JOHN WAGNER

ATARI
COMPUTER ENTHUASISTS
of COLUMBUS, OHIO
NEWSLETTER

June 11, 1984

THE ARCADE MACHINE
Strolling through MeasFORTH
ATARI COLOR GRAPHICS

Published by
Atari Computer Enthuasists of Columbus, Ohio

for ACE of Columbus membership. Dues are on an annual basis and entitle the members to all club benefits (Newsletter, Disk or Tape of the month, group discounts, etc.). Monthly meetings, in the basement of State Savings, 6895 N. High Street, Worthington, Ohio are open to nonmembers.

Upcoming meeting dates at 7:30 pm are

DATE

June 11
July 9
August 13
September 10
October 8
November 12
December 4

PRESIDENT
Bill Eckert
6632 Lisamarie Road
Columbus, Ohio 43229
614-891-9785

VICE PRESIDENT
Donald Noble
7536 Northfield Court
Reynoldsburg, Ohio 43868
614-866-4587

MEMBERSHIP CHAIRMAN
Tim Adcock
7544 Satterfield Road
Worthington, Ohio 43885
614-764-9492

TREASURER
Mike Compton
1342 Gumwood
Columbus, Ohio 43229
614-885-3757

DISK LIBRARIAN
Sheldon Wesson
444 North Pearl Street
Granville, Ohio 43823
614-587-2716

CASSETTE LIBRARIAN
Larry Fletcher
12388 Oak Drive
Orient, Ohio 43146
614-877-9538

NEWSLETTER EDITOR
Norman Knapp
1222 Norton Avenue
Columbus, Ohio 43212
614-291-2849

ADDRESS ALL MAIL TO
ACE of Columbus
P.O. Box 849
Worthington, Ohio 43085

THE ARCADE MACHINE

Disc version by Chris Jochumson, Doug Carlston, and Louis Ewens
Broderbund Software
17 Paul Drive, San Rafael, CA 94903

In my opinion, the Arcade Maching is one of the best games ever made. The reason for this is that it is not just a game but a game generator. Not only can you play games with this program, but you can actually make your very own games.

The number or variety of games you can make is almost unlimited. You do not have to have any programming knowledge. Although the Arcade Machine is a complicated program, once you understand it, it is simple to use. If I can figure it out, almost anybody can.

There are several parts to this program and I will attempt to highlight them. The first thing to do is to design with a joy stick your enemy creatures, called aliens. Each alien you create is designed in four different positions. It rotates through these four designs when it moves diagonally or horizontally. Animation effects, like opening and closing its eyes, make the alien look like it's alive; a bird with moving wings looks like it's flying across the screen. You may also design your own player, known as a tank, and explosions.

The aliens can be on a maximum of four rows. Dependent on their size, there may be from two to six aliens per horizontal row with a maximum of 24 in all the rows.

After the aliens have been designed, the Path Generator is used to create up to ten different paths that your aliens can move about on. The path generator is a complicated procedure which permits your aliens to move in a circle, horizontally, vertically, or in any direction you may choose; each of the ten paths may be designed step by step with many extra options such as speeding the aliens up, slowing them down, or even changing their shapes.

In designing your game, there are five levels. As you progress through the levels, conditions may be made easier or harder; the choice is yours.

The major flaw in the Arcade Maching is that your tank can't fire horizontally, just up or down or diagonally.

I highly recommend the Arcade Maching. Not only is it fun to play, but it is fun to make and play your own game. The Aracde Machine is also educational since you can learn about animation and the other features that make up a game.

The Aracde Maching not only gives you the inspiration to come up with new thoughts and ideas, but it also gives you a peek at how powerful our little computers really are.

