
300 REM
310 REM DOUBLE LINE RESOLUTION ******
320 REM
330 RESTORE 290:CNT=0
340 FOR I=PMBASE+512+Y TO PMBASE+640
350 READ B:IF B=0 THEN 390
360 CNT=CNT+1
370 POKE I,B
380 NEXT I
390 GOTO 150
```

```
4 REM EX5.13
5 REM (C) 1982 BY SANTA CRUZ EDUCATIONAL SOFTWARE(ROBIN SHER
ER)
10 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 240
20 FOR I=PMBASE+1024 TO PMBASE+1792:POKE I,0:NEXT I:RETURN
30 REM *****PLAYER SETUP *****
40 GRAPHICS 7:POKE 710,0:POKE 712,0:POKE 704,41:POKE 706,41:
POKE 559,62
50 A=PEEK(106)-24:POKE 54279,A:POKE 204,A+4:POKE 203,0:PMBAS
E=A*256
60 POKE 53277,3:X=150:POKE 205,120:POKE 53256,0:POKE 53250,1
50
70 GOSUB 20
80 RESTORE 110:Y=150
90 FOR I=PMBASE+1024+Y TO PMBASE+1209:READ B:IF B<>0 THEN PO
KE I,B:NEXT I
100 REM ***** DATA FOR FIRST HALF ***
110 DATA 2,2,2,2,2,2,2,2,2,4,4,4,8,112,0,0,0
120 RESTORE 150
130 FOR I=PMBASE+1536+Y TO PMBASE+1792:READ B:IF B<>0 THEN P
OKE I,B:NEXT I
140 REM ***** DATA FOR 2ND HALF *****
150 DATA 160,160,160,160,160,160,160,160,160,144,144,144,136
,135,0,0,0
160 POKE 559,62
165 ? "           16 BIT WIDE SHAPE"
167 POKE 752,1
168 ? "       <==== USE JOYSTICK =====>"
170 REM
180 REM ******MAIN LOOP ******
190 REM
200 IF STRIG(0)=0 THEN GOSUB 20:FLAG=0
210 ST=STICK(0):S=PEEK(53279)
220 IF S=3 THEN POKE 764,12:RUN "D:EX5.14"
225 X=X+(ST=7)-(ST=11):POKE 53248,X:POKE 53250,X+8
230 GOTO 200
240 GRAPHICS 17
250 ? #6;"  DO YOU WANT MORE  "
260 ? #6;"COMPLICATED SHAPES? "
270 ? #6;"JUST USE TWO PLAYERS"
280 ? #6;"NEXT TO EACH OTHER.,"
290 ? #6;"..this example is  "
300 ? #6;"almost the same as  "
310 ? #6;"before, but........."
320 FOR I=1 TO 3000:NEXT I:GOTO 40
```

santa cruz EDUCATIONAL SOFTWARE

TRICKY TUTORIAL #6
SOUND

Educational Software

presents

Tricky Tutorial(tm) #6
SOUNDS

by JERRY WHITE

This program starts with the simple sound statement, but progresses to chords, complete songs and special effects. It also demonstrates the use of direct pokes to the computers built in four channel frequency controls. All of the material can be used by a beginner, yet if it is studied, you will learn many of the tricks that Jerry White puts into his other musical programs for the ATARI(Name That Song, Player Piano, My First Alphabet's tunes). MUSIC FROM BASIC BECOMES ALMOST EASY!

This program requires 16k for tape users or 24k for disk, and a basic cartridge.

Educational Software Inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

SOUND TUTORIAL — NOTES:

This tutorial is different than the previous five in this series. Like the others, it makes a complicated subject usable for the average ATARI owner. Also, most people feel it is an excellant value for the price. However, the format is quite different in that the program is really self documenting. Also, this is the first tutorial written by someone other than myself. Both of these changes are for the better, so I hope past purchasers of TRICKY TUTORIALS like this one to.

I suggest you learn the program by first running(and playing with) all seven parts. Then, when you find a specific area that you want to understand better, go to that program and load it into your memory. Tape users should write down the starting number of each program. Now, try looking at the program code itself to see what Jerry has done. Think of a way you would like to modify the program, and go at it! For example, the song in part three,"DOE RAY ME", could be changed by modifing the data statements. For your use, this part is reprinted on the next page.

