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!
CONTENTS

ARTICLES

14 Fast Divisibility Algorithms
by H. T. Gordon
The author shows how to use the information provided by the digits of large numbers to quickly determine some things about divisibility.

18 B-Tree ISAM Concepts
by Chris Deppe and Alan Bartholomew
How do you retrieve random information quickly from a large disk file? The authors discuss the principles and techniques of an elegant method called a B-Tree.

22 CP/M BDOS and BIOS Calls for C
by Terje Bolstad
Many C compilers for CP/M either omit or severely limit the ability to make direct calls to the BDOS and BIOS. The author describes two functions which will allow you to incorporate such capability into your own C programs.

32 Printing Graphics on the IBM PC
by Dan Deetwyler
Earlier versions of PCDOS do not provide direct support for combining a "dot graphics" printer with the Color/Graphics Adapter. The author bridges the gap with this mapping utility.

42 Game of LIFE on the IBM PC
by Simson Garfinkel
While a number of implementations of Conway’s Game of Life have appeared over the years, these were usually slow. This version for the IBM PC provides both speed (up to 2.7 generations per second) and easy entry and tracking of the cells.

56 Serial Expansion in Forth
by Gordon King
In his search for a fast floating-point language, the author found an ideal in Forth. In this first of a three-part series, he presents some fundamental Forth matrix utilities.

60 Yes, You can Trace Through BDOS
by Do-White Jones
They said it couldn’t be done. . . . Taking up the challenge from our Letters column, the author shows how you can get a trace through your BDOS.

66 Julian Dates for Microcomputers
by Gordon King
Use of a Julian date system allows dates to be stored on computers as integers, but there must be a way to convert between Julian and normal calendar notation. The author presents routines for implementing such a system on your micro.

DEPARTMENTS

9 Dr. Dobb’s Clinic

12 16-Bit Software Toolbox

80 CP/M Exchange

83 Of Interest

95 Advertiser Index
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.

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 lower 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

by Simson L. Garfinkel

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

---

**THE 488+3**

**IEEE 488 TO S-100 INTERFACE**

- Handles all IEEE-488 1975/78 functions
- IEEE 696 (S-100) compatible
- MBASIC subroutines supplied; no BIOS mods required
- 3 parallel ports (8255A-5)
- Industrial quality; burned in and tested
- $375

**D&W DIGITAL**

20655 Hathaway Ave.
Hayward, CA 94541
415/ 887-5711

Circle no. 23 on reader service card.
Game of Life  (Text begins on page 42)


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

; code1 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
    js 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
    js main ;nope - loop back
    mov ah,0 ;get the character our of the buffer
    int 16h
    cmp al,TERM
    jnz s2 ;finished - go back to ms/dos
    ret

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

el: 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

jmp ah,go_left ; make a rational decision about the users
jmp ah,right ; entry.
jmp ah,up
jmp ah,down
jmp ah,point
jmp ah,del

jmp ah,esc ; loop back - unknown command

mov dx,23*256 ; put the cursor at lower-left hand corner
mov ah,2
int 10h ; go back to caller

ret

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

mov dl,0
sub dl,1
jmp el

jmp el

; move right if I can.

cmp dl,79
jz el
add dl,1
jmp el

jmp el

; move up if I can.

cmp dh,0
jz el
sub dh,1
jmp el

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

```assembly
go_down:                           ; go down if I can
    cmp dh, 24
    js el
    add dh, 1
    jmp el

go_point:                         ; put a live dot where the cursor is -- don't move it
    mov ax, live
    ; it's the live character
    jmp gp2

gp2:                               ; one character to write
    mov cx, 1
    ; code to write character
    int 10h
    jmp el

channel go__point do the rest
```

```assembly
Enter endp
```

```assembly
Cl s 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
```

```assembly
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
```

```assembly
chk macro yy, xx
    local chl, offs
offs equ (xx+yy*80)*2
    mov cx, [di+offs]
    cmp cl, live
    inz chl
    add al, 1
chl 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-around

cl: 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 ;ax:=(dh*80+dl)*2
add ax,cx

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
jmp c2

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

give_death:
    mov ch,dark

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
mov ax, OB000H
mov ds, ax

mov bx, 24*80*2-2

u1: mov cx, [bx]
cmp ch, rev
jnz u2
mov cl, live
jmp u3

u2: mov cl, dead

u3: mov ch, dark
mov [bx], cx
sub bx, 2
jg u1
ret

Update endp
cseg ends

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

end

---

Tired of The High Priced Spread?
try the
Poor Person's Spread Sheet

A Programmable Calculator is combined with a Screen Design System to yield a powerful spread sheet substitute.

Features:
- User defined fields eliminate square look, interactive screen definition.
- Simple programming language uses individual fields as variables. No row, column or group operations.
- 280 fields, 200 step program with conditional statements and branches.
- Online documentation and debug aid.
- 8 digit precision, elementary math functions.
- Pre-programmed applications include Real Estate Investment Evaluation, Check Book Balance, Calculator, Loan Payments and more.

Requirements:
- 56K CP/M 2.2 or 1.4 with one disk
- 80 x 24 video terminal with UP, DOWN, RIGHT, LEFT cursor HOME, CLEAR.

$29.95 [California residents add 6½ sales tax]. Manual $4.00
AVAILABLE ON 8 IN. IBM SD, NORTHSTAR DD OTHER FORMATS, INQUIRE
Alan Bomberger
POOR PERSON SOFTWARE
3721 Starr King Circle
Palo Alto, CA 94306

CP/M is a trademark of Digital Research

---