Reviewed by Charles W. Brown

Strolling through MesaFORTH

Boot Side 1 of DOM 20b; get the ready message

```
fig-FORTH 1.0D
```

Let's inspect the NOISES.4TH file with the screen editor, then load and run it. Enter

```
" D1:NOISES.4TH" S1EDIT
```

to view screen 1. (See Chapter 5 of the documentation for editor commands.) The purpose of NOISES.4TH is to demonstrate the central concept of FORTH programming, which is the use of small "words" (subroutines) to define larger words that are used to define still larger words. FORTH programs are therefore written "backwards", with the simplest words defined first. The most fundamental word defined in this file is

```
: SOUND  ( pitch chan# --> )
  SWAP 10 10 SO. ;
```

A definition is contained between a colon and semicolon. The parenthetical expression is a comment that lists the input and output parameters. This word is the FORTH eqivalent of the BASIC command

```
SOUND chan#,pitch,10,10
```

The words

```
: SOUND0  ( pitch --> pitch chan0 )
  0 SOUND ;

: SOUND1  ( pitch --> pitch chan1 )
  1 SOUND ;
```

call SOUND and pass to it the channel number. (Press ESC > to view screen 2.)

```
: LOWER.PITCH  ( pitch --> pitch )
  SOUND0 ;
```

accepts a value for pitch from a higher level word and calls SOUND0.

```
: RAISE.PITCH  ( pitch --> pitch )
  255 SWAP - SOUND1 ;
```

accepts a pitch value, subtracts it from 255, and calls SOUND1. We can put these words into DO loops to make whooping and whistling noises; but first, let's define a few more helper words.

```
: BEATS  ( n --> )
  50 *  0 DO LOOP ;
```

accepts a number, multiplies it by 50 and uses the product as the limit of an empty DO loop. n BEATS provides n time intervals between events, and is used in exactly the same way as empty FOR/NEXT loops in BASIC.

```
: OFF  ( chan# --> )
  0 0 0 SO. ;
```

turns off the sound on the designated channel. (Press ESC > to view screen 3.) Until now we have defined words that don't do anything by themselves, but are used in sequence by higher level words. This part of FORTH programming is like making a neat pile of bricks on the ground, in preparation for building a wall. Let's now lay a section of wall:

```
: CHIRP
  50 0
   DO  I LOWER.PITCH  LOOP
  0 OFF ;
```

passes the index of a DO loop to LOWER.PITCH, making a brief whistle, then turns channel 0 off. We want to define a word that will generate a random number of chirps, so let's build a number generator:

```
: 0<RND<8  ( --> n )
  D20A C@ 7 /MOD DROP
  DUP 0= IF 7 + ENDIF ;
```

Notice that a word name can consist of any combination of letters, numbers and characters. 0<RND<8 generates a number between 1 and 7 by collecting a random byte from address D20A, dividing the number by 7, obtaining the remainder and adding 7 to the remainder if it is zero. We are now prepared to build

```
: CHIRPS
  0<RND<8
  0 DO  CHIRP  8 BEATS  LOOP ;
```

which produces n chirps, each separated by an interval of 8 beats. Notice that CHIRPS calls all of the preceeding words except SOUND1. (Screen 4, please.)

```
: SIREN
  180 100
   DO
   I LOWER.PITCH  2 BEATS
   LOOP
  0 OFF ;
```

Nothing new here, except that we can make an entirely different sound by tinkering with CHIRP.

Screen 5 contains

```
: CROSS
  250 50
   DO
   I RAISE.PITCH
   I LOWER.PITCH
   LOOP
  0 OFF  1 OFF ;
```

which produces two tones, one ascending and the other decending. We now have three NOISEMAKER words: CHIRPS, SIREN and CROSS. What's needed is a single word that chirps, wails and whoops. But first we bake more bricks in screens 6 and 7:

```
: START?  ( --> 0/1 )
  DO1F C@ 6 = ;

: CURSOR.OFF
  1 2F0 C! ;

: CLEAN.SCREEN
  0 GR.  ." ok"  QUIT ;

: WAIT
  80 BEATS ;

: SET.SCREEN  ( col row --> )
  0 GR.
  2 0 4 SE.  4 0 8 SE.
  POS.  CURSOR.OFF ;
```

