VOLUME 1 ISSUE 5 JANUARY 1985

SUPER 99 MONTHLY

BASIC

Program Development: Games and More

STANDARD: 1A 2A (o) 9A

Games (and many other applications) often require special programming techniques. Action games have one primary objective -- speed. All of the "fat" must be trimmed from the program to gain speed. Every BASIC statement consumes precious time and must be evaluated for its necessity. So, let's examine some ways to improve speed and eliminate unnecessary steps.

It will usually be advantageous to write a preliminary program, then a final version. In fact, you may even want several versions leading to a final program. In the first version, the program can include plenty of REM statements to clarify the steps of the program. In Extended BASIC, avoiding multiple statement lines will make the program easier to read. Once a skeleton program is completed, move to a new version in which the REM's are removed and you move towards trimming out the extra steps. If you use an intermediate version, try to avoid RESEQUENCE'ing the program, so that the intermediate version will still relate to the REM's in the first version. In the final version, use every trick you know to have a sleek. efficient program. By maintaining several versions, you can also revert to a previous version if you find that your revisions are not working as you anticipated.

Timing is crucial in writing a good game. You can time a statement by setting it up in a FOR-NEXT loop that is looped many times, say 1000 times. Subtract the time the loop requires to determine how long the statement takes. You will find that some statements, such as DEF, may be efficient in bytes, but may be very slow to execute. Avoid such statements in writing games when possible.

There are a number of tricks that are allowed in BASIC that the TI manuals either do not mention or do not emphasize. For instance, to place a border around the screen that is two characters wide, a beginner is likely to read the manuals and then use eight statements, one for each row and column. Only two statements are needed:

```
>100 CALL HCHAR(23,1,48,128)
>110 CALL VCHAR(31,3,48,94)
```

When using other than console BASIC alone (E/A, MMM, CorComp Toolshed, etc.), redefining character 31, the

-->

edge character, will provide a fast edge border in some situations.

When using the CALL statements, many CALL's allow multiple sets of parameters in Extended BASIC, saving bytes and execution time, though the program may become difficult to read and debug (thus the need for several versions of a program). Here is an example of redifining the shapes of two characters in one statement:

```
>100 CALL CHAR(128,"FF",129,"
 FE716024AAC0C0CF")
```

Note that in redefining character 128, the last 14 hex digits were omitted. BASIC will fill in the "0"'s that are implied by the statement. All trailing "0"'s are filled in. At least one hex digit must be used. There is no provision for filling leading "0"'s. By letting BASIC fill trailing "0"'s, bytes are saved.

Advanced programmers who are using other than console BASIC only can peek their program to see the "tokens" used by BASIC. In doing so, you may find some cases where two statements that accomplish the same task do not use the same number of bytes. For an example, refer to our September article on using ERASE ALL. In creating programs intended for sale, you may want to develop the program outside the console BASIC environment to have access to the tokens to determine what statements are consuming the most bytes.

Variable usage is also very important in writing action games. Both common sense and actual study reveals that extra bytes mean extra time. In your first version of a program, you may want to use very descriptive variable names. In later versions, trim the length of the variable name to as short a name as is possible, saving bytes and execution time.

Another aspect to the use of variables is the number of bytes required to store the values. When a value is to be used more than once, bytes will be saved by storing the value as a variable. The variable will use less bytes in the program than the value itself. Look for other ways to save bytes in using variables, such as our November example of peeking and poking bits from Extended BASIC.

Avoid duplication of effort. This rule is especially important in using variables and in defining characters. If three characters are to be defined as completely solid characters (16 hex "F"'s), just use one character if they are all the same color or use one variable to insert in the CALL CHAR if the three are to be different colors. You will find that in conserving characters, there will be cases where a screen created by a BASIC general screen design program, using the extra characters not allowed in Extended BASIC, may later actually be transferred to Extended BASIC by not duplicating the character definitions. As for variables, if A=100, then do not set B=100 unless the two will have unequal values -- it is a waste of bytes to duplicate such efforts (again, this makes your program much more difficult to read, so use our multi-version approach).

