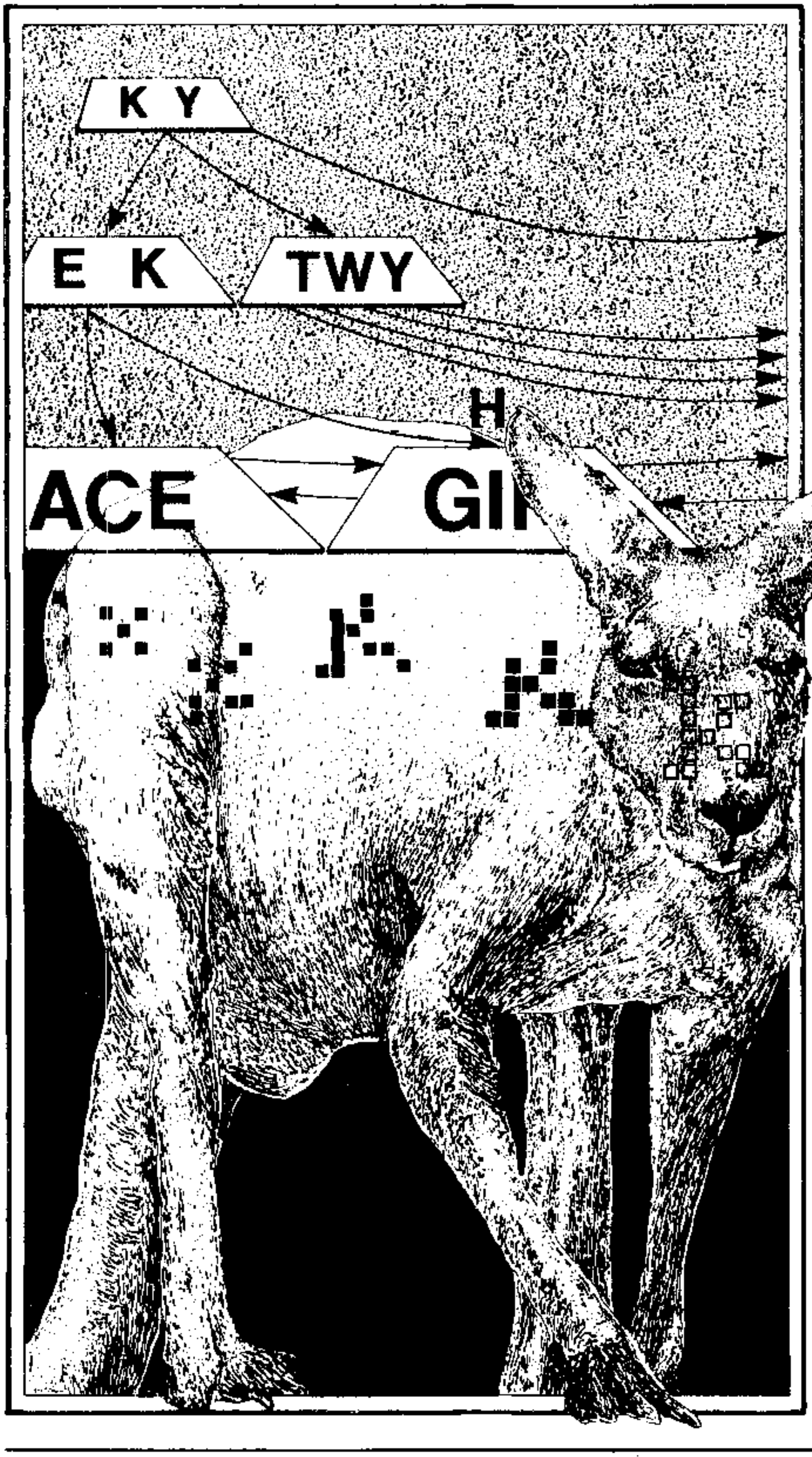
#80 June 1983 $2.50 ($3.00 in Canada)

Dr. Dobb's Journal

For Users of Small Computer Systems



B-Tree ISAM Concepts

CP/M BDOS and BIOS Calls for C

IBM PC: Printing Graphics and The Game of Life

. . . and much more!