Following that page you will find a chart of the notes and pitches available for the basic sound statement. Then comes the program for the special sound effects. You can either retype these in to your own programs as needed, modify them for new effects, or just resave them by themselves. Note that each little effect is not quite complete by itself, but needs the variables V0-V3 defined . Finally, Jerry's special gift to us is part seven for which both the listings and instructions are included.

For an interesting effect take out the remark in line 74 of parts 1,2, & 4. Most people I showed this to didn't like the sound associated with each letter being put on the screen. You may, or perhaps you might change the pitch values in the sound statement in 74 to get your own unique effect.

It's up to you to experiment with sound. This program is the foundation. Write and let us know what you think of it.

Educational Software inc.
4565 Cherryvale
Soquel, CA 95073
(408) 476-4901

BYE ,
Robin Sherer

SOUND.3 Documentation

While many of the programs in this package display documentation, SOUND.3 may require the printed type. This program plays a song and displays sing along words on the screen. If you understand this program, and you can read sheet music, you may wish to write you own BASIC songs.

If you don't know how to read sheet music, you might get by if you have a good ear and use the old trial and error method. If you don't have either, and you can't sing, then just listen.

You will need some knowledge of music to write music. I will have to assume that you understand the following musical terms: NOTE or PITCH, CHORD, SHARP, and FLAT. If you're lost already, the rest of this program documentation won't help you. Let us know if you'd like a tutorial on reading music and describing the previously mentioned musical terms.

Let's walk through the SOUND.3 program listing. We begin by DIMensioning a string called LINE$ and an array called "N" which will store 50 pitches. These pitches will correspond to musical notes. LINE$ will store one line of words in our song. We then set V0=0 (Voice 0), V1=1 (VOICE 1), etc., then GOTO 100.

Why the heck did he GOTO a line of DATA? I actually should have gone to line 120, but ATARI BASIC will bail me out a just fall through to line 120, which sends us off to a subroutine at line 21000. That routine just clears the screen, displays the heading, POKEs location 77 with a zero, and returns. In case you don't know what that POKE does, it temporarily defeats ATARI's automatic color changing routine which is also known as attract mode.

RETURNing to line 120, we read data for 50 notes into the N (NOTE) array. Then I TRAPped to line 19000 which makes no sense at all since there is no line 19000 in this program. If you decide to make modifications to this program, get rid of that useless TRAP. By the way, the reason I got away with it was only because there are no other errors in this program. The POKE 82,8 indents the left margin.

Before we start reading more data at line 210, it is important to understand the use of the "N" array and the subroutines found from line 30 through line 74.

Look at the DISTORTION LEVEL 10 PITCH CHART. The first note is quite logically numbered 1. It's corresponding PITCH is 14 and the musical note is C. This is a very high sound. The higher the sound, the lower the NOTE # and PITCH value. In our "N" array, N(1) contains a

14. The DATA in lines 100 and 110 correspond to the PITCHes on the CHART.

Why use NOTE numbers in an array when ATARI supplies PITCH values in their BASIC REFERENCE manual? I'm glad you asked. Start reading the PITCH numbers on the PITCH chart until you get to the number 21. What happened to 20? If you look further down the chart, you'll notice an increasing number of missing numbers. Now look down the column of NOTE #'s. You will find 50 consecutive numbers. This provides us with a quick and easy way to let BASIC calculate chords when we supply only the base note of the desired chord.

The subroutine beginning at line 40 is our chord calculator. You need only supply it with the NOTE # in the variable "P". The routine assumes that P will be at least 8 and not greater than 50. It then sets P0 (PITCH 0) equal to N(P). Then it calculates our chord and stores the pitches in P1, P2, and P3. In line 42, we turn on all 4 voices. Notice that voice 0 is set at a greater volume, and the three notes of our chord are played at a lower volume but equal to each other.

In line 50 we get to our WAIT routine. We POKE the value stored in the variable WAIT into location 540. This location counts backwards to zero at the rate of 60 per second (JIFFIES). We just waste time at line 52 until the countdown is completed. Then we turn off VOICE 0 ONLY and RETURN.

So what did all this accomplish? In plain English, we played a melody note along with a chord, then turned off the melody note. The chord continues to play. The subroutine at line 60 is used to turn off ALL voices.

The subroutine at line 30 will change only the melody pitch, then go on to the WAIT routine.

