University of Dhaka

Department of Computer Science & Engineering

2nd year 2nd Semester

**Project Report**

CSE-2212: Microprocessor and Assembly Language Lab

Project Name : **Minesweeper**

Submitted by:

Jargis Ahmed, Roll – 04

Pritom Saha, Roll – 56

Submitted to:

Dr. Rumana Nazmul

Anna Fariha

Muhammad Tawfiqul Islam

# Table Of Contents:

**1. Intrduction** 3

**2. Game Outline** 3

**3. Game Flow Chart** 4

**4. Navigational Layout** 5

**5. Sample Instructions** 5

**6. Used Interrupts** 5-8

**7.** **Screenshots** 9-13

**8. Program Coding**…………..……….…………………………………………………………………14-47

**9. Limitations :** 48

**10. Conclusion:** 48

**11. Reference:** 48

**Minesweeper**

**Introduction:**

Minesweeper is one of the most popular online and offline games of the world. The game originates from the 1960s, and has been written for many computing platforms in use today. It has many variations and offshoots. Minesweeper, a little game that debuted back in 1990 as part of the WINDOWS Entertainment Pack, before being promoted to a standard feature in WINDOWS 3.1 and onwards.

Minesweeper is a single player puzzle video game. The objective of the game is to clear a rectangular board containing hidden “mines” without detonating any of them, with help from clues about the number of neighboring mines in each field.

Inspired by one of the most famous games of all time, Minesweeper, we reproduced this game using assembly language. Despite the limitations and complexity of assembly language, we tried to emulate the features of the original game without slowing the game down.

**Game Outline:**

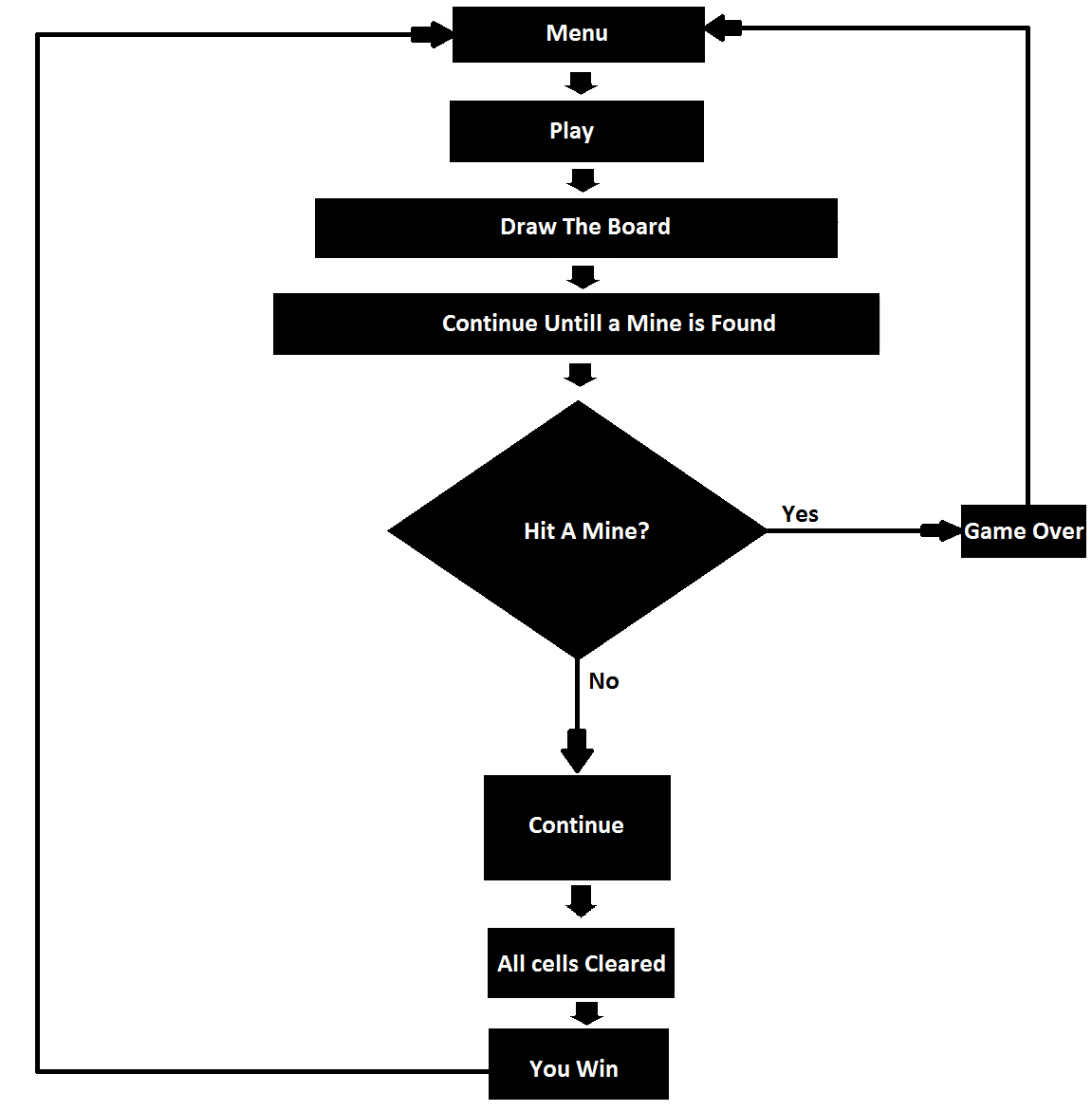
The player is initially presented with a grid of undifferentiated squares. Some randomly selected squares, unknown to the player, are designated to contain mines. Typically, the size of the grid and the number of mines are set in advance by the user, either by entering the numbers or selecting from defined skill levels, depending on the implementation.