Dr. Dobb's Journal
For Users of Small Computer Systems

June 1983 Volume 8, Issue 6

Publisher – Clifford West
Editor – Reynold Wiggins
Managing Editor – Craig LaGrow
Editorial Consultant – Marlin Ouverson
Contributing Editors –
Robert Blum, Dave Caulkins,
Dave Cortesi, Ray Duncan,
Michael Wiesenberg
Marketing Director – Craig Harper
Marketing Manager – Beatrice Blatteis
Advertising Director – Carl Landau
Advertising Sales – Doug Millison
Circulation Manager – Gloria Romanoff
Circulation Assistants –
Terri Marty, Linda Marohn
Production Manager – Barbara Ruzgerian
Production Assistants –
Alice Hinton, Alana Hunter,
Jane A. McKean, Marianne Tryens
Typesetter – Paula Fairchild
Cover Illustration – Al McCahon

Subscription Rates: $25 per year within the United States; $44 for first class to Canada and Mexico; $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Writer's Guidelines: All items should be typed, double-spaced on white paper. Listings should be produced by the computer, using a *fresh, dark ribbon* on continuous white paper. Please avoid printing on perforations. Payment is in contributor's copies. Requests to review galleys must accompany the manuscript when it is first submitted. Authors may receive a copy of the complete writer's guidelines by sending a self-addressed, stamped envelope.

Donating Subscribers **Contributing Subscriber:** $50/year ($25 tax deductible). **Retaining Subscriber:** $75/year ($50 tax deductible). **Sustaining Subscriber:** $100/year ($75 tax deductible). **Lifetime Subscriber:** $1000 ($800 tax deductible). **Corporate Subscriber:** $500/year ($400 tax deductible, receives *five* one-year subscriptions).

Contributing Subscribers: DeWitt S. Brown, Burks A. Smith, Robert M. Connors, Robert C. Luckey, Transdata Corporation, Mark Ketter, John W. Campbell, Friden Mailing Equipment, Frank Lawyer, Rodney Black, Thomas Davis, Jan Steinman, G. Hunter, Ronald E. Johnson, Kenneth Drexler, Real Paquin, Ed Malin, John Saylor, Jr., Ted A. Reuss III, Infoworld, Stan Veit, Western Material Control, S.P. Kennedy, Ed Moran, W.D. Rausch. **Lifetime Subscriber:** Michael S. Zick.

Foreign Distributors **UK & Europe:** Homecomputer Vertriebs HMBH 282, Flugelstr. 47, 4000 Dusseldorf 1, West Germany; La Nacelle Bookstore, Procedure D'Abonnement 1-74, 2, Rue Campagne – Premiere, F-75014 Paris, France; Computercollectief, Amstel 312A, 1017 AP Amsterdam, Netherlands. **Asia & Australia:** ASCII Publishing, Inc., 4F Segawa Bldg. 5-2-2, Jingumae, Shibuya-Ku, Tokyo 150, Japan; Computer Services, P.O. Box 13, Clayfield QLD 4011, Australia; Computer Store, P.O. Box 31-261, 22B Milford Rd., Milford, Auckland 9, New Zealand. *(Write for Canadian distributors)*

CONTENTS

ARTICLES

DEPARTMENTS

The Game of Life
on the IBM-PC

Computer programs for running John Conway's Game of Life were quite popular a few years ago. The problem with these older simulations was usually their speed: a single generation on a ten-by-ten grid could take up to 90 seconds. Presented here is a version of the game for the IBM/PC computer. Features of this version are adjustable speed (with 2.7 generations/sec as a top speed), easy entry of the seed generation (via a screen-editor), and marking of cells due to be "born," and cells remaining alive each generation (as different from cells marked to die). This feature gives a better impression of fluidity on the grid.

This version requires only the IBM monochrome display to run, and will run under either MS-DOS or CP/M-86, since only calls to the IBM ROM are made.

Background

The Game of Life takes place on a square grid. Every cell on the grid can be either alive or dead. Every cell on the grid has eight neighbors. Each generation, every cell on the grid is evaluated. The cell will remain alive if it has two or three neighbors, or will die if it has less than two (from loneliness), or more than three (from overpopulation). An empty cell will have a "birth" (a live cell will be placed there the next generation) if it has exactly three neighbors. The game was developed by John Conway in 1976.

by Simson L. Garfinkel

Simson L. Garfinkel, 18 Dartmouth Lane, Haverford, Pennsylvania 19041.

