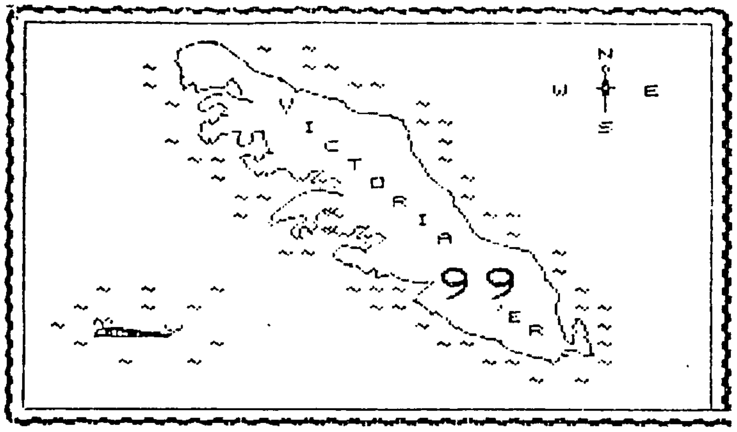

MAY 84

****
PROGRAM CONTEST:
A NEW PROGRAM CONTEST CONCEPT. THE PROGRAM MUST BE WRITTEN IN CONSOL BASIC AND BE NO LARGER THAN 20 PROGRAM LINES LONG. THE PROGRAMS MAY BE ON ANY SUBJECT AND CONTENT IS ENTIRELY UP TO YOU. PROGRAMS WILL BE JUDGED ON PERFORMANCE, ORIGINALITY, AND COMPACTNESS. PLEASE CONTACT ME FOR DETAILS.

I AM NOW EQUIPED WITH A MODEM AND RS232 CARD AND MAY BE REACHED AT 383_3946.

Tony Bigras

****
CHALLENGE : FOR EXTENDED BASIC USERS

THE TASK : CLEAR THE SCREEN AND DISPLAY AN X OF *'s ON THE SCREEN IN A 9 x 9 AREA.

RESTRICTIONS : YOU MUST USE ONLY 1 PROGRAM LINE, YOU MAY NOT USE EXTRA MEMORY IN ANY WAY.

THE VERSION WITH THE SMALLEST SIZE (after running) WILL BE JUDGED THE BEST.
I INVITE MEMBERS TO SUBMIT TASKS THAT THEY HAVE WRITTEN ON ONE LINE . THESE WILL BE PRESENTED AS A CHALLENGE IN FUTURE NEWS LETTERS.
BRING YOUR SOLUTIONS TO THE TASK TO THE NEXT MEETING.

****
IF YOU ARE INTERESTED IN THE ARTICLE IN THE CURRENT ISSUE OF SCIENTIFIC AMERICAN ON COMPUTER RECREATIONS A "MEMORY ARRAY REDCODE SIMULATOR" IS AVAIBLE FOR YOUR USE, JUST GIVE ME A CALL.