All forms of BASIC will naturally require a lot of time for some operations. Sorting or random selections are examples of slow BASIC processes. Most games will have at least a few points at which the program pauses, such as to enter the names of players, etc. Make good use of these pause points by setting up subsequent steps at the pause or before the first screen display. For instance, if you are depicting some type of race that is independent of player response, establishing the screen locations for all character movements before the race will make the race move much faster than if you calculate the next screen position during the race, although more bytes would be used in this case.

When possible, make your program upwards compatible. For example, make BASIC programs capable of being run in -->

Extended BASIC. Use characters above 143 only when absolutely necessary. To scroll the screen, separate colons following a PRINT statement with blanks so that Extended BASIC will not read them as double colons, which are multi-statement line separators. Leave about 900 bytes free in console BASIC as Extended BASIC has 864 fewer bytes available. If the 900 bytes presents a problem, some variable values can probably be inserted in your program by reading a file, which will often redistribute the bytes in memory so the program may run without an out of memory error. Additionally, disk drive buffers require up to about 2K. To be sure your program will run in any BASIC environment, allow about 3K for overhead. To determine the size of your console BASIC program, add the following lines at the beginning of your program:

```
>1 A=A+8
>2 GOSUB 1
```

Type RUN. After a few seconds, you will get a "Memory Full in 1" error. Key PRINT A. A is the number of bytes free (though not quite exact -- it can be up to 7 bytes off).

There are hundreds of ways to save bytes, but the best way is always to plan your program. We've seen many beginning programmers spend days trimming a byte at a time when proper planning would have saved hundreds or even thousands of bytes before the first line was ever written. Look for saving the large segments of memory first. Failing to recognize where a loop could be used can waste large areas of memory very quickly. This explains why we have dealt so much with saving the large areas of memory in previous issues. Once a program has been planned, written and has undergone preliminary tests, fine-tuning by trimming a few extra bytes out and adding user-friendly techniques comes into play. In case you are not familiar with user-friendly programming, we'll cover that topic in an upcoming issue.

Solitaire Checkers

STANDARD: 1A 9A

The program below is a version of the popular game commonly sold at tourist stops. Just as in checkers, the object is to jump diagonally until as few as possible checkers remain.

The program is an example of a program that uses minimal bytes while remaining fast, user-friendly and properly structured. Though of no particular significance, the program was written without using string variables, despite the use of extensive screen displays.

Be sure to press <ENTER> after your input when playing the game.

```
>100 GOSUB 1000
>110 GOSUB 2000
>120 GOSUB 3000
>130 IF Q<>161 THEN 110
>140 PRINT M;"MOVES"
>999 END
>1000 DIM C(3)
>1010 DIM Z(64)
>1020 CALL CLEAR
>1030 CALL CHAR(128,"FFFFFFFF
 FFFFFFFF")
>1040 CALL CHAR(136,"FF818181
 818181FF")
>1050 FOR I=8 TO 22 STEP 2
>1060 FOR J=2 TO 16 STEP 2
>1070 CALL HCHAR(J,I,128)
>1080 Z((J/2)*8-8+(I-6)/2)=1
>1090 NEXT J
>1100 NEXT I
>1110 FOR I=12 TO 18 STEP 2
>1120 FOR J=6 TO 12 STEP 2
>1130 CALL HCHAR(J,I,136)
>1140 Z((J/2)*8-8+(I-6)/2)=0
>1150 NEXT J
>1160 NEXT I
>1170 FOR I=8 TO 22
>1180 CALL HCHAR(1,I,(I-6)/2+
 48)
>1190 NEXT I
>1200 FOR I=2 TO 16 STEP 2
>1210 CALL VCHAR(I,7,I/2+64)
>1220 NEXT I
>1230 FOR J=1 TO 4
>1240 GOSUB 1310
>1250 NEXT J
```

