

פרויקט מסכם באסמבלי

שם המגיש: רועי מלמד

ת.ז של המגיש: 215982257

שם הפרויקט: משחק שולה המוקשים (Minesweeper)

שם בית הספר: תיכון "הדרים", הוד השרון (סמל מוסד: 441006)

שם המורה: ענבל רגב

תאריך: 19/06/2022

תוכן עניינים

סביבת העבודה.....	3
תיאור שימוש היישום	4
משתנים	7
פונקציות	11
gameFrame	11
printBoard	12
printNumbersUpTheBoard	14
printLineInBoard	16
setMines	18
addNumber	19
checkLeft, checkRight	20
initNumbers	20
inputToIndex	22
Input.....	24
revealCell.....	25
convertRowCol	26
checkWin.....	27
gameOverP, winGameP.....	28
openFile, readHeader, readPalette, copyPal, copyBitmap.....	30
playCellSound.....	33
askMine.....	34
game	35
runGame	37
אלגוריתם הפרויקט – תרשים זרימה	38
הפרויקט בשבילי.....	39
קוד הפרויקט	40

סביבת העבודה

Assembly

DOSBOX 0.74 - TASM

תיאור שימוש היישום

שולה המוקשים

תיאור כללי:

תחילה יוצג למשתמש מסך פתיחה ובו שם המשחק, בעת לחיצה על כפתור כלשהו במקלדת יוצגו לשחקן חוקי המשחק.

לאחר שהשחקן ילחץ עוד פעם על כפתור במקלדת יתחיל המשחק. במשחק מצויר הלוח 8X8 ועשרה מוקשים מפוזרים באופן רנדומלי בו, בתאים שאינם מוקשים מפוזרים מספרים המייצגים כמה מוקשים יש סביב אותו התא, כך ש"_" מייצג תא ריק שאין סביבו אף מוקש.

על פי המספרים המביאים אינפורמציה על כל תא ועל פי שימוש בשיטת האלימינציה (השלילה) ולוגיקה בסיסית ינסה המשתמש לנצח במשחק על ידי פתיחת התאים במתן מידע על כל תא שירצה לפתוח, שורה ועמודה. ניצחון נחשב כאשר המשתמש פותח את כל התאים בהם אין מוקש, כאשר עשה זאת יוצג למשתמש מסך המורה על ניצחונו. אם הפסיד, כלומר פתח תא ובו מוקש יוצג למשתמש מסך המורה לו על כך שהפסיד.

חוקים:

לוח המשחק הוא לוח מרובע, המחולק למשבצות בגודל 8X8. בלוח מפוזרים "מוקשים" בלתי נראים במיקומים אקראיים. הלוח מתחלק ל-2 סוגי משבצות:

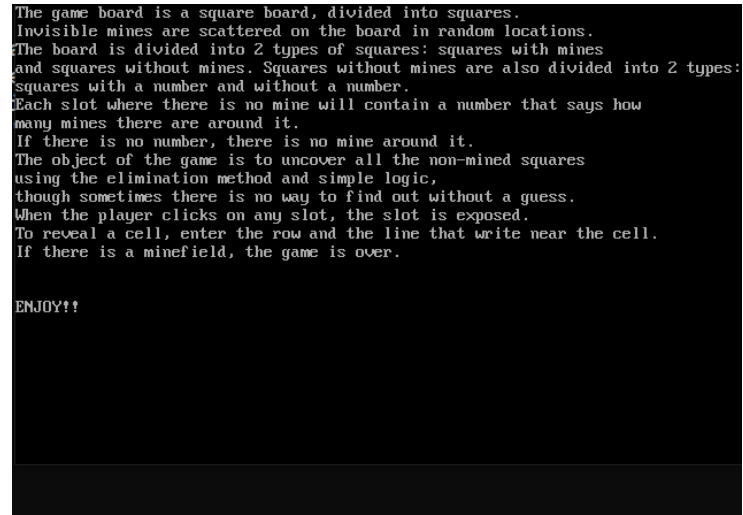
משבצות עם מוקשים ומשבצות ללא מוקשים.