The game is played by revealing squares of the grid by clicking or otherwise indicating each square. If a square containing a mine is revealed, the player loses the game. If no mine is revealed, a digit is instead displayed in the square, indicating how many adjacent squares contain mines; if no mines are adjacent, the square becomes blank, and all adjacent squares will be recursively revealed. The player uses this information to deduce the contents of other squares, and may either safely reveal each square or mark the square as containing a mine.

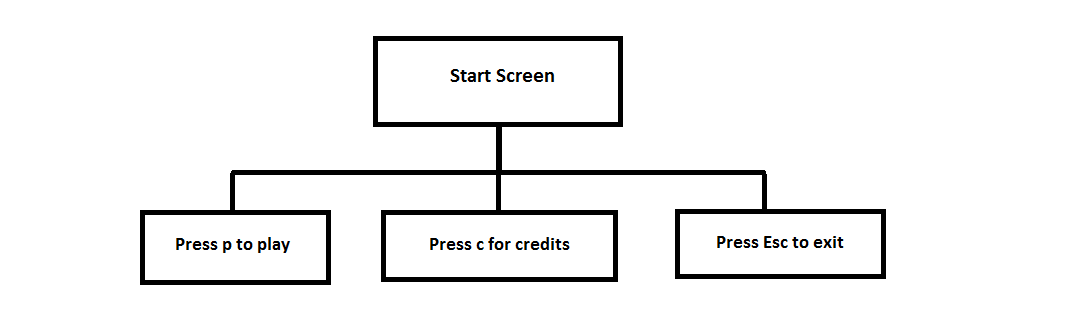
In some versions, a question mark may be placed in an unrevealed square to serve as an aid to logical deduction. Implementations may also allow players to quickly "clear around" a revealed square once the correct number of mines have been flagged around it. The game is won when all mine-free squares are revealed, because all mines have been located.

Some versions of Minesweeper will set up the board by never placing a mine on the first square revealed, or by arranging the board so that the solution does not require guessing.