-->

```
>1260 M=0
>1270 RETURN
>1280 DATA 7,10,18,70,82,79,7
 7,22,23,18,84,79
>1290 DATA 3,30,21,69,78,84,6
 9,82,32,39,70,82,79,77,39,32
 ,65,83,32,39,81,81,39,32,84,
 79,32,81,85,73,84
>1300 DATA 3,26,22,80,82,69,8
 3,83,32,39,69,82,65,83,69,39
 ,32,84,79,32,67,79,82,82,69,
 67,84
>1310 READ C1,C2,R
>1320 FOR I=C1 TO C2
>1330 READ K
>1340 CALL HCHAR(R,I,K)
>1350 NEXT I
>1360 RETURN
>2000 RESTORE 2210
>2010 FOR I=1 TO 2
>2020 READ C1,C2
>2030 C(1)=128
>2040 C(2)=32
>2050 J=1
>2060 IF J<>2 THEN 2080
>2070 C(2)=128
>2080 CALL HCHAR(19,C1,C(1))
>2090 CALL HCHAR(19,C2,C(2))
>2100 CALL KEY(5,K,S)
>2110 IF S<1 THEN 2100
>2120 IF (K<48)*(J<3)*(K<>7)T
 HEN 2100
>2130 C(J)=K
>2140 IF K=7 THEN 2030
>2150 IF J<3 THEN 2170
>2160 IF K<>13 THEN 2100
>2170 J=J+1
>2180 IF J<4 THEN 2060
>2190 NEXT I
>2200 RETURN
>2210 DATA 8,9,22,23
>3000 CALL GCHAR(19,8,F1)
>3010 CALL GCHAR(19,9,F2)
>3020 F=(F1-64)*8-8+F2-48
>3030 Q=F
>3040 IF Q=161 THEN 3220
>3050 CALL GCHAR(19,22,T1)
>3060 CALL GCHAR(19,23,T2)
>3070 T=((T1-64)*8)-8+T2-48
>3080 F1=INT((F1-1)/8)+1
>3090 F2=F-8*(F1-1)+1
>3100 T1=INT((T-1)/8)+1
>3110 T2=T-8*(T1-1)+1
>3120 IF (F1>8)+(T1>8)+(F2>9)
 +(T2>9)+(Z((T+F)/2)=0)+(Z(F)
 =0)+(Z(T)=1)THEN 3230
>3130 IF ABS(F-T)=14 THEN 315
 0
>3140 IF ABS(F-T)<>18 THEN 32
 30
>3150 Z(T)=1
>3160 CALL HCHAR(T1*2,T2*2+4,
 128)
>3170 Z(F)=0
>3180 CALL HCHAR(F1*2,F2*2+4,
 136)
>3190 Z((T+F)/2)=0
>3200 CALL HCHAR(((T1+F1)/2)*
 2,((T2+F2)/2)*2+4,136)
>3210 M=M+1
>3220 RETURN
>3230 RESTORE 3320
>3240 FOR I=3 TO 14
>3250 READ R
>3260 CALL HCHAR(24,I,R)
>3270 NEXT I
>3280 FOR I=1 TO 400
>3290 NEXT I
>3300 CALL HCHAR(24,3,32,12)
>3310 RETURN
>3320 DATA 73,76,76,69,71,65,
 76,32,77,79,86,69
```

Extending a Line

STANDARD: 1A 2A(o) 9A

Here is yet another tip from Jim Peterson, Tigercub Software, 156 Collingwood Ave., Columbus, OH 43213.

Many bytes can be saved by putting as much on one BASIC or Extended BASIC line as possible. One way to be sure to fit more on one line is to make the line longer!

The easiest way to extend a line in BASIC is to fill all 4 lines completely, enter it, then call it back by typing the line number and <FCTN> <X>, then you can run the cursor onto the 5th line.

In Extended BASIC, if you type multiple statements without spaces before and after the "::", the computer will fill in the spaces and automatically go into the next line. Or, just enter the line and hit <FCTN> <8> to bring it back.

EDITOR'S NOTE: When Jim issues a challenge to accomplish a task in one -->

line, he may mean one of these extended lines. He sure caught me on that one and we had a big laugh about it! - Richard Mitchell

---

EXTENDED BASIC

"Mouse"-ing Around in Extended BASIC

STANDARD: 1A 2A 9A 11A

Most of you have probably seen the "mouse" devices available for many computers. They are really just a fancy joystick, so let's see what we can do with a joystick on our TI Home Computers.

One use is for demonstrations, such as for retail sales, user group meetings and business meetings. If you are demonstrating a series of screens, you can stand back a few feet, so that you do not block anyone's view, and change screens as if you were operating a slide show. To do this, use the usual CALL KEY statement, making sure to use 1 and 2 in the first parameter to designate the joysticks:

```
>100 CALL KEY(1,K,S):: KE=S
 :: CALL KEY(2,K,S):: KE=KE+S
 :: IF KE<1 THEN 100
```

The above line will await the pressing of the fire button on either joystick or any key.