The Program

The program is straightforward. When first run, the screen is cleared and the cursor is positioned in the center. The cursor may be moved by pressing the four arrow keys on the keyboard (Num Lock doesn't matter because the keyscan code is interrogated, not the ASCII code). A live cell may be deposited by depressing Ins, and a cell may be cleared by pressing Del. When the screen is complete, pressing Esc starts evaluation of the next generation.

The program stores the grid in the screen memory of the monochrome display. Associated with every displayable character on the screen are two bytes of memory, the low byte for the character displayed, and the hi byte for the display attributes (underlined, reversed, etc.). The program uses the attribute byte of every screen position as a second array to hold the next generation in while it evaluates the present generation.

After the time delay, subroutine *count* is called. *Count* counts the number of neighbors that every cell has, and decides whether or not the cell will be alive in the next generation. If it is going to be alive, the display attribute for that character is set to *rev*. If the cell is going to be dead, the attribute is set to *dark*. *Rev* and *dark* are reverse video and normal video in my listing, so when a cell is going to have a birth or stay alive, it is inverted on the screen, but *rev* and *dark* can be changed to three and five, causing both to display as normal if the flickering this produces is annoying.

After all of the decisions regarding life and death are made, subroutine *update* goes through screen memory putting in the character for a live cell if the cell is supposed to be alive, or a dead cell if the cell is supposed to be dead, and resetting the attribute byte.

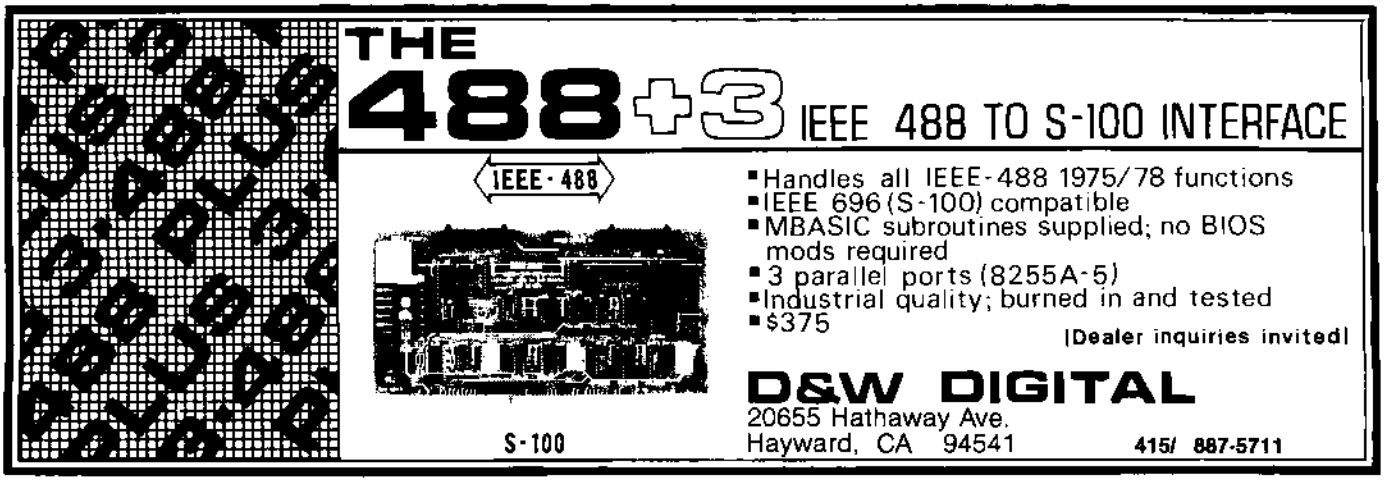
The program then looks for a user key press. If one has occurred, the program quits if it was an Esc, or changes the time delay value if it was a digit. Pressing 0 gives no delay, or about 2.7 generations per second. Pressing 9 gives a 3.5 second delay per generation.

Expansions

The one problem with this implementation is that the screen doesn't have enough rows to allow a simulation of a complex colony. I would love to see an implementation of life in medium-resolution color graphics, with births in blue and deaths in red, but I don't have access to a color display, so that will have to wait. The program could also be cleaned up to make it run faster, but 2.7 generations a second is really fast enough for most applications.