משבצות ללא מוקשים מתחלקות גם הן ל-2 סוגים:

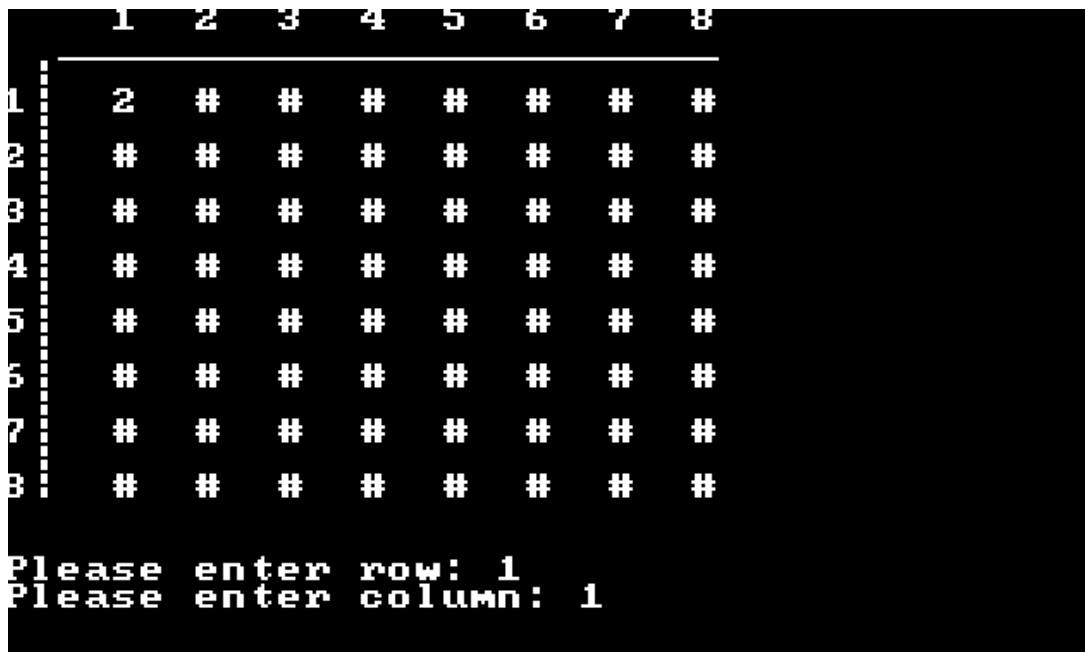
משבצות עם מספר ובלי מספר. כל משבצת שבה אין מוקש תכיל מספר האומר כמה מוקשים יש סביבו. אם אין מספר, תיפתח קבוצה של משבצות - כל המשבצות הריקות עד לגבול שבו יש משבצת עם מספר, כולל המשבצות עם המספר. מטרת המשחק היא לחשוף את כל המשבצות נטולות המוקשים תוך כדי שימוש בשיטת האלימינציה ובלוגיקה פשוטה, אם כי לעיתים אין דרך לגלות ללא ניחוש.

כאשר השחקן לוחץ על משבצת כלשהי, המשבצת נחשפת. אם יש במשבצת מוקש, המשחק נגמר.

תמונות של הצגת חוקי המשחק ושמו:



תמונה של הצגת מהלך המשחק (לוח המשחק):



תמונת מצב ניצחון:




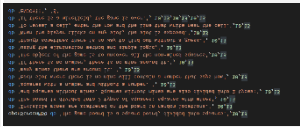
תמונת מצב הפסד:



משתנים

שם משתנה	גודל	ערך התחלתי	הסבר
board	db	64 dup(2dh)	מערך לוח התשובות של המשחק – בו מאותחלים כל המוקשים, המספרים והתאים הריקים
revealBoard	db	64 dup(23h)	מערך לוח המשחק שהמשתמש רואה, המשתמש מבקש לפתוח תא, אותו תא נבדק במערך התשובות, נחשף במערך זה ומודפס
fileLose	db	'LOSE.bmp'	שם קובץ התמונה המוצגת לאחר הפסד
fileWin	db	'WIN.bmp'	שם קובץ התמונה המוצגת לאחר ניצחון
filehandle	dw	0	מחזיק את ערך הקובץ לאחר פתיחתו