**Game Flow Chart:**

****

**Navigational Layout:**

****

**Sample Instructions:**

The following is the instructions that we have used widely to build this program. These all are the 8086 simple instructions. All of these have helped us to perform the desired task. Since all the registers are 16-bit, all the instructions work on 16- bit task.

mov – copy word or byte to destination from source

jmp - used to perform an unconditional jump

call - call a procedure and save return address on stack

cmp - compare two bytes or words.

jge - jump if greater than or equal

jg - jump if greater than

jle - jump if less than or equal

lea - load effective address

equ - creates the equivalent of a named constant

inc- increment the bytes or words

dec- decrement the bytes or words

ret- to return from procedure

**Used Interrupts:**

**Int 10h:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function**  **Number** | **Function** | | **Input** | | **Output** |
| 1 | Set text-mode cursor shape | | CH = cursor start line (bits 0-4) and options (bits 5-7). CL = bottom cursor line (bits 0-4). | | when bit 5 of CH is set to 0, the cursor is visible. when bit 5 is 1, the cursor is not visible. |
| 2 | Set cursor position | | DH = row. DL = column. BH = page number (0..7). | | - |
| 3 | Get cursor position and size. | | BH = page number. | | DH = row. DL = column. CH = cursor start line. CL = cursor bottom line. |
| 5 | Select active video page. | | AL = new page number (0..7). | | the activated page is displayed. |
| 8 | Read character and attribute at cursor position. | | BH = page number. | | AH=attribute AL = character. |
| 9 | Write character and attribute at cursor position. | | AL = character to display. BH = page number. BL = attribute CX = number of times to write character. | | - |
| 13h | | Write string | | AL = write mode BH = page number. BL = attribute if string contains only characters (bit 1 of AL is zero). CX = number of characters in string DL,DH = column, row at which to start writing. ES:BP points to string to be printed. | |

**Int 16h:**

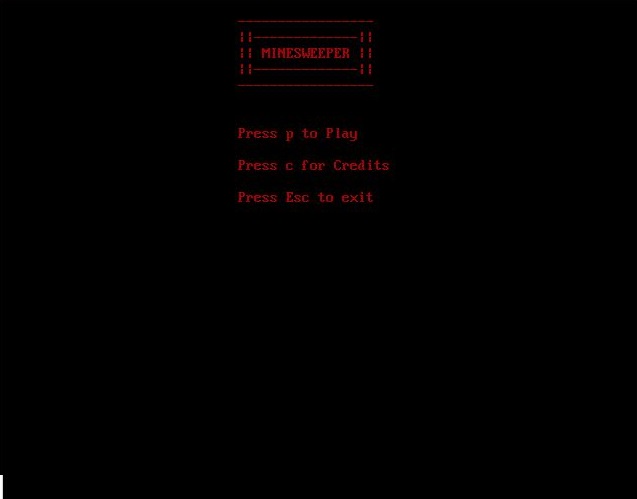
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Function**  **Number** | | **Function** | | | **Input** | | | **Output** | |
| 0 | | Get keystroke from keyboard | | | - | | | AH = BIOS scan code. AL = ASCII character. | |
| 1 | | Check for keystroke in the keyboard buffer. | | | - | | | ZF = 1 if keystroke is not available. ZF = 0 if keystroke available. AH = BIOS scan code. AL = ASCII character. | |
| **Int 21h:**  **Function**  **Number** | | **Function** | | | **Input** | | | **Output** | |
| 1 | | read character from standard input, with echo | | | - | | | result is stored in AL. | |
| 2 | | write character to standard output. | | | DL = character to write | | | AL = DL. | |
| 9 | | | output of a string at DS:DX. | | | DS:DX-string to print | | |
| 2ch | get system time; | | | - | | | CH = hour.  CL = minute.  DH = second.  DL = 1/100 seconds. | |
| 3ch | create file | | | CX = file | | | CF clear if successful, AX = file handle. | |
| 3dh | open existing file | | | AL = access and sharing modes; DS:DX -> ASCIZ filename. | | | CF clear if successful, AX = file handle. | |
| 3eh | close file | | | BX = file handle | | | CF clear if successful, AX destroyed. | |
| 3fh | read from file | | | BX = file handle CX = number of bytes to read. DS:DX -> buffer for data. | | | CF is clear if successful - AX = number of bytes actually read;  0 if at EOF before call. | |
| 40h | Write file | | | BX = file handle. CX = number of bytes to write. DS:DX -> data to write. | | | CF clear if successful;  AX = number of bytes actually written. CF set on error;  AX = error code. | |
| 42h | Seek- set current file position. | | | AL = origin of move: 0 - start of file. 1 - current file position. 2 - end of file. BX = file handle CX:DX = offset from origin of new file position. | | | CF clear if successful,  DX:AX = new file position in bytes from start of file. CF set on error,  AX = error code. | |
| 4ch | return control to operating system | | | - | | | - | |