(Listing begins at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 237



Game of Life (Text begins on page 42)

```
Comment $

        *********************************
        *                               *
        *       The Game of Life        *
        *                               *
        *********************************

        John Conway's mathematical game of life, implemented on
        the IBM/PC, by Simson L. Garfinkel.

        Written in 8088 assembly language using the Microsoft
        Macro Assembler.

        Notes on running the program:

          When program is run:

                1.  Screen clears.
                2.  User enters first generation from keyboard.
                    Arrow keys move the cursor.  INS key deposits
                    a live cell, DEL removes a live cell, (in case
                    the user makes a mistake.)
                3.  Pressing ESC starts program.
                4.  For each generation, cells which will have life
                    on the next turn are inverted.
                5.  Screen is updated to next generation.
                6.  Keyboard is interrogated for command.
                7.  If ESC is pressed, program terminates.
                8.  If a number 0-9 is pressed, speed is selected.
                    At speed 0, approx. 2.7 generations/sec are performed
                    At speed 9, each generation takes 3.5 sec.
                9.  program loops to #4.

        $


        ;global definitions

live    equ  02           ;character for a live cell
dead    equ  00           ;character for a dead cell

rev     equ  70h          ;reverse video (marks cell to live)
dark    equ   2           ;normal video  (marks cell to die )

time    equ  300          ;time delay base

TERM    equ  27           ;Character to exit mode


cseg    segment para public 'code1'
start   proc far
        assume cs:cseg,ds:nothing,ss:stack,es:nothing

                          ;set up return location
```

(Continued on next page)

Game of Life (Listing continued, text begins on page 42)

```
        push ds
        sub  ax,ax
        push ax             ;now I can go home when I'm finished.


        call Enter          ;Enter board
        mov  cx,0           ;initial delay, 0
main:
        push cx             ;save delay variable
        cmp  cx,0
        jz   s13

s1:     push cx
          mov  cx,time
s11:      push cx
            mov  cx,time
s12:        loop s12
          pop  cx
          loop s11
        pop  cx
        loop s1             ;what a time delay!

s13:    call count          ;Count up every cell's neighbours,
        call update         ;Update screen
        pop  cx             ;get back the time delay

        mov  ah,1           ;See if user has pushed a key
        int  16h
        jz   main           ;nope - loop back

        mov  ah,0           ;get the character our of the buffer
        int  16h

        cmp  al,TERM
        jnz  s2
        ret                 ;finished - go back to ms/dos

s2:     cmp  al,'0'         ;see if it is a speed command
        jb   main
        cmp  al,'9'
        jnbe main
                            ;It's a number
        sub  al,'0'         ;now it goes from 0 to 9
        mov  ah,0
        mov  cx,ax          ;put it in cx
        jmp  main
start   endp



Enter   proc near           ;Subroutine to enter board
                            ;define scan-codes:
left    equ  75
right   equ  77
up      equ  72
```

```
down     equ  80
point    equ  82
del      equ  83
esc      equ   1

         call cls          ;clear the screen

         ;Registers are used as follows:
         ;DH - Y position
         ;DL - X position

         mov  dh,12
         mov  dl,40

e1:      mov  bh,0         ;move the cursor to x,y position
         mov  ah,2         ;code for cursor move
         int  10h          ;interrupt for cursor move

         mov  ah,0         ;set up to read the next keypress
         int  16h          ;keypress read

         cmp  ah,left      ;make a rational decision about the users
         jz   go_left      ;entry.
         cmp  ah,right
         jz   go_right
         cmp  ah,up
         jz   go_up
         cmp  ah,down
         jz   go_down
         cmp  ah,point
         jz   go_point
         cmp  ah,del
         jz   go_del

         cmp  ah,esc
         jnz  e1           ;loop back - unknown command
         mov  dx,23*256    ;put the cursor at lower-left hand corner
         mov  ah,2
         int  10h
         ret               ;go back to caller

go_left:                   ;move left if I can
         cmp  dl,0         ;in leftmost collumn?
         jz   e1           ;yes - go back
         sub  dl,1         ;no  - subtract one
         jmp  e1           ;go back

go_right:                  ;move right if I can.
         cmp  dl,79
         jz   e1
         add  dl,1
         jmp  e1

go_up:                     ;go up if I can
         cmp  dh,0
         jz   e1
         sub  dh,1
         jmp  e1
```