START looks to see if the console key is held down. CURSOR.OFF places 1 in the appropriate address; CLEAN.SCREEN calls GRAPHICS 0 and does some housekeeping. SET.SCREEN calls GR. 0, SETCOLORs the screen and border, positions the cursor to write a line of text and then hides the cursor. Screen 8 contains a nice neat section of wall:

```
: NOISES
  10 10 SET.SCREEN
  ." Hold START to exit"
   BEGIN
    SIREN
    WAIT
    CHIRPS
    WAIT
    CROSS
    WAIT
   START? UNTIL ;
```

NOISES prints a prompt onscreen, then enters a BEGIN/UNTIL loop, makinr each noise in sequence until the the console key is held down. Next, th payoff: a menu from which we can choose SIREN, CHIRPS, CROSS, or all three in sequence (NOISES).

But first, two more bricks on screens 9 and 10:

```
: ENTER.CHOICE  ( n --> m )
  KEY  DUP  EMIT
  48 - ;

: TEST.CHOICE  ( m --> )
  DUP 1 = IF SIREN  ELSE
   DUP 2 = IF CHIRPS ELSE
    DUP 3 = IF CROSS  ELSE
     DUP 4 = IF NOISES ELSE
      DUP 0 = IF DROP
                 CLEAN.SCREEN
                 QUIT
               ENDIF
             ENDIF
           ENDIF
         ENDIF
       ENDIF    DROP ;
```

ENTER.CHOICE is a menu builder word that takes a number from the keyboard, prints it onscreen, and subtracts 48 to change the internal representation of the keystroke from ATASCII to its numerical value. TEST.CHOICE is a set of nested IF/ELSE/THEN structures that calls the selected noisemaker word, quits the program on input of zero, and rejects input that is out of range. And now, the word we've all been waiting for (screen 11):

```
: MENU
  BEGIN
   9 5 SET.SCREEN
   ." The L&P NOISEMAKER"
   15  7 POS. ." 1  SIREN"
   15  8 POS. ." 2  CHIRPS"
   15  9 POS. ." 3  CROSS"
   15 10 POS. ." 4  NOISES"
   15 11 POS. ." 0  QUIT"
   12 14 POS. ." Enter choice: "
   ENTER.CHOICE  TEST.CHOICE
  AGAIN ;

MENU
```

BEGIN/AGAIN is an endless loop that prints the prompts, takes a cue from the keyboard and sends it off to be processed by TEST.CHOICE. Note the word MENU at the bottom of the screen. Words not encased by : and ; are excecuted when the program is compiled. MENU thus serves as an autostart, executing the last definition as soon as it is compiled into the dictionary.

The L&P NOISEMAKER is an example of FORTH programming that illustrates some important points: (1) write short words (2) make them general, so that the same function can be performed in different situations (3) define a word to do anything that is done more than once (4) use the parameter stack to pass numbers between words (5) leave the stack clean at the end of words that don't pass parameters (6) document stack input and output in words to aid debugging (7) use descriptive names that make

the higher-level words readable (8) format the code, grouping parts of words together according to function. There is no penalty in RAM usage for spacing and commenting in FORTH files, as there is in BASIC.

Now let's compile and run the NOISEMAKER. Press ESC a to leave the editor without changing the file. (ESC x updates the disk on leaving, for creating or editing code.) When you get the prompt

OK

enter

1 LOAD

to load screen 1. The arrows in the lower right corner of each screen link the screens for successive loading. Words enclosed by : and ; are compiled, while words listed in the immediate mode (such as HEX and DECIMAL) are excecuted. When MENU is excecuted the NOISEMAKER program begins.

Turn up the volume on your TV and play with the menu options. Notice that any keystroke that is out of range falls through TEST.CHOICE, passing control to the bottom of the endless loop in MENU.