****

```
                    VI 99 MEMBERSHIP LIST

                       APRIL 15 1984


************************************************************
    STEVE McCUAIG                           *     DON & HAZEL TUFFORD
    3109 PARLOW PL                          *     313-4026 QUADRA ST
    VICTORIA BC                             *     VICTORIA BC  V8X 4E3
    478-1305                                *     479-9565
                                            *
************************************************************
    JOE.TERRY.RICK.CHRIS MACMURCHIE         *     BRIAN ATWELL
    4476 DENSMORE RD                        *     1615 HOLLYWOOD CRESCENT
    RR #3 VICTORIA BC V8X 3X1               *     VICTORIA BC V8S 1H8
    479-7605                                *     595-0096
                                            *
************************************************************
    TOM SWIRSKI                             *     ROBERT SORENSON
    1369 FINLAYSON ST                       *     3228 SERVICE ST
    VICTORIA BC V8T 2V5                     *     VICTORIA BC V8P 4M7
    384-4457                                *     592-6790
                                            *
************************************************************
    JOHAN VAN IMSCHOOT                      *     NICK SHEMDIN
    100 VIADUCT AVE WEST                    *     1631 GARNET RD
    VICTORIA BC V8X 3X3                     *     VICTORIA BC V8P 3C7
    479-7503                                *     721-2003
                                            *
************************************************************
    HERBERT A STARK                         *     STEVE HOLLAND
    188 ATKINS AVE                          *     2602 PEATT RD
    VICTORIA BC V9B 2Z8                     *     VICTORIA BC V9B 3T8
    ???????                                 *     474-1933
                                            *
************************************************************
    RUSS WATSON                             *     TED SIERMACHESKY & MARG SHARE
    4709 BARROW RD                          *     #2 2229 VICTOR ST
    RR #2 VICTORIA BC V9B 5B4               *     VICTORIA BC
    478-3495                                *     592-0612
                                            *
************************************************************
    TONY BIGRAS                             *     STEPHEN MILLS
    644 BELTON AVE                          *     690 MARLISA PL
    VICTORIA BC                             *     VICTORIA BC
    383-3946                                *     474-1947
                                            *
************************************************************
    RAY GALLAGHER                           *     RICHARD NICOLSON
    4290 HAPPY VALLEY RD                    *     1297 DERBY RD
    VICTORIA BC                             *     VICTORIA BC V8P 1S7
    478-7460                                *     384-0237
                                            *
************************************************************
    WALTER.LITA ROLOFS                      *     BARRY VIVIAN
    7912 PATTERSON RD                       *     8850 MORESBY PK TRCE
    SAANICHTON BC VOS 1MO                   *     SIDNEY BC V8L 4A9
    652-9455                                *     656-1727
                                            *
************************************************************
    MICHAEL EGBERTS                         *
    BOX 1784                                *
    12 ALBERTA DRIVE                        *
    MACKENZIE BC VOJ 2CO                    *
    997-3906                                *
```

BOB WILLLINHNGANZ	MIKE SEYMOUR
1039 ST. DAVID ST.	8515 BEXLEY TERC
VICTORIA BC V8S 4Y7	SIDNEY BC V8L 1M3
592-7798	656-3305
ALLEN B PAGE	STEVE RAYNER
139 ST LAWRENCE ST	341 BERWICK ST
VICTORIA BC V8V 1X9	VICTORIA BC V8V 1C8
382-3778	382-1828
JOHN STEIN	RICHARD McCREA
#309 628 DALLAS RD	216 885 DUNSMUIR RD
VICTORIA BC V8B 1B5	VICTORIA BC V9A 6W6
383-8382	386-1688
VIC WAGER	ALAN SORENSEN
4184 BUCKINGHAM PL	3228 SERVICE ST
VICTORIA	VICTORIA
477-8995	592-6790
DEREK HAMLET	JOHN WYNN
2373 CENTRAL AVE	
VICTORIA	
595-2569	

KEN ARMSTRONG
1176 BRIARWOOD DRIVE
RR#1 COBBLE HILL BC
V0R1L0

FORTH HINTS: BY JOHAN VAN IMSCHOOT

TI AND Wycove FORTH for the TI994/A:

Some hints for absolute beginners:

For TI Forth: Unless you know what you are doing, you are advised not to edit the system disk. At least, edit a copy, not your original. (Actually as there is at least one error on the system disk, it does need to be edited.)

Also, do NOT use disks which have non-Forth material on them, like Basic programs. The reason is that TI FORTH is capable of writing onto any formatted diskette it is given, it ignores the disk catalogue, and writes wherever it itself wishes.

If, on purpose or accidentally, you have changed any screen or information brought in by the editor, forth will when loading an other screen automatically place the modified screen on the diskette.
If you are careful, however, you can look at the material on any diskette using TI FORTH. For example, you could look perhaps at some of the words stored on an adventure diskette, to see what items you hadn't run accross yet.
If you are worried, you can tape the write-protect notch temporarily.

As a general rule, whenever you wish to load new material into Forth, or wish to look at new screens, you should empty the buffers with the EMPTY-BUFFERS command. This is especially so when you are using more than one diskette, with different material on each one. If screen 20 is resident in memory, for example and you wish to bring in screen 20 from another diskette you cannot do so without some preparation.