The subroutine at line 70 turns on all four voices at equal volume, then decreases the volume gradually, until all sounds are off.

Now, where were we??? Ah yes, line 210 where we read LINE$,CHORD,P,WAIT:PRINT LINE$:and go off to the subroutine beginning at line 40. We are reading the DATA which begins at line 1000. We read the words, "DOE A DEER A FEMALE DEER" into a string then put it on the screen. We also read the number 49 into the variable CHORD, 37 into the variable P, and 45 into the variable WAIT.

Remember, we GOSUB 40 to play a melody note, calculate our chord, play the chord, and kill some time. The best way to learn from examining someone elses program, is by acting as if you were the computer. Follow the instructions, and see what you, or the computer, will do. Let's try it.

I'll be the computer this time. I have just read the data as I was instructed to do in line 210, and now I'm at the subroutine at line 40. I am told to make P0=N(P). I just read DATA and set the value of P=37. I look up the value of N(37) and see that it is 121. I set P0=121.

My next instruction is to make P1=N(CHORD). I read the value of CHORD in line 210 and know that CHORD=49. I lookup the value of N(49) which is 243, and set P1=243. P2 must be set to the value stored in N(45) and P3 must be set to the value stored in N(42). P2=193 and P3=162.

I turn on all four voices as indicated in line 42 then POKE the number 45 into my memory location 540. At line 52 I look at the value stored in memory location 540 and compare it to 0. It's not 0 so I check it again. Each time I check that location, it's value is less than it was the last time I looked, but it's not zero so I keep checking. I'm getting bored.

Finally I find a zero and go on to line 54. I turn off the sound of Voice 0 then RETURN from this subroutine to line 220.

Now it's your turn. Continue through the program doing what I just did. If you start getting confused, take a pencil and write down values as you read and change them. You should soon understand what I've put your computer through, to play this simple song.

The logic in this program is not suitable for all songs. You will have to make minor modifications for different tempos, or if other than standard Major chords are required. This program demonstrates one way to play a simple song and an easy method of finding the notes of a chord. Don't think you can just add a few lines of DATA and create the Nutcracker Suite.

If you just want to enter the music, then see it in sheet music form while it is played, I'd recommend ATARI's MUSIC COMPOSER. If you'd like to see your melody as it would be played on a piano, or play your keyboard as if it were a piano, consider the SANTA CRUZ SOFTWARE PLAYER PIANO package. I know the author, he used to be a piano player.

SOUND.3 (c) 1981 by Jerry White

```
10    DIM LINE$(40),N(50):
      V0=0:
      V1=1:
      V2=2:
      V3=3:
      GOTO 100
30    SOUND V0,N(P),10,14:
      GOTO 50
40    P0=N(P):
      P1=N(CHORD):
      P2=N(CHORD-4):
      P3=N(CHORD-7)
42    SOUND V0,P0,10,14:
      SOUND V1,P1,10,6:
      SOUND V2,P2,10,6:
      SOUND V3,P3,10,6
50    POKE 540,WAIT
52    IF PEEK(540)<>0 THEN 52
54    SOUND V0,0,0,0:
      RETURN
60    FOR OFF=0 TO 3:
          SOUND OFF,0,0,0:
      NEXT OFF:
      RETURN
70    P0=N(P):
      P1=N(CHORD):
      P2=N(CHORD-4):
      P3=N(CHORD-7)
72    FOR DECAY=8 TO 0 STEP -1:
          SOUND V0,P0,10,DECAY:
          SOUND V1,P1,10,DECAY:
          SOUND V2,P2,10,DECAY:
          SOUND V3,P3,10,DECAY
74    NEXT DECAY:
      RETURN
100   DATA 14,15,16,17,18,19,21,22,23,24,26,27,29,31,33,
  35,37,40,42,45,47,50,53,57,60,64,68,72,76,81,85,91,96
110   DATA 102,108,114,121,128,136,144,
      153,162,173,182,193,204,217,230,243,255
120   GOSUB 21000:
      FOR X=1 TO 50:
          READ IT:
          N(X)=IT:
      NEXT X
200   TRAP 19000:
      POKE 82,8:
      ? " ":
      ?
