|  |
| --- |
| Microprocessor  And Assembly Language  CSE-2202 |
| **Project Report** |
| Submitted to:  Dr. Rumana Nazmul  Ms. Anna Fariha  M. Tawfiqul Islam |

Submitted by:

Sharmin Mahzabin (03)

Melody Soptaka (43)

**Contents:**

-Name of project

-How we got the idea

-Platform

-Work-plan

-Description of the game

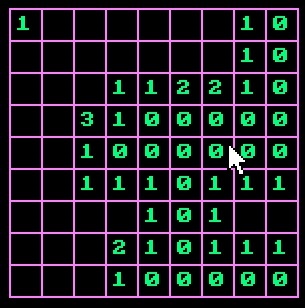
-How to play

-Macros and procedures in the code

-Source code

Presenting…

**Bomb Alert!!**



Bomb alert: The idea

In the broad arena of computer programming, most prefer contest programming as their field of interest. But we all know, traditional coding despite of being a huge part, there are parts like software sectors and graphics sectors. As this project focuses on our interest for graphics sector, no matter how much reluctant we were, we had to think about it.

And the challenging part was doing it in Assembly language.

One simple note about our game is, it’s a basic and traditional Minesweeper game. We changed the name for variation.

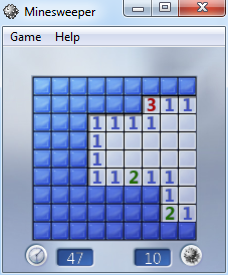
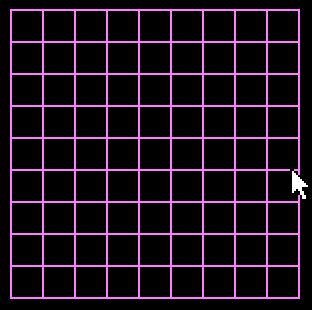


Figure 1: Original minesweeper game

In this game a 9\*9 grid is introduced which contains 81 cells.



A cell may be empty or contain a number or a mine. The goal is to discover all the cells that don’t contain bombs. If the player clicks on a cell containing a bomb, he/she loses.

PLATFORM:

We wrote our code in Assembly language. We made this game as our assembly lab project.

Work-plan…

When ma’am asked for the proposals, we didn’t have a plan. We thought about doing something too simple. After submitting the proposal, we both were quite reluctant about making the game. But again when the defense day was fixed, we started working all over. Even this simple game seemed a lot!

Then we decided to work by parts. We learnt about some functions which we didn’t even touch in the assembly lab. Actually we didn’t get much of a chance of learning graphics portion of assembly in the lab. So, we were all by ourselves actually.

The Game…

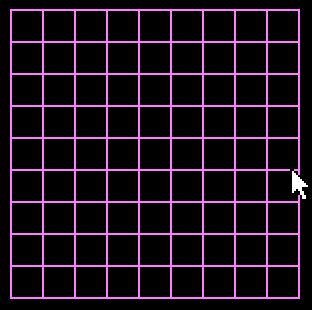


Fig. A 9x9 board in which no cell has been discovered

Some of the cells in the 9x9 board contains bombs.

The goal of this game is to discover all the cells that don’t contain bombs.



Fig. After winning the game

If a cell that contains a bomb is clicked, the game is over.

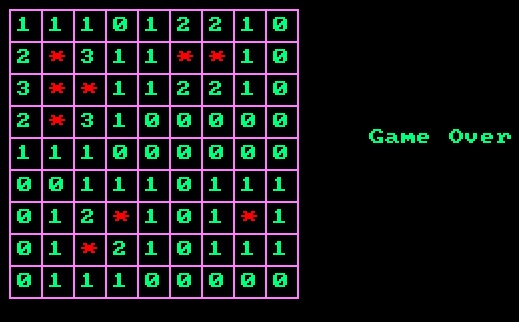


Fig. After the game is over

A cell may be empty. An empty cell indicates that there is no bomb in the cells which are adjacent to it. When an empty cell is clicked, all the connected cells to it that don’t contain any bomb is discovered automatically.