The easiest and surest way is EMPTY-BUFFERS, but that is not always useable, since you may wish to keep some of the buffer contents and not others. Suppose you had screens 20 and 21 from one diskette, and now wanted 21 from another, but you want to keep screen 20 from the first in memory. There is a way to do this: there is a user variable in the system which is called PREV. This variable stores the buffer location in RAM of the most recently used screen or block. The first two bytes of each screen buffer contain in the first byte an update flag, and in the second the screen number. (For Wycove Forth there is additionally a device offset contained in this number, which tells whether the screen belongs on CS1, DSK1, DSK2, or DSK3.

Thus, if you give the command HEX PREV @ U. the address of the last accessed screen buffer is printed out in hexadecimal. (@ retrieves the variable value, and U. prints the number out as an unsigned number).
Thus PREV @ @ U. will print out the first two bytes of the buffer giving you the screen number. We are interested in changing the screen number, let's say. Then all we need do is n PREV @ ! where n is the new screen number. (This does not included possibly needed offsets in the case of Wycove.) In this command, the ! does the storing (like a poke) to the value (as an address) placed on the stack by the @. In this command the n is the new screen number. Now that screen has a new number, though the material is still the same. It is however considered non-updated material because the update flag hasn't been set along with the screen number. The first bit of the first two bytes of the buffer must be a 1 to show it has been update; only then will the FLUSH command cause it to be saved to disk. (The automatic replacement policy when new screens are brought in, could then also save it, and wouldn't otherwise.) An example of making the latest screen have a new number that is updated:
HEX 8020 PREV @ ! DECIMAL will cause screen number 32 (in decimal) to be the number of the latest screen, and the 8 marks it as updated.
An easier way to cause the latest screen to be updated is to use the command UPDATE . An easy way to cause a screen to be the latest accessed is to edit it LIST it, as in 20 LIST. Then the PREV variable will contain its address.

Once you get on to it, it is not really all that complicated once you play with it a few times, and is worth getting to know if you're going to be using Forth. And to revert to our previous example of wanting to bring in the new screen 21 without emptying the buffers: we simply do 21 LIST, for example, followed by 0 PREV @ ! (7FFF in hex is better, but harder to enter).
Actually any number without the update flag should do, but it is best to use a number that isn't possible. (In TI Forth, 0 is a possible screen number only if you reset the DISK_LO variable) Now you can bring in the new screen 21.

In general, it is best to start an editing session, or a loading session, with EMPTY-BUFFERS. It is also a good command to use after fooling around: I accidentally destroyed one screen without knowing it. All I had done was look at some screens in an editor, then switched to another diskette, but I must have accidentally hit a key in the editor typing on the screen contents, or changing them (the editor shouldn't update otherwise), and thus unbeknownst to me, the screen was considered updated, and automatically rewritten to disk. Unfortunately, the disk was by then a different one, and the screen did not belong on it. In the process it took the place of one that did!

So user beware--though the automatic screen replacement system (almost a virtual memory system) has definite advantages, it has decided disadvantages in its present simplistic format. If screens were identified by diskette as well as by screen number, this problem would be relieved.

For the not quite beginner I've got another hint, concerning programming in FORTH. Although FORTH is totally interactive, and each piece of the program can be independently tested, making for greater ease in program development, there is one tremendous drawback. In Forth most parameters to routines are passed on the data stack. This stack is available to all routines. Much of Forth code is concerned with manipulating the order of the data on this stack, and retrieving it from this stack. This particular code is unreadable in the sense that it offers no clue as to what is going on. The code itself must carefully be worked out, in many ways like doing a puzzle: how do I get the 4th item to add with the 2nd, then compare it to the 3d, sort of thing.... The lesson is, keep your routines short, and the parameters to them as few as possible. Three should be the highest number to consider as a general rule, and even that is too much. Of course, it is the nature of the routine which decides how many parameters it needs, so keep the routines short. (They are that way also more flexible.) Variables are useful for items which need to be remembered and used at various times in the program. Variables use up memory space; stacked parameters do not. Finally, still in regard to stack manipulation, it is very important not only to check that a routine uses up its parameters properly, but that it also leaves on the stack only those items which it should leave, even if that is none. Check each branch of the routine. If the routine leaves 'pollution' on the stack, the routine itself may function properly, but the polluted stack will probably affect other routines further on adversely, especially if the polluting routine was a nested one. That sort of bug can be hard to trace down, (I speak from experience, since I am somewhat careless at complete testing!). The fact that Forth passes parameters on a common stack gives it some of its speed, and saves memory, but it also greatly complicates the writing, testing, and maintenance of FORTH code, and is my biggest complaint about it. If you do not place comments in your FORTH programs, you will not recall what is going on with the code two weeks after you wrote it. To be wise, at least place the before and after parameters for each routine in a comment after the routine name (as is standard practice). It is not easy to tell from the code what parameters are required, or even how many. (The latter would be obvious say, in Pascal, or Extended Basic.) So, to repeat, be careful with stack manipulations-- they are the source of most of the bugs. FINALLY, a quite dangerous, and hacky, stack manipulation, is that of temporarily storing data on the return stack. This appears to be standard FORTH practice, but can get you into a lot of trouble, so should be done with caution.