210   READ LINE$,CHORD,P,WAIT:
      ? LINE$:
      GOSUB 40
220   FOR ME=1 TO 6:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME:
      GOSUB 60:
      WAIT=10:
      GOSUB 50
230   READ LINE$,CHORD,P,WAIT:
      ? :
      ? LINE$:
      GOSUB 40
240   FOR ME=1 TO 6:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME:
      GOSUB 60:
      WAIT=30:
      GOSUB 50
250   READ LINE$,CHORD,P,WAIT:
      ? :
      ? LINE$:
      GOSUB 40
260   FOR ME=1 TO 6:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME:
      GOSUB 60:
      WAIT=10:
      GOSUB 50
270   READ LINE$,CHORD,P,WAIT:
      ? :
      ? LINE$:
      GOSUB 40
280   FOR ME=1 TO 6:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME:
      GOSUB 60:
      WAIT=30:
      GOSUB 50
290   READ LINE$,CHORD,P,WAIT:
      ? :
      ? LINE$:
      GOSUB 40
300   FOR ME=1 TO 5:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME
310   READ CHORD,P,WAIT:
      GOSUB 40:
      GOSUB 60:
      WAIT=30:
      GOSUB 50
320   READ LINE$,CHORD,P,WAIT:
      ? :
      ? LINE$:
      GOSUB 40
330   FOR ME=1 TO 5:
          READ P,WAIT:
          GOSUB 30:
      NEXT ME
340   READ CHORD,P,WAIT:
      GOSUB 40:
      GOSUB 60:
      WAIT=30:
      GOSUB 50
350   READ LINE$,CHORD,P,WAIT:
      ? :
        ? LINE$:
        GOSUB 40
360     FOR ME=1 TO 5:
            READ P,WAIT:
            GOSUB 30:
        NEXT ME
370     READ CHORD,P,WAIT:
        GOSUB 40:
        GOSUB 60:
        WAIT=10:
        GOSUB 50
380     READ LINE$,CHORD,P,WAIT:
        ? :
        ? LINE$:
        GOSUB 40
390     READ P,WAIT:
        GOSUB 30
400     READ CHORD,P,WAIT:
        GOSUB 40
410     READ P,WAIT:
        GOSUB 30
420     READ CHORD,P,WAIT:
        GOSUB 40
430     READ P,WAIT:
        GOSUB 30
440     READ CHORD,P,WAIT:
        GOSUB 40
450     GOSUB 60:
        WAIT=10:
        GOSUB 50:
        FOR DECAY=15 TO 0 STEP -0.5:
            SOUND V0,N(1),10,DECAY:
        NEXT DECAY
460     GRAPHICS 18:
        ? #6:
        ? #6;"    major chords"
510     FOR ME=1 TO 8:
            READ CHORD,P,LINE$:
            POSITION ME*2,10-ME:
            ? #6;LINE$:
            GOSUB 70:
        NEXT ME
530     FOR ME=8 TO 1 STEP -1:
            READ CHORD,P,LINE$:
            POSITION ME*2,10-ME:
            ? #6;LINE$:
            GOSUB 70:
        NEXT ME
540     WAIT=15:
        GOSUB 50:
        POSITION 4,11:
        ? #6;"PRESS"
550     FOR DECAY=15 TO 0 STEP -0.5:
            SOUND V0,N(6),10,DECAY:
        NEXT DECAY
560     WAIT=15:
        GOSUB 50:
        POSITION 10,11:
        ? #6;"START"
```

```
570     FOR DECAY=15 TO 0 STEP -0.5:
            SOUND V0,N(1),10,DECAY:
        NEXT DECAY
600     SETCOLOR 0,PEEK(20),10:
        IF PEEK(53279)<>6 THEN 600
700     GRAPHICS 18:
        SETCOLOR 0,1,10:
        SETCOLOR 1,11,12:
        SETCOLOR 3,4,12
710     ? #6:
        ? #6:
        ? #6;"    PRESS option":
        ? #6;"      TO RERUN"
720     WAIT=60:
        GOSUB 50
730     ? #6:
        ? #6;"    PRESS start":
        ? #6;"    TO CONTINUE"
740     IF PEEK(53279)=3 THEN
           RUN
750     IF PEEK(53279)=6 THEN 900
760     GOTO 740
900     RUN "D:SOUND.4"
1000    DATA DOE A DEER A FEMALE DEER
1010    DATA 49,37,45,35,15,33,45,37,15,33,30,37,30,33,45
1020    DATA RAY A DROP OF GOLDEN SUN
1030    DATA 42,35,45,33,15,32,15,32,15,33,15,35,15,32,90
1040    DATA ME A NAME I CALL MYSELF
1050    DATA 49,33,45,32,15,30,45,33,15,30,30,33,30,30,45
1060    DATA FA' A LONG LONG WAY TO RUN
1070    DATA 44,32,45,30,15,28,15,28,15,30,15,32,15,28,90
1080    DATA SEW A NEEDLE PULLING THREAD
1090    DATA 49,30,45,37,15,35,15,33,15,32,15,30,15,44,28,90
1100    DATA LA A NOTE TO FOLLOW SEW
1110    DATA 44,28,45,35,15,33,15,31,15,30,15,28,15,42,26,90
1120    DATA TEA A DRINK WITH JAM AND BREAD
1130    DATA 42,26,45,33,15,31,15,29,15,28,15,26,15,49,25,90
1140    DATA THAT WILL BRING US BACK TO DOE
1150    DATA 49,25,15,26,15
1160    DATA 44,28,30,32,30
1170    DATA 42,26,30,30,30
1180    DATA 49,25,90
1200    DATA 49,37,C
1210    DATA 47,35,D
1220    DATA 45,33,E
1230    DATA 44,32,F
1240    DATA 42,30,G
1250    DATA 40,28,A
1260    DATA 38,26,B
1270    DATA 37,25,C
1300    DATA 37,25,c
1310    DATA 38,26,b
1320    DATA 40,28,a
1330    DATA 42,30,g
1340    DATA 44,32,f
1350    DATA 45,33,e
1360    DATA 47,35,d
1370    DATA 49,37,c
21000   GRAPHICS 0:
        SETCOLOR 2,9,0:
        SETCOLOR 4,9,0:
        SETCOLOR 1,9,12:
        POKE 752,1:
        POKE 82,2:
        POKE 83,39:
        POKE 201,7
21010   ? " ":
        ? ,"QRRRRRRRRRRRRRE"
21020   ? ,"| MAJOR CHORD HARMONY |"
21030   ? ,"ZRRRRRRRRRRRRRC"
21040   POKE 77,0:
        RETURN
```

DISTORTION LEVEL 10

PITCH CHART

NOTE #	PITCH	MUSICAL NOTE
1	14	C
2	15	B
3	16	A# or Bb
4	17	A
5	18	G# or Ab
6	19	G
7	21	F# or Gb
8	22	F
9	23	E
10	24	D# or Eb
11	26	D
12	27	C# or Db
13	29	C
14	31	B
15	33	A# or Bb
16	35	A
17	37	G# or Ab
18	40	G
19	42	F# or Gb
20	45	F
21	47	E
22	50	D# or Eb
23	53	D
24	57	C# or Db
25	60	C
26	64	B
27	68	A# or Bb
28	72	A
29	76	G# or Ab
30	81	G
31	85	F# or Gb
32	91	F
33	96	E
34	102	D# or Eb
35	108	D
36	114	C# or Db
37	121	C
38	128	B
39	136	A# or Bb
40	144	A
41	153	G# or Ab
42	162	G
43	173	F# or Gb
44	182	F
45	193	E
46	204	D# or Eb
47	217	D
48	230	C# or Db
49	243	C
50	255	B

```
0 REM SOUND.6 (c) 1981 by Jerry White 11/9/81
10 V0=0:V1=1:V2=2:V3=3:POKE 82,2:POKE 83,39:GOTO 2000
50 POKE 540,WAIT
52 IF PEEK(540)<>0 THEN 52
54 FOR OFF=0 TO 3:SOUND OFF,0,0,0:NEXT OFF:RETURN
89 REM
90 REM *** MACHINE GUN ***
91 REM
100 FOR SHOT=1 TO 12:FOR VOL=15 TO 0 STEP -5:SOUND V0,80,0,VOL:SOUND V1,60,0,VOL
110 SOUND V2,200,4,VOL:SOUND V3,10,4,VOL:NEXT VOL:GOSUB 54:NEXT SHOT
120 RETURN
189 REM
190 REM *** SURF/WAVES ***
191 REM