Rest of the cells contain positive numbers. This numbers indicate how many of the eight adjacent cells, contain bombs.

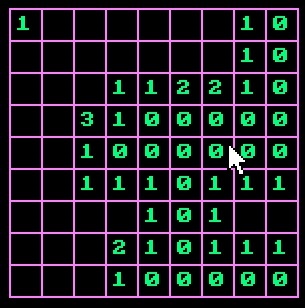


Fig. After clicking some cells

How to play:

The player can simply click on a cell to discover it. The first click may be on a random cell. If the player is lucky, the game continues. One can guess which cells may contain bombs by the numbers written on the cells which have already been discovered.

MACROS and PROCEDURES:

**saveReg MACRO:**

This macro pushes the register values to the stack.

**recoverReg MACRO:**

This macro pops register values from the stack.

**rctoInd MACRO:**

This macro converts row and columns of the grid to the index of the array which stores the values or mine for each cell.

Index = (9 x row) + column

**indtoRowCol MACRO:**

This macro takes index of an array as parameter and finds which row and column of the grid is indicated by this index.

**rtoCurY MACRO:**

This macro converts the row of the grid to the row of cursor for writing text.

**ctoCurX MACRO:**

This macro converts the column of the grid to the column of cursor for writing text.

**Setboard MACRO:**

This macro selects some cells to put mines and for the other cells, it computes how many adjacent cells of the corresponding cell contains mines and saves the number in a global array.

**pixtoRow procedure:**

This is a procedure which calculates the row of the grid from a pixel of the screen.

**pixtoCol procedure:**

This is a procedure which calculates the column of the grid from a pixel of the screen.

**dfs procedure:**

This procedures finds the adjacent cells of an empty cell recursively by using Depth First Search algorithm. When a cell containing a number is reached, this function immediately returns, otherwise it keeps finding adjacent cells recursilvely.

Program Code:

.model small

;.stack 100h

;org 100h ; set location counter to 100h

.data

xStart dw 20 ; x coordinate of line start

yStart dw 20 ; y coordinate of line start

length dw 160 ; length of line

incr dw 16

endl dw 180

r db 0

c db 0

cnt db 0

curx db 0

cury db 0

node dw 0

x dw 0

y dw 0

cell dw 0

board dw 200 dup ('0')

visit dw 200 dup ('0')

over db "Game Over$"

win db "You Won!!$"

;Macro to push the register values to the stack

saveReg MACRO

push ax

push bx

push cx

push dx

ENDM

;Macro to pop the register values from the stack

recoverReg MACRO

pop dx

pop cx

pop bx

pop ax

ENDM

;convert row and column to index

rctoInd MACRO

saveReg

mov ax,0

mov al,r

mov bl,9

mul bl

add al,c

mov cell,ax

shl cell,1

recoverReg

ENDM

;convert cell index to row and column

indtoRowCol MACRO ind

saveReg

mov al,ind

mov ah,0

mov bl,8

div bl

mov r,al

mov c,ah

recoverReg

ENDM

;convert grid Row to Cursor row

rtoCurY MACRO

saveReg

mov al,r

shl al,1

mov cury, al

add cury,3

recoverReg

ENDM

ctoCurX MACRO

saveReg

mov al,c

shl al,1

mov curx, al

add curx,3

recoverReg

ENDM

;macro to set board

setboard MACRO

saveReg

mov board[20],'\*'

mov board[28],'\*'

mov board[30],'\*'

mov board[38],'\*'

mov board[40],'\*'

mov board[56],'\*'

mov board[114],'\*'

mov board[122],'\*'

mov board[130],'\*'

;mov board[72],'\*'

mov cl,0

lea si,board

boardlp:

cmp [si],'\*'