Another idea is to use the joystick to make menu selections. Below is an example program. Note that sprites can go 8 rows "below" the bottom of the screen, so the last data in line 120 is 32, which represents "row 32". Also note that the MOUSE subprogram can detect whatever rows are established by the data statements in line 120 in case you need a different number of menu options or you wish to relocate the menu on the screen.

```
>100 CALL MENUSCREEN
>110 RESTORE 120 :: CALL MOUS
 E(FLAG)
>120 DATA 5,7,9,32
>130 CALL CLEAR :: CALL DELSP
 RITE(ALL):: ON FLAG GOSUB 10
 00,2000,3000,999
>140 FOR DELAY=1 TO 200 :: NE
 XT DELAY :: GOTO 100
>999 END
>1000 PRINT "LINE 1000" :: RE
 TURN
>2000 PRINT "LINE 2000" :: RE
 TURN
>3000 PRINT "LINE 3000" :: RE
 TURN
>20000 SUB MENUSCREEN
>20010 CALL CLEAR :: CALL CHA
 R(128,"00FFE0F0E0F0E0F")
>20020 RESTORE 20070
>20030 FOR I=4 TO 10 STEP 2
>20040 READ L$
>20050 DISPLAY AT(I,5):L$
>20060 NEXT I :: SUBEXIT
>20070 DATA "1. LINE 1000"
>20080 DATA "2. LINE 2000"
>20090 DATA "3. LINE 3000"
>20100 DATA "4. END"
>20110 SUBEND
>21000 SUB MOUSE(FLAG)
>21010 CALL SPRITE(#1,128,2,2
 8,40)
>21020 CALL JOYST(1,X1,Y1)::
   CALL JOYST(2,X2,Y2)
>21030 CALL MOTION(#1,-6*(Y1+
 Y2),0)
>21040 CALL KEY(1,K,S):: IF K
 =18 THEN CALL POSITION(#1,X,
 Y)ELSE CALL KEY(2,K,S):: IF
 K=18 THEN CALL POSITION(#1,X
 ,Y)ELSE 21020
>21050 FLAG=INT((X+7)/8)
>21060 I=1
>21070 READ MARKER
>21080 IF FLAG<=MARKER THEN F
 LAG=I :: SUBEXIT :: ELSE I=I
 +1 :: GOTO 21070
>21090 SUBEND
```

This example shows the basics of how you can simulate a "mouse". You could also do graphics designs and many other activities with a joystick. Let us know what applications you have come up with.

-->

Speech On?

STANDARD: 1A 2A 5A 7A 9A 12A

Many of you are probably already familiar with the address to peek to determine whether the speech synthesizer is attached. The earliest source for the information that we have been able to trace was Miller's Graphics in late 1983. However, we have discovered that a few consoles return different values than those previously mentioned in print.

Each console has an LTA number on its underside, indicating the period in which the console was manufactured. Several small differences in the various models are known to exist. Before you panic, we assure you that most of these differences do not affect most programs at all.

The address to peek to determine whether speech is on is -28672. The values usually returned are 96 for speech attached and 0 for speech not attached. Consoles with number LTA4583 and possibly other versions return 255 for attached and 127 for not attached. Therefore, the proper Extended BASIC statement to test for the synthesizer follows (NOTE: Use CALL INIT only if you have not already done so in your program):

```
>100 CALL INIT
>110 CALL SPEECHTEST(FLAG)
>120 IF FLAG=1 THEN CALL SAY(
 "VERY GOOD")
>999 END
>20000 SUB SPEECHTEST(FLAG)
>20010 CALL PEEK(-28672,A)::
 IF (A=96)+(A=255)THEN FLAG=1
  ELSE FLAG=0
>20020 SUBEND
```

Note that we are testing for whether speech is attached. Testing for speech not being attached would still cause a system lockup (which is what happens if you attempt to access speech without it being attached) if still more valid values exist. This is a very good example of a case where it makes a lot of difference whether you test for true or false.

Please note that we checked several configurations to be positive that this difference is in the console.