מערך המייצג את 54 הבתים בקובץ התמונה	54 dup (0)	db	Header
מערך המייצג את 54 הבתים בצבעים כל בית מיוצג ב-4 בתים (54 בתים X 4 צבעים)	256*4 dup (0)	db	Palette
משתנה עזר לקריאת שורה בתמונה	320 dup (0)	db	ScrLine
הודעת שגיאה במידה והקובץ לא נפתח כראוי	'ERROR', 10, 13	db	ErrorMsg
מחזיק מידע 0 או 0.1 – כאשר נרצה לפתוח את תמונת הניצחון. 1 – כאשר נרצה לפתוח את תמונת ההפסד	0	db	isOpenFileLose
הודעת בקשה למשתמש להכנסת שורה	'Please enter row: ', '\$'	db	inputRowMSG
הודעת בקשה למשתמש להכנסת עמודה	'Please enter column: ', '\$'	db	inputColMSG
הודעת מסך הפתיחה		db	gameFrameMSG

הודעת חוקי המשחק		db	openScreenMSG
הודעה להדפסת שורה חדשה	'\$,10,13,'	db	newLineMSG
הודעה להדפסת רווח	' \$'	db	spaceMSG
הודעת בקשה מהמשתמש למיקום מוקש לבחירתו	'Place mine at place that you want', 10,13, '\$'	db	askMineMSG
הודעה השואלת את המשתמש אם ירצה רמה יותר קלה בכך שהוא יבחר מקום שבו יהיה מוקש	'Do you want an easier game level?',10,13 db '(1 - yes. any key - no): ', '\$'	db	easierlevelMSG
מחזיק מידע אם המשתמש ניצח. 0 - אם לא, 1 - אם כן	0	db	isWin
מחזיק מידע אם המשתמש הפסיד. 0 – אם לא, 1 – אם כן	0	db	isGameOver
מחזיק מידע באיזו שורה המשתמש רוצה לבצע שינוי	0	db	inputRow

מחזיק מידע באיזו עמודה המשתמש רוצה לבצע שינוי	0	db	inputCol
מחזיק מידע לאחר המרה משורה ועמודה לאינדקס בלוח	0	db	indexAfterCon
מחזיק מידע על קלט לאחר שנבדק שהוא תקין	0	db	validInput
מונה לצורך הדפסת הלוח	0	db	cnt
תדר הצליל שמושמע בעת חשיפת תא	0a98h	db	frequency

פרוצדורות

gameFrame

קוד הפרוצדורה:

```
proc gameFrame
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; set the resolution to (640 x 480) with 256 colors
    mov ax, 4F02h
    mov bx, 101h
    int 10h

    ; print game frame screen
    mov dx, offset gameframeMSG
    mov ah, 9h
    int 21h

    ; wait for key press
    mov ah, 0h
    int 16h

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; print opening screen
    mov dx, offset openScreenMSG
    mov ah, 9h
    int 21h

    ; wait for key press
    mov ah, 0h
    int 16h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    popa
    ret
```

```
endp gameFrame
```

הסבר:

הפרוצדורה מדפיסה את מסכי הפתיחה, מנקה את הDOS ומחליפה למצב גרפי.

משתנים:

gameframeMSG – לצורך הדפסת שם המשחק המורכב מתווי אסקי.

printBoard

קוד הפרוצדורה:

```
proc printBoard

    push bp
    mov bp, sp
    pusha

    call printnumbersuptheboard

    ;Inserts the values from the stack into registers
    mov bx, [bp + 4] ;board
    mov cx, [bp + 6] ;board length
    printArrayLoop:

    mov al, [byte ptr bx]
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    ;-----
    mov ax, cx
    dec ax
    mov dl, ROW_COL_LEN
    div dl
```

```

cmp ah, 0
je printLine
jmp notPrintLine
;-----

printLine:
call printLineInBoard

notPrintLine:
inc bx
loop printArrayLoop

mov dx, offset newlineMSG
mov ah, 9h
int 21h

popa
pop bp
ret 4

endp printBoard

```

הסבר:

הפרוצדורה מדפיסה את לוח המשחק, עוברת על המערך המייצג את הלוח ומדפיסה את ערכיו עם רווחים וירידת שורות בין כל ערך. בנוסף נעזרת בפרוצדורת עזר ומדפיסה את מספר כל שורה ועמודה.