200 FOR P0=10 TO 2 STEP -0.02:VOL=P0/2:SOUND V0,P0,8,VOL:SOUND V1,P0+1,8,VOL
210 SOUND V2,P0+2,8,VOL:SOUND V3,RND(0)*3,8,VOL
220 FOR P0=3 TO 12 STEP 0.02:VOL=P0/2:SOUND V0,P0,8,VOL:SOUND V1,P0+1,8,VOL
230 SOUND V2,P0+2,8,VOL:SOUND V3,RND(0)*3,8,VOL:NEXT P0
240 FOR P0=10 TO 2 STEP -0.02:VOL=P0/2:SOUND V0,P0,8,VOL:SOUND V1,P0+1,8,VOL
250 SOUND V2,P0+2,8,VOL:SOUND V3,RND(0)*3,8,VOL:NEXT P0:GOSUB 54:RETURN
289 REM
290 REM *** LAZERS/PHOTONS ***
291 REM
300 FOR SHOT=1 TO 6:FOR P0=0 TO 200 STEP 10
310 SOUND V0,P0,0,8:SOUND V1,P0,10,8:SOUND V2,P0,12,8:SOUND V3,P0,4,8
320 NEXT P0:NEXT SHOT:GOSUB 54:RETURN
389 REM
390 REM *** POLICE/FIRE SIREN ***
391 REM
400 FOR P0=200 TO 50 STEP -1:SOUND V0,P0,10,8:SOUND V1,P0+2,10,6:SOUND V2,P0+4,10,2:SOUND V3,P0+6,10,2:NEXT P0
420 FOR P0=50 TO 160 STEP 0.2:SOUND V0,P0,10,8:SOUND V1,P0+2,10,6:SOUND V2,P0+4,10,4:SOUND V3,P0+6,10,2:NEXT P0
430 GOSUB 54:RETURN
489 REM
490 REM *** AIR RAID SIREN ***
491 REM
500 FOR LOOP=1 TO 6:FOR P0=1 TO 20:SOUND V0,80+P0,12,8:NEXT P0
510 SOUND V0,80,10,12:SOUND V1,100,10,12:SOUND V2,13,4,12
520 WAIT=30:GOSUB 50:NEXT LOOP
530 FOR V=12 TO 0 STEP -0.1:SOUND V0,(20-V)*10,10,V:SOUND V1,(20-V)*10+20,10,V:SOUND V2,13,4,V:NEXT V
540 GOSUB 54:RETURN
589 REM
590 REM *** TELEPHONE RINGING ***
591 REM
600 FOR RING=1 TO 2:FOR LOUD=1 TO 35:SOUND V0,20,10,8:SOUND V1,1,2,8
610 FOR LOOP=1 TO 2:SOUND V0,25,10,8:SOUND V1,0,2,8:NEXT LOOP:SOUND V0,0,0,0:SOUND V1,0,0,0:NEXT LOUD
620 FOR V=7 TO 0 STEP -0.2:SOUND V0,20,10,V:SOUND V1,0,2,V:NEXT V
630 WAIT=90:GOSUB 50:NEXT RING:GOSUB 54:RETURN
689 REM
690 REM *** WHISTLING BOMB ***
691 REM
700 FOR P0=0 TO 150:SOUND 0,P0,10,P0/15+2:NEXT P0
710 FOR P0=0 TO 240 STEP 5:VOL=14-P0/20:SOUND V0,P0,0,VOL:SOUND V1,P0,8,VOL
720 SOUND V2,P0+15,2,VOL:NEXT P0:GOSUB 54:RETURN
789 REM
790 REM *** SPACE SHIP ***
791 REM
800 SOUND V2,0,8,2:FOR VOL=1 TO 15 STEP 0.1:SOUND V0,25,4,VOL:SOUND V1,13,4,VOL:NEXT VOL
810 FOR VOL=14 TO 0 STEP -0.1:SOUND V0,25,4,VOL:SOUND V1,13,4,VOL:NEXT VOL
820 GOSUB 54:RETURN
889 REM
```

SOUND.7

If you have run the first 6 programs in this package, you should now have a good background for using the Atari Basic SOUND command. The SOUND.5 program gave you some idea of what you can do by quickly changing the values of SOUND command variables. The SOUND.6 program demonstrated sound effects and hopefully you are now thinking about creating some of your own sound effect routines.

This machine has truly amazing sound capabilities. Believe it or not, the SOUND command is not the best way to create sound effects. Using machine language speed is one way to get sounds that Basic is just too slow to create. But you don't need machine language to get more sounds from this computer.

Some truly amazing sounds can be created using the POKE command. SOUND.7 lets you experiment with POKEs by using your joystick plugged into the first port.