ASSEMBLY

More on Screen Buffers

STANDARD: 1A 2AC 4B 5A 6B 7A 9A

Last month we covered building screen buffers using a memory image file that could be accessed from Extended BASIC and displayed using the CorComp Disk Controller Toolshed statements. This month we have some more thoughts on the subject.

First, we hope everyone received the addendum that we inserted in last month's mailing. It corrected a few errors that we caught at the last moment before mailing. On page 8, in the section with label DSRLNK, after MOV *R14+,R5 add a line for MOV R0,R9. In the Extended BASIC test for the Assembly program, insert an asterisk before the 8 in lines 140, 200 and 260.

Just before going to press with this issue, we received a question from a reader regarding the file construction on this project. The 7 screens of 768 bytes require 21 sectors. Our file requires 22 sectors. In order to get down to 21 sectors, the file must be reduced by one sector, changing the source code under PDATA, CPUDSK, and DSKCPU from >1500 to >1400. The other sector used is for file overhead. Next month we'll try to cover in detail the construction of this and some other files on your disk, explaining just what makes up the file overhead of the files. We hope this explanation will do for now.

We now have a demonstration disk showing what can be accomplished with our screen buffer techniques. It shows several screen, character set and color changes. The demo is set up to change screens by either pressing

-->

the fire button on a joystick or by pressing any key. Subscribers may obtain the program by sending an initialized disk, disk mailer, return label, return postage, and $2.00 to cover handling. There is no charge for the programs. In appreciation of the support shown us by user groups, non-profit user groups are not subject to the $2.00 handling fee. If your group has restrictions on meeting presentations, you should be aware that the demo does reference CorComp and Super 99 Monthly.

TI-WRITER

Accessing Files From Extended BASIC

STANDARD: 1A 2ACE 4B 5A 6B 7A 9A

In October we told how to create a dual-column right-adjusted TI-Writer document. The primary problem with the method was that it was not very friendly when a lot of changes were necessary. The solution seemed to be to manipulate the sequence of the file records in Extended BASIC, at the same time doing some of the repetitious formatting required. However, page 157 of the TI-Writer manual states that accessing TI-Writer files from Extended BASIC is not possible. A file can be created in Extended BASIC and used in TI-Writer, but the reverse was said to not be feasible.

WARNING: THE OPERATIONS DESCRIBED IN THIS ARTICLE PROVIDE A POSSIBILITY OF FILE ERRORS. USE YOUR DATA DISKETTE FOR ONLY YOUR ORIGINAL FILE AND YOUR FINAL FILE. ALL INTERMEDIATE FILES SHOULD BE CREATED ON A NEWLY INITIALIZED DISK AS IT IS EASY TO CREATE A FILE WITHOUT HAVING IT OPEN OR CLOSE PROPERLY.

However, there is a fairly simple way to access TI-Writer files from Extended BASIC. First, you must load the file into the Editor/Assembler, as if it were an Assembly source document to be edited. When the document loads, an error message may occur stating that control characters have been lost. You may need to work with this for awhile to see if the loss of control characters has an effect on what you are doing. We have had no problems with our projects. Next, instead of selecting EDIT, select SAVE. You will be prompted "DISPLAY/80 FORMAT (Y/N)?". Answer "N". This will give you a file in DIS/FIX 80 format, which you should note is usually a much longer file. The DIS/FIX 80 format is a RELATIVE format. Be sure to SAVE the file with a new name. Unlike the DIS/VAR 80 format of TI-Writer, the DIS/FIX 80 format is easy to work with in Extended BASIC.

In Extended BASIC, the DIS/FIX 80 format can be used as a RELATIVE file. Here is an example of how to get started in accessing the file:

```
>100 OPEN #1:"DSK1.TIW",DISPL
 AY ,RELATIVE,FIXED 80
>110 LINPUT #1,REC 1:A$
>120 LINPUT #1,REC 35:B$
>130 PRINT #1,REC 1:B$
>140 PRINT #1,REC 35:A$
>150 CLOSE #1
```

The example above will move line 35 of the TI-Writer file to line 1 and line 1 to line 35. Obviously, this type of access could be used to manipulate many records and would be especially useful if there was a pattern whereby loops could be used in the Extended BASIC program. In some cases, more than one file would be necessary for manipulating the lines.