משתנים:

board – המערך להדפסה.

spaceMSG – לצורך הדפסת רווח בין תאים.

newLineMSG – לצורך הדפסת שורות בין התאים.

printNumbersUpTheBoard

קוד הפרוצדורה:

```
proc printNumbersUpTheBoard
    pusha

    mov [cnt], 31h
    xor cx, cx
    mov cl, ROW_COL_LEN

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h
numLoop:

    mov al, [cnt]
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    inc [cnt]
    loop numloop

    mov dx, offset newlineMSG
    mov ah, 9h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    mov al, '_'
    mov dx, ax
    mov ah, 2h
    int 21h

    mov al, '_'
    mov dx, ax
    mov ah, 2h
    int 21h
```

```

xor cx, cx
mov cl, 22
numLoop2:

mov al, '_'
mov dx, ax
mov ah, 2h
int 21h

loop numloop2

mov dx, offset newlineMSG
mov ah, 9h
int 21h

mov al, ' '
mov dx, ax
mov ah, 2h
int 21h

mov al, '|'
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset newlineMSG
mov ah, 9h
int 21h

mov [cnt], 31h
mov al, [cnt]
mov dx, ax
mov ah, 2h
int 21h

inc [cnt]

mov al, '|'
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset spaceMSG
mov ah, 9h

```

```

int 21h

popa
ret
endp printNumbersUpTheBoard

```

הסבר:

פרוצדורת עזר לצורך הדפסת המספרים המסמנים את העמודות.

משתנים:

spaceMSG – לצורך הדפסת רווח בין תאים.

newLineMSG – לצורך הדפסת שורות בין התאים.

cnt – מונה לצורך ספירת העמודות

printLineInBoard

קוד הפרוצדורה:

```

proc printLineInBoard
pusha

mov dx, offset newLineMSG
mov ah, 9h
int 21h

cmp [cnt], 39h
je enddddd

mov al, ' '
mov dx, ax
mov ah, 2h
int 21h

mov al, '|'
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset newLineMSG

```



```

mov ah, 9h
int 21h

mov al, [cnt]
mov dx, ax
mov ah, 2h
int 21h

inc [cnt]

mov al, '|'
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset spaceMSG
mov ah, 9h
int 21h

enddddd:
popa
ret
endp printLineInBoard

```

הסבר:

פרוצדורת עזר לצורך הדפסת המספרים המסמנים את השורות.

משתנים:

spaceMSG – לצורך הדפסת רווח בין תאים.

newLineMSG – לצורך הדפסת שורות בין התאים.

cnt – מונה לצורך ספירת השורות

setMines

קוד הפרוצדורה:

```
proc setMines
  pusha

  mov cl, NUMBER_OF_MINE
  setMinesRndloop:
    mov ah, 2Ch
    push cx ;preserve value of cx
    int 21h ;dl stores milliseconds
    pop cx
    xor ax, ax
    mov al, dl
    mov bl, BOARD_LEN
    div bl ;ah stores the remainder
    mov bx, offset board
    add bl, ah
    cmp [byte ptr bx], MINE ;is there a bomb already?
    je setMinesRndloop
    ; avoid the 4 center cells
    cmp bl, 27
    je setMineRndloop
    cmp bl, 28
    je setMinesRndloop
    cmp bl, 35
    je setMinesRndloop
    cmp bl, 36
    je setMinesRndloop
    mov [byte ptr bx], MINE

    loop setMinesRndloop
  popa
  ret
endp setMines
```

הסבר:

הפרוצדורה מגרילה "מוקשים" באופן אקראי ומכניסה אותם ללוח המשחק. במרכז הלוח התוכנית דואגת שלא יהיו מוקשים כדי לדמות את כך שבמשחק האמיתי לאחר הלחיצה הראשונה בטוח אין מוקש.