Other than that, I have no great hints for beginning FORTH users. It is the best language presently available on the TI for simply playing with the machine, and exploring its features, interactively. Though it is to my mind far from satisfactory as a language, it may well be the best we currently have for the TI (except for those who have the P-Code peripheral). Certainly both LOGO and EXTENDED BASIC are preferable from a readability, and a learning viewpoint, but neither of those offer much in the way of speed, and both limit one's access to the machine. Therefore FORTH has a definite place in our library of languages. It is much easier to use than Assembly Language, yet will allow the user to accomplish most anything he would wish. To those willing to explore the mysteries of the TI99/4A: GO FORTH !

```
********************************************************************************

                                  THE 9-PUZZLE
                                    extended
                                     basic

                                    *******
                                    *1 2 3*
                                    *4 5 6*
                                    *7 8  *
                                    *******
                                       by
                                  TONY BIGRAS


100 DIM TILE(9)      * make array for tiles.
110 RANDOMIZE
120 CALL MAGNIFY(2) * set sprite size for dblsize i used sprites for the tiles.

130 RANDOM$="973815642137845962387194625491836275416235798"
140 RANDOM$=RANDOM$&"263751849361459278356921874631598724184935672"
150 RANDOM$=RANDOM$&"351764892345628719748961352896745321987654321"
160 RANDOM$=RANDOM$&"741852963238495617314796258248135697891743265"

    * make one big string with the scrambled patterns for the puzzles.


170 MOVES=-1         * this gets incremented before it is first displayed.

180 X(1),X(2),X(3)=73    *
190 X(4),X(5),X(6)=89    * sets the X and Y coordinates for the sprites
200 X(7),X(8),X(9)=105   * that are used to display the tiles.
210 Y(1),Y(4),Y(7)=105   *
220 Y(2),Y(5),Y(8)=121   *
230 Y(3),Y(6),Y(9)=137   *

240 CALL CLEAR
250 DISPLAY AT(24.5)BEEP:"release alpha-lock"

    * both the arrow keys and the joysticks are usable if alpha-lock is up!


260 CALL COLOR(14,16,7)
270 CALL CHAR(142."0000000000000000FFFFFFFFFFFFFFFF")

    * sets game color and initializes character patterns.
    * all values have been initialized *


280 FOR T=72 TO 112 STEP 8        *
290 CALL HCHAR(1+T/8,14,143,6)  * draws a 6x6 white rectangle on the screen.
300 NEXT T                        *

310 CALL HCHAR(9,14,142,6)        *
320 CALL HCHAR(16,14,142,6)       * draws a red border around the rectangle.
330 CALL VCHAR(9,13,142,8)        *
340 CALL VCHAR(9,20,142,8)        *

350 FOR T=1 TO 8 :: CALL SPRITE(#T,T+48,2,240,100):: NEXT T
360 CALL SPRITE(#9.143,9,240,100)
```

```
    * sets sprite patterns to 1 thru 8 and a solid and holds them off screen.


370 JUMBLE$=SEG$(RANDOM$,(INT(RND*20)+1)*9-8,9)
380 FOR T=1 TO 9
390 TILE(T)=VAL(SEG$(JUMBLE$,T,1))
400 IF TILE(T)=9 THEN BLANK=T
410 NEXT T

    * randomly selects pseudo-random patern from RANDOM$ and loads the pattern
    * into the TILE() array.


420 FOR T=1 TO 9 :: CALL LOCATE(#TILE(T),X(T),Y(T)):: NEXT T