Back to our October discussion of dual-column right-adjustment, you can build your TI-Writer file at single-column width, move the file to Extended BASIC, then use SEG$ and concatenate to build the file as we had described in October. This virtually eliminates the problem we mentioned in October of difficulties in making massive changes to a document.

Once the file is changed in Extended BASIC, return to the Editor/Assembler and LOAD and SAVE again, this time using the DIS/VAR 80 format

-->

so the file can be loaded back into TI-Writer. To be sure the file is ready for TEXT FORMATTER, save the file again after it has been loaded into TI-Writer. Also, add any special techniques that you might find difficult to carry over from Extended BASIC.

There are other tricks in using TI-Writer files. For instance, a file does not have to be printed to a printer. It can be "printed" from TEXT FORMATTER to disk if you find a use for such manipulations. This same principle is true in many TI Modules. For instance, in using Editor/ Assembler, instead of printing a LIST, direct the "print" to disk. This will allow you to try the program and print the listing only if there are bugs in the program, if that is your desire.

We will be covering a number of other uses for accessing TI-Writer files from Extended BASIC in upcoming months.

CORCOMP CONTROLLER

Avoiding Some Problems

STANDARD: 1A 2 (various) 4B 5A 6B (no) 7A 9A

The CorComp 9900 Disk Controller is substantially free of bugs. However, there is one peculiar problem that is easily solved.

Instead of the regular TI title screen on power-up, the CorComp Card yields a special menu selection screen, which is similar to the screen that is normally seen after the TI title screen. Selecting from the CorComp menu sometimes gives problems. To be sure to avoid these problems, press the space bar to get the color bar screen, then press the space bar again to get the regular TI menu.

Problems have been noted in attempting to use Plato and in trying to OPEN a file for the Speech Synthesizer from the Terminal Emulator II if the selections are made from the CorComp menu screen. Using the TI menu prevents such problems. You may want to make a habit of always using the TI menu for cartridges that you are not sure about. We are not aware of a fix yet by CorComp. If you have knowledge of a fix, please write to let us know.

MULTIPLAN™

Multiplan™ vs. Extended BASIC

STANDARD: 1A 2D 4B 5A 6B 7A 9A

One of the first things to be noticed in using Multiplan™ is that inputs are painfully slow when compared to Extended BASIC. So, why use Multiplan™?

Multiplan™ is a spreadsheet program. What does that really mean? First, it means that nearly all of the programming has already been done -- one need only fill in the numbers, names, formulae, etc. Next, it is a spreadsheet, which means that all inputs are automatically placed into a format.

In analyzing whether to write an Extended BASIC program or use Multiplan™, one must consider the following:

1. Will the program be used more than once? If not, Multiplan™ will likely be the best choice.

2. Will it be difficult to write a program for the amount of print formatting involved? It is very easy to generate a report that is perfectly formatted when using Multiplan™. If very much print formatting is involved, consider using Multiplan™.

3. Are you a very good programmer? If not, use Multiplan™ whenever possible.

4. How much data manipulation is involved? Sorts, comparisons and -->

complex cross-references are not easy to program. Consider using Multiplan™.

5. Do you like to be able to reformat your printouts? If so, definitely use Multiplan™. Changing a print layout in a program can be very involved.

6. Can you key input quickly? If not, you may never notice that the major drawback to Multiplan™ is how slowly the input is accomplished.

In short, if you are confident you can write an Extended BASIC program in what you consider a reasonable time to accomplish your task satisfactorily, by all means do so -- it should run much faster than Multiplan™, especially if you know how to make use of Assembly sorts, etc. Otherwise, use Multiplan™ and save yourself a lot of time and frustration.

Of course, there is one other alternative. Do your input in Extended BASIC and store the data in a file that will link to Multiplan™. This is not a beginner's project and will be a topic for a future article.

LOGO

LOGO -- The Challenge to Explore

STANDARD: 1A 2F 4B(o) 5A 6B(o) 7A 9A

With most computer languages, the objective is to cause the computer to do something significant. That is why LOGO presents a problem to experienced computer users -- the definition of significant must be revised. LOGO is oriented toward the user instead of the computer. It is very effective when used as a step-by-step learning experience or for recreation. In attempting to force LOGO away from creative processes, one is likely to bog down in a futile effort. LOGO challenges you to explore.