משתנים:

board – הלוח בו שמים את המוקשים.

addNumber

קוד הפרוצדורה:

```
proc addNumber
    add bx, offset board
    add bx, cx
    cmp [byte ptr bx], MINE ;is there a mine?
    je ADDNUM_continue

    cmp [byte ptr bx], EMPTY_IN_BOARD
    je isEmpty
    add [byte ptr bx], 1
    jmp ADDNUM_continue

isEmpty:
    mov [byte ptr bx], ONE_ASCII

ADDNUM_continue:
    ret
endp addNumber
```

הסבר:

Bx מחזיק ערך המייצג אינדקס לתא אליו נרצה להוסיף 1 מכיוון שכנראה לידו יש מוקש. אם בתא יש ערך ריק נאתחל אותו ל-1 באסקי כדי שיודפס 1 ואם נוסף מספרים הם יודפסו גם כן.

משתנים:

board – הלוח לצורך גישה לאינדקס בו.

checkLeft, checkRight

קוד הפרוצדורות:

```
proc checkLeft
    mov ax, cx
    mov bl, 8
    div bl
    cmp ah, 0
    ret
endp checkLeft

proc checkRight
    mov ax, cx
    mov bl, 8
    div bl
    cmp ah, 7
    ret
endp checkRight
```

הסבר:

באס מוחזק ערך האינדקס שרצים עליו, בקריאת הפרוצדורה היא בודקת האם קיים איבר משמאל או מימין לאינדקס. אם לא קיים איבר משמאל שארית החלוקה של האינדקס ב-7 (אורך שורה ועמודה פחות אחד) היא 0 אז השיוויון מתקיים. אם לא קיים איבר מימין שארית החלוקה של האינדקס ב-8 (אורך שורה ועמודה) היא 0 אז השיוויון מתקיים.

משתנים:

אין.

initNumbers

קוד הפרוצדורה:

```
proc initNumbers
    pusha

    mov cx, 0
MAP_loop:
    mov bx, offset board
    add bx, cx
    cmp [byte ptr bx], MINE ;is there a mine?
    jne MAP_continue
```

```

        ; check for free space above
        cmp cl, 8
        jl MAP_left
        ; check for free space to the left
        call checkLeft
        je MAP_top
        ; add number
        mov bx, -9
        call addNumber
MAP_top:
        ; add number
        mov bx, -8
        call addNumber
MAP_topright:
        ; check for free space to the right
        call checkRight
        je MAP_left
        ; add number
        mov bx, -7
        call addNumber
MAP_left:
        ; check for free space to the left
        call checkLeft
        je MAP_right
        ; add number
        mov bx, -1
        call addNumber
MAP_right:
        ; check for free space to the right
        call checkRight
        je MAP_bottomleft
        ; add number
        mov bx, 1
        call addNumber
MAP_bottomleft:
        ; check for free space below
        cmp cl, 55
        jg MAP_continue
        ; check for free space to the left
        call checkLeft
        je MAP_bottom
        ; add number
        mov bx, 7
        call addNumber
MAP_bottom:
        ; add number
        mov bx, 8
        call addNumber

```

```

MAP_bottomright:
    ; check for free space to the right
    call checkRight
    je MAP_continue
    ; add number
    mov bx, 9
    call addNumber
MAP_continue:
    inc cx
    cmp cx, BOARD_LEN
    jne MAP_loop

    popa
    ret
endp initNumbers

```

הסבר:

הפרוצדורה עוברת על הלוח ומעדכנת סביב כל תא שהוא מוקש ומוסיפה לכל תא כזה 1. עד שמסיימת לעבור על כל המערך והוא יוצא מעודכן לפי חוקי המשחק.