Press 0 to quit. All of the words defined in the file are now compiled and available for use in the immediate mode. Enter

SIREN CHIRPS SIREN SIREN CROSS MENU

to make the point. Leave the menu, and enter

80 1 SOUND

then

1 OFF

to demonstrate parameter passing in the immediate mode. Be sure to provide parameters to words that require them, or the computer may hang.

It's time to clear NOISES.4TH from RAM to make space for another program. The first word in the file is

: TASK ;

a dummy word that serves as a "bookmark" in the dictionary. Enter

FORGET TASK

to delete all NOISES.4TH words from RAM starting with MENU, and proceeding back to and including TASK. (Use TASK at the begining of each file so that you can compile and FORGET successive versions of the program without filling up RAM during debugging.)

Let's run the snowflake algorithm. Enter

```
" D1:FLAKES.4TH" $LOAD
```

and follow the prompt. Press any key to exit the program. NOISES.4TH and FLAKES.4TH are small programs and could coexist in RAM together, except that the word MENU is used in both, and assumes the function of the most recently compiled version. Enter

```
FREE .
```

to monitor free RAM in the course of a program or during debugging.

Now we want to create a new file DOODLE.4TH on the disk with 10 blank screens. Enter

```
10 " D1:DOODLE.4TH" CREATE.FILE
```

CREATE.FILE was appended to the DISK.4TH file on Side 2 and is defined as

```
: S1  ( filespec -$> )
  $SETDR1 DR1 ;

: DOS.FILE  ( filespec -$> )
  #3 8 0 $FILE OPEN
  #3 CLOSE ;

: CREATE.FILE  ( n --> )
  DOS.FILE     ( filespec -$> )
  S1
  1 SWAP ADDBLKS
  $DROP ;
```

along with another useful word

```
: S1EDIT  ( filespec -$> )
  S1
  1 EDIT ;
```

Unfortunately, a bug in ADDBLKS causes the computer to hang, but not until after it does its good work. When the endless OKs begin shut off the computer and reboot, then enter

```
" D1:DOODLE.4TH" S1EDIT
```

in preparation for writing a new program. When screen 1 comes up, enter

```
: TASK ;

: POSITION.TEXT
  CR CR  5 SPACES ;

: GREETINGS
  POSITION.TEXT
  ." Hello from Screen 1." ;

-->
```

in which the arrow is written "minus minus greater-than". Proceed to screen 2 by entering ESC > . Enter

```
: SALUTATIONS
  POSITION.TEXT
  ." Aloha from Screen 2." ;

3 LOAD
```

Notice that we can chain with LOAD as well as --> . Press ESC > to edit screen 3 and enter

```
: WAIT
  20000 0 DO LOOP ;

: &
  WAIT
  2 0 0 SE. ;

: GOODBYE!
  WAIT
  POSITION.TEXT
  ." Screen 3: over and out."
  WAIT
  0 GR. ;

-->
```

Proceed to screen 4 and write the word that puts it all together:

```
: TALK.TO.ME
  GREETINGS & SALUTATIONS
  GOODBYE! ;
```

Close out the file with ESC x and enter

```
1 LOAD  TALK.TO.ME
```

Wowie zowie, your first FORTH program is online. Go back to the editor, improve it and load it again. Don't forget to FORGET TASK between loads.

Side 2 of DOM #20b contains the kernel and source files that were combined to make the AUTORUN.SYS file on Side 1. Side 2 boots up and loads INTRFACE.IN0,MX80.IN1,FORTHD1B.IN2,BOOTEDIT.4TH,EDITOR.4TH,SYSTEM.4TH and TURNKEY.4TH. I appended a few screens to DISK.4TH, LOADed it, and created the AUTORUN.SYS file for Side 1 using TURNKEY, as described in the documentation. You can customize a larger or smaller boot file in the same way. Happy computing, and

May the FORTH be with you.

+ - +

Published by Sheldon P. Wesson, ACEC Disk Librarian -- 444 North Pearl Street, Granville, Ohio 43023