    * puts the sprites on the screen in scrambled pattern.


430 GOSUB 690 * start game enter play loop at end of loop to put MOVES on screen

440 CALL JOYSTK(1,K,S) * call JOYSTK to scan input from keys and from joysticks.

    * JOYSTK converts joystick input to call key(1,k,s) output.


450 IF S=0 THEN 440 * if no input then ask for input again.

460 IF K=18 THEN MOVES=-1 :: GOTO 370 * if fire button is hit or key Q is hit

    * then the player wants to end game or pick a different pattern, or both.


470 IF K<>0 AND K<>2 AND K<>3 AND K<>5 THEN 440 * if invalid key then ask again.

480 IF K<>5 THEN 520     * if input is not down then next check.
490 IF BLANK>6 THEN 440 * if down move not legal at this time then ask again.
500 TEMP=BLANK+3        * TEMP = new position of sprite #9 solid.
510 GOSUB 660           * go move the tile.

520 IF K<>0 THEN 560     *
530 IF BLANK<4 THEN 440 * same as above except in up direction.
540 TEMP=BLANK-3        *
550 GOSUB 660           *

560 IF K<>2 THEN 600                           *
570 IF BLANK=3 OR BLANK=6 OR BLANK=9 THEN 440 * same but to the right.
580 TEMP=BLANK+1                              *
590 GOSUB 660                                 *

600 IF K<>3 THEN 640                           *
610 IF BLANK=1 OR BLANK=4 OR BLANK=7 THEN 440 * same but to the left.
620 TEMP=BLANK-1                              *
630 GOSUB 660                                 *

640 GOTO 440            * this is the end of the play loop start over again.

650 GOTO 650            * this is left over from program development!

660 TILE(BLANK)=TILE(TEMP) * this is where the tile get moved around.
670 TILE(TEMP)=9           * have to keep track of what was and what will be.

680 CALL LOCATE(#TILE(BLANK),X(BLANK),Y(BLANK)):: CALL LOCATE(#TILE(TEMP),X(TEMP
),Y(TEMP)):: BLANK=TEMP

    * moving sprites around and keeping track of wher the blank tile is now.
```

```
690 MOVES=MOVES+1 :: DISPLAY AT(4,11):"MOVES";MOVES   * this is where we came to
700 CALL SOUND(40,1400,1,4000,5,3000,15)              * enter the play loop from
710 RETURN                                            * line 430

   * this shows moves makes noise and return from whence it came.


720 SUB JOYSTK(SIDE,KEY,STATUS)                       *
730 CALL JOYST(SIDE,X,Y)                              * this is where key and
740 IF X=0 AND Y=0 THEN STATUS=0 :: GOTO 840          * joystick input come in
750 STATUS=1                                          * and become output that
760 IF X=0 AND Y=4 THEN KEY=5 :: GOTO 840             * works like a
770 IF X=4 AND Y=4 THEN KEY=6 :: GOTO 840             * call key(1,k,s) statment
780 IF X=4 AND Y=0 THEN KEY=3 :: GOTO 840             *
790 IF X=4 AND Y=-4 THEN KEY=14 :: GOTO 840           * i alreadv know how it
800 IF X=0 AND Y=-4 THEN KEY=0 :: GOTO 840            * works but you can figure
810 IF X=-4 AND Y=-4 THEN KEY=15 :: GOTO 840          * it out for yourself or
820 IF X=-4 AND Y=0 THEN KEY=2 :: GOTO 840            * give me a call if you
830 IF X=-4 AND Y=4 THEN KEY=4 :: GOTO 840            * have any questions about
840 CALL KEY(SIDE,K,S):: IF S=0 THEN 870              * this program 383-3946
850 IF SIDE=1 THEN KEY=K :: STATUS=S
860 IF SIDE=2 THEN KEY=K :: STATUS=S :: IF KEY=18 THEN KEY=11
870 SUBEND
```

****