משתנים:

board – לוח המשחק לעדכון מספרים בתאים שהם לא מוקש.

inputToIndex

קוד הפרוצדורה:

```

proc inputToIndex
    pusha

    mov dx, offset newLineMSG
    mov ah, 9
    int 21h

    mov dx, offset inputRowMSG
    mov ah, 9
    int 21h

    call input
    mov al, [validInput]

```

```

mov [inputRow], al
dec [inputRow]

mov dx, offset newLineMSG
mov ah, 9
int 21h

mov dx, offset inputColMSG
mov ah, 9
int 21h

call input
mov al, [validInput]
mov [inputCol], al
dec [inputCol]
call convertRowCol

popa
ret

endp inputToIndex

```

הסבר:

הפרוצדורה מבקשת מהמשתמש קלט על שורה ועמודה, כאשר אם הוא מכניס שורה ועמודה שלא קיימת היא לא עוברת הלאה ולא מאשרת לו את הקלט, רק כאשר הקלט תקין הפונקציה מדפיסה אותו למסך ושומרת את האינדקס המייצג את השורה ואת העמודה במערך במשתנה לאחר ההמרה.

משתנים:

newLineMSG - מחרוזת להדפסת שורה חדשה בין הודעות.

inputColMSG – הודעה לבקשת עמודה מהמשתמש.

inputRowMSG – הודעה לבקשת שורה מהמשתמש.

Input

קוד הפרוצדורה:

```
proc input
    pusha

    checkInputt:
    mov ah, 08h
    int 21h
    sub al, '0'

    cmp al, 0
    je checkInputt

    cmp al, 8
    ja checkInputt
    jmp successCheck

    successCheck:

    mov [validInput], al

    add [validinput], '0'

    mov dl,[validinput]
    mov ah,2
    int 21h

    sub [validinput], '0'

    popa
    ret
endp input
```

הסבר:

פרוצדורת עזר לקליטת השורה והעמודה מהמשתמש. פרוצדורה זו קולטת את את הקלט ומבצעת בו את בדיקת השגיאות ולא משחררת את הקלט את שמוכנס קלט תקין.

משתנים:

validInput – מחזיק את הקלט התקין לאחר שעבר מתו אסקי למספר.

revealCell

קוד הפרוצדורה:

```
proc revealCell
    pusha

    mov si, offset revealBoard
    mov di, offset board

    add si, [indexAfterCon]
    add di, [indexAfterCon]

    cmp [byte ptr di], MINE
    je mineCell

    notMineCell:
    mov al, [byte ptr di]
    mov [byte ptr si], al
    jmp endd

    mineCell:
    mov [isGameOver], 1
    jmp endd

    endd:
    popa
    ret
endp revealCell
```

הסבר:

חושף את התא שהוכנס בקלט בלוח להדפה, אם התא עם מוקש, נסמן זאת בעזרת משתנה ולאחר מכן נחשוף הודעת הפסד. אם התא לא מוקש הוא יחשוף.

משתנים:

revealBoard - הלוח בו יחשף התא.

board – הלוח ממנו לוקחים את הערך לחשיפה.

indexAfterCon – האינדקס אשר נחשוף בערך שלו את התא.

isGameOver – מחזיק מידע האם האינדקס מחזיק ערך של מוקש.

convertRowCol

קוד הפרוצדורה:

```
proc convertRowCol
    pusha

    xor ax, ax
    mov al, ROW_COL_LEN
    mul [inputRow]
    add al, [inputCol]
    mov [indexAfterCon], ax

    popa
    ret
endp convertRowCol
```

הסבר:

ממיר את הערכים במשתנים של שורה ועמודה לאינדקס במערך המשחק.