**Int 1ah:**

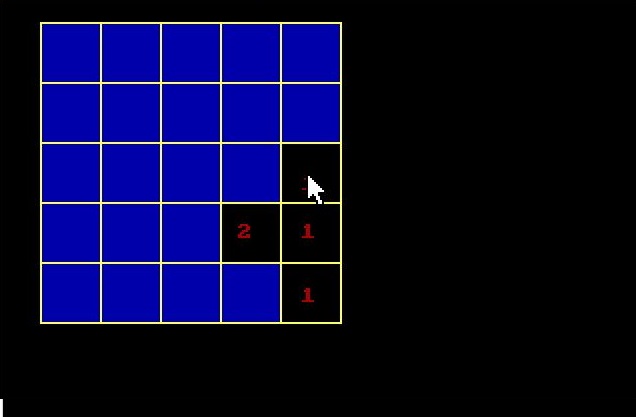
|  |  |  |  |
| --- | --- | --- | --- |
| **Function**  **Number** | **Function** | **Input** | **Output** |
| 0 | Get system time | - | CX:DX = number of clock ticks since midnight. AL = midnight counter, advanced each time midnight passes. |
| 2 | Read real clock time | - | CH=Hour  CL=Minute  DH=Second  DL=Daylight Savings |

**Screenshots:**

**1. Menu Screen**

****

**2. Playing the Game**

****

**3. Hitting Mine**

****

**4. Winning the Game**

****

**5. Credits**

****

**Source Code:**

.model small

.stack 100

.data

row dw 51,81,111,141,171

col dw 41,71,101,131,161

elr db 4,8,12,15,19

elc db 3,7,11,14,18

elem db '\*','1','1','1','1','1','2','2','\*','1','1','2','\*','2','1','\*','2','2','2','1','1','1','1','\*','1','$'

rv dw 0

cv dw 0

clv dw 0

oldX dw -1

oldY dw 0

mr db 0

mc db 0

break db 0

vis db '0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','$'

dis db 0

n\_cell db 0

m db 'MinesWeeper $'

p db 'Press p to Play $'

c db 'Press c for Credits $'

J1 db '-----------------$'

J2 db '||-------------||$'

J3 db '|| MINESWEEPER ||$'

J4 db '||-------------||$'

J5 db '-----------------$'

GoBack db 'Press g to return to the Main Menu.$'

c1 db ' Jargis Ahmed $'

c2 db ' 20th Batch $'

c3 db ' Roll - 04 $'

c4 db ' Pritom Saha $'

c5 db ' 20th Batch $'

c6 db ' Roll - 56 $'

c7 db 'Department of Computer Science & Engineering$'

uni db 'University of Dhaka$'

kala db ' $'

ywin db 'Winner ',3,3,3,' ',1,'$'

game db ' Game Over$'

ak db 'Press any key$'

exit db 'Press Esc to exit$'

boom db ' Booooom!!$'

cdval db 0

;line db 100,120

.code

main proc

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

mov ax,@data

mov ds,ax ;

mov es,ax ; ;