That is why we've chosen a very simple project for this month. It moves sprites on the screen. But try looking at the pattern of the movement and think about why it appears the way it does. If you begin to contemplate why the pattern repeats when it does, then you are on the path to understanding the power of LOGO.

```
MAKE "FLAKE 7
MAKESHAPE :FLAKE
```

```
X O O O O O O X X O O O O O O X
O X O O O O O O O O O O O O X O
O O X O O O O X X O O O O X O O
O O O X O O O O O O O O X O O O
O O O O O O O O O O O O O O O O
O O O O O X O O O O X O O O O O
O O O O O O O O O O O O O O O O
O O O O O O O X X O O O O O O O
O O O O O O O X X O O O O O O O
O O O O O O O O O O O O O O O O
O O O O X O O O O X O O O O O O
O O O O O O O O O O O O O O O O
O O O X O O O O O O O O X O O O
O O X O O O O O O O O O O X O O
O X O O O O O O O O O O O O X O
X O O O O O O O O O O O O O O X
```

```
TELL :ALL
SETSPEED 0
SETHEADING 0
HOME
CARRY :FLAKE
SETCOLOR :WHITE
EACH [SETSPEED 1 * YOURNUMBER ]
```

FORTH

Shooting on the Move Game

STANDARD: 1A 2C 4B 5A 6B 7A 9A

FORTH offers an excellent set of graphics and sprite commands. Most of the commands are very similar to Extended BASIC, but offer more flexibility. For instance, instead of working with 112 characters with 28 sprites, as in Extended BASIC, FORTH offers 255 characters and 32 sprites. To accomplish this, FORTH allows the Sprite Description Table to be easily segregated from the Pattern Description Table. In Extended BASIC, -->

the 28 sprites take on the shape of regular characters as the SDT and PDT are one in the same table. There are many other such advantages to using FORTH -- it is very flexible.

Of course, the biggest advantage to using FORTH for games is the speed of FORTH. Sprite coincidences can be checked so fast that a coincidence will seldom be missed. Entire screens can be set up very quickly, too.

The biggest disadvantage to using FORTH for games is the difficulty in slowing it down. A FORTH delay loop is sometimes one of the longest definitions in a program!

For the game we wrote, we decided that since FORTH uses so many reverse implementations, we'd write a reverse game as a symbolic gesture (pun intended). Instead of shooting a moving object, our game uses a moving "gun" and a still target.

Most of the definitions we've used are straight-forward from reading the TI FORTH manual, but a few special techniques were used. Note that ?FIRE tests both joysticks. The JOYST statement will return 255 as the third value on the stack if no key has been pressed. Therefore, we are looking for any value other than 255. This allows us to shoot by pressing any key or pressing the fire button on either joystick. To do this, we subtract 255, which is FF in hex, from the value on the stack after two DROP's. We then continue this test UNTIL neither key value is 255.

Since mixing HEX and DECIMAL sometimes gets very confusing, we have chosen to stay with HEX throughout the program. This is in keeping with the standard notations for CHAR statements.

The game is not intended to be one that will likely fascinate you for hours, but rather is just an example of the use of some of the graphics and sprite statements available in FORTH.

Here is the program listing:

```
SCR #122
 0 BASE->R ( S99M GAME ) GRAPHICS HEX
 1 0 VARIABLE SCORE
 2 : SET1 CLS
 3        FFFF 0000 0000 FFFF 80 CHAR
 4        FF00 0000 0000 0000 81 CHAR ;
 5 : SET2 0 5 20 80 HCHAR
 6         F 5 2  88 HCHAR 1 F 10 COLOR
 7         7 SCREEN ;
 8 : SP1 2000 SSDT
 9        8080 8080 FFFF FFFF 1 SPCHAR
10        0000 1010 0000 0000 0 SPCHAR
11        FF B0 1 1 1 SPRITE
12        C0 0 1 MOTION 0 0 0 MOTION
13        2 #MOTION
14        1 MAGNIFY ;
15 --> ( GO TO NEXT SCREEN )

SCR #123
 0 ( S99M GAME cont. )
 1 : CHECK1 1 JOYST DROP DROP FF - ;
 2 : CHECK2 2 JOYST DROP DROP FF - ;
 3 : ?FIRE BEGIN CHECK1 CHECK2 +
 4           UNTIL ;
 5 : SHOOT 1 #MOTION
 6         1 SPRGET SWAP 5 - SWAP
 7         1 0 0 SPRITE
 8         0 E5 0 MOTION ;
 9
10
11
12 : CHECK3 80 B0 1 SPRDISTXY
13           SCORE ! ;
14
15 -->

SCR #124
 0 ( S99M GAME cont. )
 1
 2
 3
 4 : GAME SET1 SET2 SP1 ?FIRE SHOOT
 5        CHECK3 09 03 GOTOXY
 6        ." SCORE = " SCORE ? ;
 7
 8 : RUN 9 0 DO GAME
 9                8 0 DO
10                     1000 0 DO LOOP
11                   LOOP
12            LOOP
13       ABORT ;
14
15 R->BASE
```