JE check1

jmp check9

check1:

mov di,si

sub di,2

cmp [di],'\*'

je check2

add [di],1

check2:

mov di,si

add di,2

cmp [di],'\*'

je check3

add [di],1

check3:

mov di,si

sub di,18

cmp [di],'\*'

je check4

add [di],1

check4:

mov di,si

add di,18

cmp [di],'\*'

je check5

add [di],1

check5:

mov di,si

sub di,20

cmp [di],'\*'

je check6

add [di],1

check6:

mov di,si

sub di,16

cmp [di],'\*'

je check7

add [di],1

check7:

mov di,si

add di,16

cmp [di],'\*'

je check8

add [di],1

check8:

mov di,si

add di,20

cmp [di],'\*'

je check9

add [di],1

check9:

add si,2

add cl,1

cmp cl,81

jne boardlp

recoverReg

ENDM

.code

main proc

mov ax,@data

mov ds,ax

mov es,ax

; set the video mode 320x200, 256 colors

mov al, 13h

mov ah, 0

int 10h

;SET BOARD

setboard

; initialize cx (x coord) to xStart + length

mov cx, xStart

add cx, length

sub cx,incr

mov dx, yStart

; loop from (xStart+length) to xStart to draw a horizontal line

LoopStart:

; draw a pixel

; set color in al, x in cx, y in dx

mov al, 60

; set sub function value in ah to draw a pixel

; and invoke the interrupt

mov ah, 0ch

int 10h

; decrement the x coord

sub cx, 1

; test to see if x coord has reached start value

cmp cx, xStart

; continue loop if cx >= xStart

jae LoopStart

jmp changey

changey:

add dx,incr

mov cx, xStart

add cx, length

sub cx,incr

cmp dx,endl

jl LoopStart

CodeStart2:

; set the video mode 320x200, 256 colors

; initialize cx (x coord) to xStart + length

mov cx, xStart

mov dx, yStart

add dx, length

sub dx,incr

; loop from (xStart+length) to xStart to draw a horizontal line

LoopStart2:

; draw a pixel

; set color in al, x in cx, y in dx

mov al, 60

; set sub function value in ah to draw a pixel

; and invoke the interrupt

mov ah, 0ch

int 10h

; decrement the x coord

sub dx, 1

; test to see if x coord has reached start value

cmp dx, yStart

; continue loop if cx >= xStart

jae LoopStart2

jmp changex

changex:

add cx,incr

mov dx, yStart

add dx, length

sub dx,incr

cmp cx,endl

jl LoopStart2

mov ax,00h

int 33h

mov ax,01h

int 33h

lea si,board

mov cnt,0

trackMouse:

mov ax,03h

int 33h

cmp bx,1

jne trackMouse

shr cx,1

cmp cx,20

jl trackMouse

cmp cx,164

jg trackMouse

cmp dx,20

jl trackMouse

cmp dx,164

jg trackMouse

mov x,dx

call pixtoRow

mov x,cx

call pixtoCol

rtoCurY

ctoCurX

mov dl,curx

mov dh,cury

mov ax,02h

int 33h

mov ah, 02h

int 10h ; interrupt to set cursor position

rctoInd

mov si,cell

mov ax, [board+si] ; char

cmp ax,'\*'

je gameOver

cmp ax,'0'

jne printOneCell

mov ax,cell

mov node,ax

mov ax,[board+si]

call dfs

jmp checkwin

printOneCell:

mov bh, 0 ; layer

mov bl, 50 ; color

mov cx, 1 ; repeatation

mov ah, 09h

int 10h

cmp visit[si],'0'

jne hide

mov visit[si],'1'

add cnt,1

checkwin:

cmp cnt,72

jge gameover

hide:

mov ax,01h

int 33h

jmp trackMouse

gameOver:

; PRINT CHAR

mov dl, 3 ; set cursor position [ x ]

mov dh, 3 ; set cursor position [ y ]

lea si,board

printChar:

mov ah, 02h

int 10h ; interrupt to set cursor position

mov al, [si] ; char

mov bh, 0 ; layer

mov bl, 50 ; color

cmp al,'\*'

jne print

mov bl,40

print:

mov cx, 1 ; repeatation

mov ah, 09h

int 10h

add dl,2

inc si

inc si

cmp dl,19

jbe printChar

mov dl,3

add dh,2

cmp dh,19

jbe printChar

printmsg:

mov ax,02

int 33h

mov ah,13h ;SERVICE TO DISPLAY STRING WITH COLOR.

lea si, over ;STRING TO DISPLAY.