;

start:

mov ah, 0

mov al, 12h ;

int 10h

mov n\_cell,0

mov cl,25

lea si,vis

mov ch,'0'

set\_0:

mov [si],ch

inc si

dec cl

cmp cl,0

jne set\_0

mov dh, 1 ; ;

mov dl, 30 ;

lea si, J1 ;

call print

mov dh, 2

mov dl, 30

lea si, J2

call print

mov dh, 3

mov dl, 30

lea si, J3

call print ;;;;;;; minesweeper

mov dh, 4

mov dl, 30

lea si, J4

call print

mov dh, 5

mov dl, 30

lea si, J5

call print

mov dh, 8

mov dl, 30

lea si, p

call print

mov dh, 10

mov dl, 30

lea si, c ;

call print ;

mov dh,12

mov dl,30

lea si,exit

call print

wait\_for\_key:

mov ah, 0h

int 16h

cmp al, 'p' ; esc?

je new\_game

cmp al, 'c'

je cred

cmp al,27

je stop

jne wait\_for\_key

stop:

mov ax, 3 ; back to text mode: 80x25

int 10h

;;;;;;;;;;;;;;;;;;; ending main proc ;;;;;;;;;;;;;;;;;;;;

mov ah,4ch

int 21h

main endp

new\_game:

;call kala\_banao

;lea si, m

;mov dh,1

;mov dl,1

;call print

jmp new\_start

cred:

mov dh, 6

mov dl, 30

lea si, c1

call print1

mov dh, 7

mov dl, 30

lea si, c2

call print1

mov dh, 8

mov dl, 30

lea si, c3

call print1

mov dh, 10

mov dl, 30

lea si, c4

call print1

mov dh, 11

mov dl, 30

lea si, c5

call print1

mov dh, 12

mov dl, 30

lea si, c6

call print1

mov dh, 14

mov dl, 17

lea si, c7

call print1

mov dh, 15

mov dl, 29

lea si, uni

call print1

mov cdval,0

mov dh, 17

mov dl, 22

lea si, GoBack

call print

sec\_w\_f\_k:

mov ah, 0h

int 16h

cmp al, 'g'

je j\_start

jne sec\_w\_f\_k

j\_start:

mov ax,12h

int 10h

;call kala\_banao ;

jmp start ;

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;

;;;;;;;;;;;;;;;;;;;;;;;;;;;; ;;;

new\_start:

;set graphics mode

mov ah,0

mov al,13h

int 10h

;;;;;;;;;;;;;;;;;; row 1 ;;;;;;;;;;;;;;;

mov bx,11

mov rv,bx

;cell no. 1

mov cv,21

mov clv,50

x1:

;mov rv,bx

call back\_print

inc rv

cmp rv,41

jne x1

;cell no. 2

add cv,30

mov rv,11

add clv,30

x2:

call back\_print

inc rv

cmp rv,41

jne x2

;cell no. 3

add cv,30

mov rv,11

add clv,30

x3:

call back\_print

inc rv

cmp rv,41

jne x3

;cell no. 4

add cv,30

mov rv,11

add clv,30

x4:

call back\_print

inc rv

cmp rv,41

jne x4

;cell no. 5

add cv,30

mov rv,11

add clv,30

x5:

call back\_print

inc rv

cmp rv,41

jne x5

;;;;;;;;;;;;;;;;;;;;; row 2 ;;;;;;;;;;;;;;;;;;;

mov bx,41

mov rv,bx

;cell no. 1

mov cv,21

mov clv,50

x6:

call back\_print

inc rv

cmp rv,71

jne x6

;cell no. 2

add cv,30

mov rv,41

add clv,30

x7:

call back\_print

inc rv

cmp rv,71

jne x7

;cell no. 3

add cv,30

mov rv,41

add clv,30

x8:

call back\_print

inc rv

cmp rv,71

jne x8

;cell no. 4

add cv,30

mov rv,41

add clv,30

x9:

call back\_print

inc rv

cmp rv,71

jne x9

;cell no. 5

add cv,30

mov rv,41

add clv,30

x10:

call back\_print

inc rv

cmp rv,71

jne x10

;;;;;;;;;;;;;;;;;; row 3 ;;;;;;;;;;;;;;;;;;;

mov bx,71

mov rv,bx

;cell no. 1

mov cv,21

mov clv,50

x11:

call back\_print

inc rv

cmp rv,101

jne x11

;cell no. 2

add cv,30

mov rv,71

add clv,30

x12:

call back\_print

inc rv

cmp rv,101

jne x12

;cell no. 3

add cv,30

mov rv,71

add clv,30

x13:

call back\_print

inc rv

cmp rv,101

jne x13

;cell no. 4

add cv,30

mov rv,71

add clv,30

x14:

call back\_print

inc rv

cmp rv,101

jne x14

;cell no. 5

add cv,30

mov rv,71

add clv,30

x15:

call back\_print

inc rv

cmp rv,101

jne x15

;;;;;;;;;;;;;;;;;; row 4 ;;;;;;;;;;;;;;;;;

mov bx,101

mov rv,bx

;cell no. 1

mov cv,21

mov clv,50

x16:

call back\_print

inc rv

cmp rv,131

jne x16

;cell no. 2

add cv,30

mov rv,101

add clv,30

x17:

call back\_print

inc rv

cmp rv,131

jne x17

;cell no. 3

add cv,30

mov rv,101

add clv,30

x18:

call back\_print

inc rv

cmp rv,131

jne x18

;cell no. 4

add cv,30

mov rv,101

add clv,30

x19:

call back\_print

inc rv

cmp rv,131

jne x19

;cell no. 5

add cv,30

mov rv,101

add clv,30

x20:

call back\_print

inc rv

cmp rv,131

jne x20

;;;;;;;;;;;;;;;; row 5 ;;;;;;;;;;;;;;;

mov bx,131

mov rv,bx

;cell no. 1

mov cv,21

mov clv,50

x21:

call back\_print

inc rv

cmp rv,161

jne x21

;cell no. 2

add cv,30

mov rv,131

add clv,30

x22:

call back\_print

inc rv

cmp rv,161

jne x22

;cell no. 3

add cv,30

mov rv,131

add clv,30

x23:

call back\_print

inc rv

cmp rv,161

jne x23

;cell no. 4

add cv,30

mov rv,131

add clv,30

x24:

call back\_print

inc rv

cmp rv,161

jne x24

;cell no. 5

add cv,30

mov rv,131

add clv,30

x25:

call back\_print

inc rv

cmp rv,161

jne x25

;;;;;;;;;;;;;;;;;; printing row line ;;;;;;;;;;;;;;;;;;

mov ch,0

mov cl,6h

mov bx,11

for\_rl\_p:

call rl\_print

add bx,30

dec cl

cmp cl,0h

;inc di

jg for\_rl\_p

;;;;;;;;;;;;;;;;;; printing column line ;;;;;;;;;;;;;;

mov ch,0

mov cl,6h

mov bx,21

for\_cl\_p:

call cl\_print

add bx,30

dec cl

cmp cl,0h

;inc di

jg for\_cl\_p

;////////////////////////////////////////////////////////////

int 3h

;;;;;;;; use of mouse;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

check\_mouse\_button:

;push bx

mov ax,1 ;;;;;; show mouse cursor

int 33h

mov ax, 3 ;;;;;;;;;;position and clicked or not clicked

int 33h

shr cx, 1 ; x/2 cx is doubled ah=13h,int 10h 320\*200 video mode

;mov cx,40 ;; cx is row val

;mov dx,50 ;; dx is col val of pixel

cmp bx, 1;;;;;;;;;;;;;; click of left mouse button

;je point\_draw

je find\_cell

cmp bx,2

je place\_qm

jmp check\_mouse\_button

check\_any\_key:

win:

mov dh,9

mov dl,23

lea si,ywin

call print

mov dh,11

mov dl,23

lea si,ak

call print1

mov n\_cell,0

mov ah, 0h

int 16h

jmp start

place\_qm:

mov ah,2

int 33h

cmp dx,11

jl check\_mouse\_button

cmp dx,161

jg check\_mouse\_button

cmp cx,21

jl check\_mouse\_button

cmp cx,171

jg check\_mouse\_button

call quest\_mk

jmp check\_mouse\_button

find\_cell:

mov al,-1

mov ah,-1

lea si,row

lea di,col

cmp dx,11

jl check\_mouse\_button

cmp dx,161

jg check\_mouse\_button

cmp cx,21

jl check\_mouse\_button

cmp cx,171

jg check\_mouse\_button

count\_row:

inc ah ;;inc row no.

mov bx,si

add si,2

cmp [si-2],cx

jl count\_row

count\_col:

inc al

mov bx,di

add di,2

cmp [di-2],dx

jl count\_col

;push ax

mov mr,ah

mov mc,al

mov bx,[si-2]

mov rv,bx

;sub rv,10

sub bx,29

;suv rv,10

mov cx,[di-2]

sub cx,29

mov cv,cx

mov clv,cx

add clv,28

push ax

mov ax,02 ;;;;; hide mouse cursor

int 33h

run\_black:

call black\_the\_cell

inc bx

cmp bx,rv

jne run\_black

point\_draw:

pop ax

push dx

push cx

;push bx

lea di,elr

lea si,elc

inc\_si: ;;find row val

cmp ah,0

je inc\_di

inc di

dec ah

jmp inc\_si

inc\_di: ;;find col val

cmp al,0

je st\_draw

inc si

dec al

jmp inc\_di

st\_draw:

mov dl,[di] ;;;set col

mov dh,[si] ;;;set row

;push dx

;xchg dh,dl

mov al,mc

mov cl,5

mul cl

mov cl,'0' ;;; to check vis or not vis

add al,mr

mov dis,al

lea di,elem

lea si,vis ;;;;;;;;; make n-th visited

add si,ax

cmp [si],cl

je counter

resume\_point\_draw:

mov cl,'1'

mov [si],cl

;mov al,'1'

add di,ax

mov al,[di]

;pop dx

mov bh,0 ; \\//same page\\//

mov ah,02 ;set cursor

int 10h

mov ah,9h ;write char func

mov cx, 1 ;how many char

mov bl,0100b ; color of char wanted

int 10h

cmp al,'\*'

je set\_break

;jne counter

pop cx

pop dx

cmp n\_cell,20

je jmp\_check\_any\_key

;pop bx

;pop ax

jmp check\_mouse\_button

set\_break:

pop cx

pop dx

;mov break,1

mov dh,8

mov dl,23

lea si,boom

call print

mov dh,10

mov dl,23

lea si,game

call print

mov dh,12

mov dl,23

lea si,ak

call print1

mov ah, 0h

int 16h

jmp start

counter:

inc n\_cell

jmp resume\_point\_draw

jmp\_check\_any\_key:

jmp check\_any\_key

;;;;;;;;;;;;;;;;;;;;;; 25 cells ;;;;;;;;;;;;;;;;;;;;;;;;;

back\_print proc

;draw line

push cx

mov ah,0ch

mov al,1

mov cx,cv

mov dx,rv

L:

int 10h

inc cx

cmp cx,clv

jle L

pop cx

ret

back\_print endp

;;;;;;;;;;;;;;;;;;;;;;;; yellow row line ;;;;;;;;;;;;;;;;;

rl\_print proc

;draw line

push cx

mov ah,0ch

mov al,1110B

mov cx,21

mov dx,bx

L1:

int 10h

inc cx

cmp cx,170

jle L1

pop cx

ret

rl\_print endp

;;;;;;;;;;;;;;;;;;;;;;; yellow column line ;;;;;;;;;;;;;;;;

cl\_print proc

;draw line

push cx

mov ah,0ch

mov al,1110B

mov cx,bx

mov dx,11

LL2:

int 10h

inc dx

cmp dx,161

jle LL2

pop cx

ret

cl\_print endp

;;;;;;;;;;;;;;;;;;;;;;;; blacking the clicked cell ;;;;;;;;;;;

black\_the\_cell proc

push ax

mov ah,0ch

mov al,0

mov cx,bx

mov dx,cv

blk:

int 10h

inc dx

cmp dx,clv

jle blk

pop ax

ret

black\_the\_cell endp

quest\_mk proc

;push bx

mov al,-1

mov ah,-1

lea si,row

lea di,col

count\_row\_qm:

inc ah ;;inc row no.

mov bx,si

add si,2

cmp [si-2],cx

jl count\_row\_qm

count\_col\_qm:

inc al

mov bx,di

add di,2

cmp [di-2],dx

jl count\_col\_qm

lea di,elr

lea si,elc

inc\_si\_qm: ;;find row val

cmp ah,0

je inc\_di\_qm

inc di

dec ah

jmp inc\_si\_qm

inc\_di\_qm: ;;find col val

cmp al,0

je st\_draw\_qm

inc si

dec al

jmp inc\_di\_qm

st\_draw\_qm:

mov dl,[di] ;;;set col

mov dh,[si] ;;;set row

mov bh,0 ; \\//same page\\//

mov ah,02 ;set cursor

int 10h

mov al,'?'

mov ah,9h ;write char func

mov cx, 1 ;how many char

mov bl,0100b ; color of char wanted

int 10h

;pop cx

;pop dx

;jmp check\_esc\_key

ret

quest\_mk endp

print proc

push cx

LP:

mov bh, 0 ; \\//same page\\//

mov ah, 02 ;set cursor

int 10h

mov al, [si]

cmp al, '$'

je break\_print

mov ah, 9h ;write char func

mov cx, 1 ;how many char

mov bl, 0100b ; color of char wanted

int 10h

inc si

inc dl

jmp LP

break\_print:

pop cx

ret

print endp

print1 proc

push cx

LP1:

mov bh, 0 ; \\//same page\\//

mov ah, 02 ;set cursor

int 10h

mov al, [si]

cmp al, '$'

je break\_print1

mov ah, 9h ;write char func

mov cx, 1 ;how many char

mov bl, 0101b ; color of char wanted

int 10h

inc si

inc dl

jmp LP1

break\_print1:

pop cx

ret

print1 endp

end main

**Limitations:**

1. It’s very hard to debug a code while doing these kind of works. It may take someone hours to find a small bug in the code. It took us a whole day just to find out a small bug while doing mouse-related works.

2. Coding in assembly language is a hard task. It is very hard to code these games/apps in assembly.

3. Assembly language has it’s limitations. The running speed is just too slow. It takes a lot of time just to draw a single line.

4. It takes a lot of time just to write a simple code.

5. The size of the Minesweeper board is considered for only 5x5 square. It can be made dynamic for the user

**Conclusion:**

Despite the limitations, it was great fun doing the project of making one of the most played games Minesweeper. Although it was our initial plan to make the game more dynamic and to make the 5x5 square more random but due to the shortage of time and limitations of the assembly language it was hard to do those things. But the game is competitive enough to play again and again. This is our first project. We tried our best to make the game like the original one.

It has been a memorable experience to make our own game and it has been far more special because it is the first project we ever did using assembly language. We've learnt a lot from this experience and we look forward to implement our experience in the future projects.

**References:**

This project has been made according to the suggestions of our honorable teachers and is inspired by the classic "Minesweeper". We used Ytha Yu and Charles Marut's book on 'Assembly Language Programming and Organization of IBM PC' to learn about the interrupts and other features of assembly language .Online resources also helped us in some regards.

We would specially thank our honorable teachers - Dr.Rumana Nazmul madam, Anna Fariha madam and Tawfiqul Islam sir, for their continuous guidance and suggestions.