-->

99 POTPOURRI

News, Corrections, Updates, Editorials, Kudos, and Come-what-may

CORRECTIONS:

OCTOBER: The subscription price for The National Ninety-Niner, the publication of the 99er Users Group Association, was misquoted. The correct price is $12.00 per year. The Association is an excellent non-profit organization. If you remitted the amount we quoted, please remit the balance to the Association.

DECEMBER: See this month's Assembly article for corrections to the December Assembly article.

TIBBS™ should have been stated to be a Registered Trademark of Ralph Fowler.

Computer Show!

Apparently the show in the Chicago area in December was a big success, as there is already another one planned -- and soon!

The Computer Central Show will be Sunday, March 3, 1985, from 9:30 am to 4:00 pm at Rand Park Field House, 2025 Dempster/Miner, Des Plaines, IL. Admission will be $4, $2 for kids 8-15 and under 8 free. Information is available by phoning 1-312-940-7547. TI-99/4A products were available at the December show and are expected for the March show. This sounds like a great way to spend a weekend!

COMING IN FEBRUARY

Hardware Construction Project!
Great tips on TI-Writer and Multiplan™!
Much, much more!

Effective immediately, we are accepting applications for free advertisements for unique or difficult to obtain items and Freeware (pay what you think the software is worth). Only a very small portion of this section will be used for such ads. We make absolutely no guarantees relating to submission, publication, or the information in such ads. This will simply be a service to our readers.

St. Louis 99ers, a non-profit user group, published an outstanding article in their December newsletter. The article includes a program listing for using a true 40 column screen from Extended BASIC. A memory expansion card is required. To obtain a copy, send $1.00 for a back issue to St. Louis 99ers, 6112 Staely Ave., St. Louis, MO 63123. The article is credited to Roy T. Tamashiro. Good work, Mr. Tamashiro! Remember, rights to the article and the program belong to St. Louis 99ers.

We have received a number of complaints recently about the policies of some of the commercial "user groups" or "clubs". "Dues" does not entitle "members" to the right to vote, which makes the label "club" very misleading, to say the least. If you pay such "dues", your only assurance is for a smaller checkbook balance. Let the buyer beware!

We have confirmed that the Myarc RS-232 and Disk Controller cards do use TI clamshell cases. The primary advantage of such cases is that hinges are provided for grasping the card for removal.

Multiplan and Microsoft are Registered Trademarks of Microsoft Corporation.

-->

Super 99 Monthly is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663. Subscription rate in U.S. and possessions is $12.00 per year; all other countries $16.00 U.S. funds for surface mail. All correspondence received will be considered unconditionally assigned for publication and copyright and subject to editing and comments by the editors of Super 99 Monthly. Each contribution to this issue and the issue as a whole Copyright 1984 by Bytemaster Computer Services.

STANDARD KEY

1	Computer	A	TI-99/4A (LTA4583)
2	Cartridge	A	Extended BASIC
		C	Editor/Assembler
		D	Multiplan (TM)
		E	TI-Writer
		F	LOGO
4	Disk Drive	B	TEAC 55B
5	Expansion Box	A	TI
6	Disk Controller	B	CorComp
7	32K Card	A	TI
9	Monitor or TV	A	TV & RF Modulator
11	Joystick	A	Prostick II (TM)
12	Speech Synthesizer	A	TI

Prostick II is a trademark of Cal-Tron Corporation
Multiplan and Microsoft are trademarks of Microsoft Corporation

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663

FIRST CLASS MAIL

08/85

[redacted]