cmp cnt,72

jl msg

lea si,win

msg:

mov dl,24 ;X (SCREEN COORDINATE).

mov dh,10 ;Y (SCREEN COORDINATE).

mov r,0

lpmsg:

mov dh,10

add dl,1

mov ah,02h

int 10h

mov al, [si] ; char

mov bh, 0 ; layer

mov bl, 50 ; color

mov cx,1

mov ah,09h

int 10h

add si,1

add r,1

cmp r,9

jl lpmsg

mov ax,4ch

int 21h

;ret

main endp

;convert pixel to row and column

pixtoCol proc

saveReg

mov c,0

cmp x,36

jle lastCol

add c,1

cmp x,52

jle lastCol

add c,1

cmp x,68

jle lastCol

add c,1

cmp x,84

jle lastCol

add c,1

cmp x,100

jle lastCol

add c,1

cmp x,116

jle lastCol

add c,1

cmp x,132

jle lastCol

add c,1

cmp x,148

jle lastCol

add c,1

cmp x,164

jle lastCol

lastCol:

recoverReg

ret

pixtoCol endp

;convert pixel to row

pixtoRow proc

saveReg

mov r,0

cmp x,36

jle last

add r,1

cmp x,52

jle last

add r,1

cmp x,68

jle last

add r,1

cmp x,84

jle last

add r,1

cmp x,100

jle last

add r,1

cmp x,116

jle last

add r,1

cmp x,132

jle last

add r,1

cmp x,148

jle last

add r,1

cmp x,164

jle last

last:

recoverReg

ret

pixtoRow endp

proc dfs

saveReg

mov dl,curx

mov dh,cury

mov ax,02h

int 33h

mov ah, 02h

int 10h ; interrupt to set cursor position

;rctoInd

mov bx,node

mov ax, [board+bx] ; char

mov bh, 0 ; layer

mov bl, 50 ; color

mov cx, 1 ; repeatation

mov ah, 09h

int 10h

mov ax,01h

int 33h

mov bx,node

cmp [visit+bx],'0'

jne compare

mov [visit+bx],'1'

add cnt,1

compare:

mov bx,node

cmp [bx+board],'0'

je calculate

recoverReg

ret

calculate:

up:

cmp node,18

jl right

mov bx,node

add bx,-18

cmp [visit+bx],'0'

jne right

sub node,18

add cury,-2

call dfs

add node,18

add cury,2

right:

mov ax,node

mov bl,18

div bl

cmp ah,16

je down

mov bx,node

add bx,2

cmp [visit+bx],'0'

jne down

add node,2

add curx,2

call dfs

add node,-2

add curx,-2

cmp node,18

jl down

mov bx,node

add bx,-16

cmp [visit+bx],'0'

jne down

sub node,16

add cury,-2

add curx,2

call dfs

add node,16

add curx,-2

add cury,2

down:

cmp node,144

jge left

mov bx,node

add bx,18

cmp [visit+bx],'0'

jne left

add node,18

add cury,2

call dfs

add node,-18

add cury,-2

left:

mov ax,node

mov bl,18

div bl

cmp ah,0

je ul

mov bx,node

add bx,-2

cmp [visit+bx],'0'

jne ul

sub node,2

add curx,-2

call dfs

add node,2

add curx,2

cmp node,144

jge ul

mov bx,node

add bx,16

cmp [visit+bx],'0'

jne ul

add node,16

add cury,2

add curx,-2

call dfs

add node,-16

add curx,2

add cury,-2

ul:

mov ax,node

mov bl,18

div bl

cmp ah,0

je dr

cmp node,18

jl dr

mov bx,node

add bx,-20

cmp [visit+bx],'0'

jne dr

add node,-20

add cury,-2

add curx,-2

call dfs

add node,20

add curx,2

add cury,2

dr:

mov ax,node

mov bl,18

div bl

cmp ah,16

je enddfs

cmp node,144

jge enddfs

mov bx,node

add bx,20

cmp [visit+bx],'0'

jne enddfs

add node,20

add cury,2

add curx,2

call dfs

add node,-20

add curx,-2

add cury,-2

enddfs:

recoverReg

ret

dfs endp

end main