(Continued on next page)

Game of Life (Listing continued, text begins on page 42)

```
go_down:                ;go down if I can
        cmp  dh,24
        jz   e1
        add  dh,1
        jmp  e1

go_point:               ;put a live dot where the cursor is -- don't move it
        mov  al,live    ;it's the live character
gp2:    mov  cx,1       ;one character to write
        mov  ah,10      ;code to write character
        int  10h        ;do it
        jmp  e1         ;get next command

go_del:                 ;delete character at cursor
        mov  al,dead
        jmp  gp2        ;let go_point do the rest
Enter   endp



Cls     proc near       ;Subroutine to clear the screen
        mov  ax,6*256
        mov  cx,0
        mov  dx,24*256+79
        mov  bh,2
        int  10h
        ret
cls     endp


Count   proc near       ;Subroutine to count up every cell's neighbours
        ;Registers used:
        ;DH,DL:  Y,X of current cell being interrogated
        ;DS   :  Base offset - into screen memory
        ;DI   :  offset for character presently being looked at
        ;
        ;Outline for each character
        ;  1.  Count up number of neighbours
        ;  2.  If three neighbours, or if two and cell is live, put
        ;      a rev  on the screen at the attribute position, else
        ;      put a dark
        ;  3.  Go to next character


chk     macro  yy,xx
        local  ch1,offs
offs    equ    (xx+yy*80)*2
        mov    cx,[di+offs]                ;get byte to check
        cmp    cl,live                     ;check to see if this cell is alive
        jnz    ch1                         ;nope
        add    al,1                        ;yes - increase neighbour count
ch1:
        endm
```

(Continued on page 50)

Game of Life (Listing continued, text begins on page 42)

```
        mov    ax,0B000H
        mov    ds,ax          ;offset value for monochrome display

        mov    dh,1           ;Start at 1,1 and go to 23,78
        mov    dl,1           ;to prevent wrap-arround

c1:     mov    ax,160         ;get true offset from ds into screen memory
        mul    dh

        mov    cx,dx
        mov    ch,00          ;just get dl
        add    ax,cx
        add    ax,cx          ;ax:=(dh*80+dl)*2

        mov    di,ax          ;di:=ax
        mov    ax,0           ;ax will be used for neighbour counting

        chk    -1,-1          ;count number of neighbours
        chk    -1, 0
        chk    -1,+1
        chk     0,-1
        chk     0,+1
        chk    +1,-1
        chk    +1, 0
        chk    +1,+1          ;test all of the neighbours

        mov    cx,[di]        ;get byte to check
        cmp    al,3
        jz     give_life      ;life if has 3 neighbours
        cmp    cl,live        ;is it alive?
        jnz    give_death     ;no
        cmp    al,2           ;he lives if he has 2 neighbours and he is already
                              ;alive
        jnz    give_death     ;nope

give_life:                            ;make this one alive
        mov    ch,rev
        jmp    c2

give_death:
        mov    ch,dark
c2:     mov    [di],cx                ;put back on the screen

next_cell:
        cmp    dl,78          ;am I at the end of the X line?
        jz     c3             ;yes
        add    dl,1           ;nope
        jmp    c1
c3:     mov    dl,1
        cmp    dh,23          ;am I at the end of the Y line?
        jz     c4             ;yes
        add    dh,1           ;nope
        jmp    c1
c4:     ret                   ;yes - go home!
Count   Endp
```

```
Update  proc near               ;This updates the generation on the screen
        mov  ax,0B000H          ;Get screen offset
        mov  ds,ax

        mov  bx,24*80*2-2       ;loop through all of the screen but last line
u1:     mov  cx,[bx]            ;line
        cmp  ch,rev             ;is it to live?
        jnz  u2                 ;no
        mov  cl,live            ;yes
        jmp  u3
u2:     mov  cl,dead            ;no
u3:     mov  ch,dark            ;turn off reverse
        mov  [bx],cx            ;put it back on the screen
        sub  bx,2               ;loop back until done with the screen
        jg   u1
        ret                     ;go back to caller
Update  endp
cseg    ends


stack   segment para stack 'stack'
        db 30 dup('stack  ')
stack   ends


        end
```

End Listing