I'll give you more information on these POKE locations later but first let me explain how to use the SOUND.7 program. You can read the technical stuff later.

The numbers 53760 thru 53768 are displayed on the screen. A greater than (>) symbol should appear next to the top number. Move that > down to the number 53768 by pulling back on the joystick. This is how we select the location we wish to POKE. Once the > points to the desired location, press the trigger button. The location number should turn blue.

Now push up on the joystick to make the number to be poked into location 53768,80. Push up to make the number higher and pull down to make the number lower. Once you have the number 80 next to the 53768, press the trigger button. This will cause the 53768 to go back to it's original yellow color and the > to return to it's original position next to the 53760.

No sound? Not yet, be patient. Now lets change the POKE for location 53760. Since the > is already in position, just press the trigger and that top number should turn blue. Use your joystick to make the poke 53760,10, then press the trigger.

Still no sound? Boy are you impatient. O.K. Move the > to location 53761 and press the trigger. Now change the poke to that location randomly. SOUND!!! It's about time. Now it's time to experiment. That location at the screen bottom is the key. Try changing it to 83, 85, and other numbers between 0 and 255. Then go back and change those top two locations.

What about 53762 thru 53767? Go ahead, do what you want. You can't hurt anything but you might drive some people crazy if the volume on your TV is too high.

To exit this program, just press any key on the keyboard. This is also a good way to shut off the sounds and start over by giving the RUN command.

Before using these POKE locations in Basic, there are two things you must know. The POKEY chip must first be initialized. This is accomplished with a simple SOUND 0,0,0,0 command and should be done at the beginning of your program.

These locations or sound registers are write only. For sound, you can POKE into these locations but PEEKing will not reflect the value you just poked. Confusing? You've got that right! The main thing is that you can create sounds using experimental POKEs without understanding what's happening or why.

For a detailed more explanation of these registers and other SOUND information, I'd strongly suggest you read the SOUND chapter in De Re Atari. This book is scheduled to be available from APEX.

LOCATION	DESCRIPTION
53760	Voice 0 Frequency
53761	Voice 0 Control
53762	Voice 1 Frequency
53763	Voice 1 Control
53764	Voice 2 Frequency
53765	Voice 2 Control
53766	Voice 3 Frequency
53767	Voice 3 Control
53768	Audio Control

```
890 REM *** SPACE ECHO ***
891 REM
900 FOR VOL=15 TO 0 STEP -0.2:FOR P0=0 TO 5:SOUND V0,P0,2,VOL:SOUND V1,P0+1,2,VOL:NEXT P0
910 FOR P1=VOL*10 TO VOL STEP -10:SOUND V0,P1,10,VOL:SOUND V1,P1+VOL,10,VOL:NEXT P1:NEXT VOL
920 RETURN
989 REM
990 REM *** DOOR BELL ***
991 REM
1000 FOR VOL=15 TO 0 STEP -0.5:SOUND V0,29,10,VOL:SOUND V1,30,10,VOL:NEXT VOL
1010 FOR VOL=15 TO 0 STEP -0.5:SOUND V0,35,10,VOL:SOUND V1,36,10,VOL:NEXT VOL
1020 RETURN
1999 STOP
2000 REM *** MENU OPTIONS ***
2010 GRAPHICS 0:SETCOLOR 2,15,0:POKE 752,1:POKE 201,10:?
2020 ? ,"QRRRRRRRRRRRRRRRE"
2030 ? ,"| SOUND EFFECTS |"
2040 ? ,"ZRRRRRRRRRRRRRRRC":POKE 201,9:?
2100 ? ," <1> MACHINE GUN"
2110 ? ," <2> SURF WAVES"
2120 ? ," <3> LAZER FIRE"
2130 ? ," <4> POLICE SIREN"
2140 ? ," <5> AIR RAID SIREN"
2150 ? ," <6> TELEPHONE RINGING"
2160 ? ," <7> WHISTLING BOMB"
2170 ? ," <8> SPACE SHIP"
2180 ? ," <9> SPACE ECHO"
2190 ? ,"<10> DOOR BELL"
2500 ? ,"<11> RUN LAST PROGRAM"
3000 POKE 53279,0:? :? ,"     CHOICE";:TRAP 9000:INPUT CHOICE:TRAP 40000
3010 CHOICE=INT(CHOICE):IF CHOICE<1 OR CHOICE>11 THEN 9000
3020 IF CHOICE=11 THEN RUN "D:SOUND.7"
3030 GRAPHICS 0:SETCOLOR 2,CHOICE,0:? :? :? :? :LIST CHOICE*100-10,CHOICE*100+80
3040 GOSUB CHOICE*100:? :? :? "     PRESS ANY KEY FOR OPTIONS";:POKE 764,255
3050 IF PEEK(764)=255 THEN 3050
3060 POKE 764,255:GOTO 2000
9000 RUN
```

```
*******************************
*** SOUND.7  (SOUND EDITOR) ***
*** (c) 1981 by Jerry White ***
*******************************
```

```
100    GOTO 10000
500    POSITION 1,0:
       FOR ME=0 TO 8:
           POSITION 6,ME+2:
           ? #6;53760+ME;"=";N(ME);"  ":
       NEXT ME
600    ? #6;"  select location  "
700    POSITION 5,2:
       ? #6;"1" :
       Y=2
720    S=STICK(0):
       IF S=14 AND Y>2 THEN
          POSITION 5,Y:
          ? #6;" ":
          Y=Y-1:
          POSITION 5,Y:
          ? #6;CHR$(30):
          GOSUB 3000
730    IF PEEK(764)<>255 THEN 12000
740    IF S=13 AND Y<10 THEN
          POSITION 5,Y:
          ? #6;" ":
          Y=Y+1:
          POSITION 5,Y:
          ? #6;CHR$(30):
          GOSUB 3000
760    IF STRIG(0)=1 THEN 720
800    GC=Y-2
900    POSITION 0,11:
       ? #6;"  release trigger  "
1000   IF STRIG(0)=0 THEN 1000
1010   POSITION 5,Y:
       ? #6;" "
1020   P=53760+GC:
       L$=STR$(P):
       FOR ME=1 TO 5:
           IT=ASC(L$(ME,ME)):
           IT=IT+128:
           L$(ME,ME)=CHR$(IT):
       NEXT ME
1030   POSITION 6,Y:
       ? #6;L$
1040   POSITION 0,11:
       ? #6;"    change poke  "
2000   S=STICK(0):
       IF S=14 AND N(GC)<255 THEN
          N(GC)=N(GC)+1
2020   IF PEEK(764)<>255 THEN 12000
2100   IF S=13 AND N(GC)>0 THEN
          N(GC)=N(GC)-1
2200   POKE P,N(GC):
       POSITION 12,Y:
       ? #6;N(GC);"  "
2300   IF STRIG(0)=1 THEN 2000
2400   POKE 65,0:
       GOTO 500
3000   FOR ME=1 TO 10:
           POKE 53279,0:
           POKE 53279,8:
       NEXT ME:
       RETURN
10000  GRAPHICS 18:
       DIM N(8),L$(5):
       SOUND 0,0,0,0:
       POKE 710,154:
       POKE 709,204
10100  ? #6;"    SOUND EDITOR":
       ? #6;"    ************":
       POKE 764,255
11000  FOR ME=0 TO 8:
           N(ME)=0:
       NEXT ME:
       CLOSE #1:
       OPEN #1,4,0,"K:":
       GOTO 500
12000  GRAPHICS 0:
       LIST :
       ? :
       ? "END OF SOUND TUTORIAL BY JERRY WHITE.":
       ? :
       END
```

Educational Software's
DISPLAY LISTS TRICKY TUTORIAL #1
FOR ATARI™ 16K COMPUTER
Use 'RUN:C' To Load
Educational Software's
HORZ/VERT SCROLL Tricky Tutorial #2
FOR ATARI™ 16K COMPUTER
Use 'RUN:C' To Load
Educational Software's
PAGE FLIPPING TRICKY TUTORIAL #3
FOR ATARI™ 16K COMPUTER
Use 'RUN:C' To Load
Educational Software's
BASICS OF ANIMATION Tricky Tutor #4
FOR ATARI™ 16K COMPUTER
Use 'RUN:C' To Load
Educational Software's
PLAYER MISSILE GRAPHICS T.T. #5
For Atari™ 32K Computer
Use 'RUN:C' To Load
Educational Software's
SOUND TRICKY TUTORIAL #6
FOR ATARI™ 16K COMPUTER
Use 'RUN:C' To Load