משתנים:

inputRow – האינדקס של השורה.

inputCol – האינדקס של העמודה.

indexAfterCon – האינדקס לאחר ההמרה.

checkWin

קוד הפרוצדורה:

```
proc checkWin
    push bp
    mov bp, sp
    pusha

    ;Inserts the values from the stack into registers
    mov si, [bp + 4] ;board
    mov di, [bp + 6] ;reavel board
    mov cx, [bp + 8]

    xor bx, bx
    checkWinLoop:
    mov al, [di + bx]
    cmp al, 23h ;Check if is a cell that not opened
    je checkMine
    jmp continueLoop

    checkMine:
    mov al, [si + bx]
    cmp al, 2ah
    jne notWin

    continueLoop:
    inc bx
    loop checkWinLoop
    jmp win

    win:
    mov [iswin], 1
    jmp enddd

    notWin:
    mov [iswin], 0

    enddd:
    popa
    pop bp
    ret 6
endp checkwin
```

הסבר:

הפּרוּצדורה בודקת האם המשתמש ניצח. עוברת על כל הלוח ובודקת האם התאים שעוד לא נפתחו ישנם רק מוקשים. אם בתא כלשהו שעוד לא נפתח לא קיים מוקש המשתמש אינו ניצח והמשתנה המייצג ניצחון מתעדכן בהתאם. אם הוא ניצח כלומר מקיים את הכלל מעלה המשתנה המייצג ניצחון מתעדכן גם כן בהתאם.

משתנים:

board - לוח התשובות, בו נבדוק האם קיים מוקש בתא שעוד לא נפתח.
revealBoard – הלוח להדפסה, בו נבדוק את התאים שעוד לא נחשפו.
isWin – מייצג האם המשתמש ניצח או לא.

gameOverP, winGameP

קוד הפרוצדורות:

```
proc gameOverP
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov [isOpenFileLose], 1
    call OpenFile
    call ReadHeader
    call ReadPalette
    call CopyPal
    call CopyBitmap

    ; wait for key press
    mov ah, 0h
    int 16h

    ; returns to text mode
    mov ah, 0h
    mov al, 03h
    int 10h

    popa
    ret
```

```

endp gameOverP

proc winGameP
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov [isOpenFileLose], 0
    call OpenFile
    call ReadHeader
    call ReadPalette
    call CopyPal
    call CopyBitmap

    ; wait for key press
    mov ah, 0h
    int 16h

    ; returns to text mode
    mov ah, 0h
    mov al, 03h
    int 10h

    popa
    ret
endp winGameP

```

הסבר:

הפרוצדורות נקראות כאשר המשתמש הפסיד או כאשר ניצח, פועלות באופן דומה. בהתאם למצבו, אם ניצח או הפסיד מוצגת תמונה המורה לו על כך ולאחר מכן נגמר המשחק.

משתנים:

isOpenFileLose – מחזיק מידע אם להציג את תמונת ההפסד או הניצחון.

openFile, readHeader, readPalette, copyPal, copyBitmap

קוד הפרוצדורות:

```
proc openFile
    mov ah, 3Dh
    xor al, al

    cmp [isOpenFileLose], 1
    je openLose
    jne openWin

    openLose:
    mov dx, offset fileLose
    jmp continueOpen

    openWin:
    mov dx, offset fileWin
    jmp continueOpen

    continueOpen:
    int 21h
    jc openerror
    mov [filehandle], ax

    ret
    openerror:
    mov dx, offset ErrorMessage
    mov ah, 9h
    int 21h
    ret
endp openFile

proc readHeader
    ; Read BMP file header, 54 bytes
    mov ah, 3fh
    mov bx, [filehandle]
    mov cx, HEADER_LEN
    mov dx, offset Header
    int 21h
    ret
endp readHeader

proc readPalette
    ; Read BMP file color palette, 256 colors * 4 bytes (400h)
    mov ah, 3fh
```

```

    mov cx, 400h
    mov dx, offset Palette
    int 21h
    ret
endp readPalette

proc copyPal
    ; Copy the colors palette to the video memory
    ; The number of the first color should be sent to port 3C8h
    ; The palette is sent to port 3C9h
    mov si, offset Palette
    mov cx, 256
    mov dx, 3C8h
    mov al, 0
    ; Copy starting color to port 3C8h
    out dx, al
    ; Copy palette itself to port 3C9h
    inc dx
PalLoop:
    ; Note: Colors in a BMP file are saved as BGR values rather than RGB.
    mov al, [si + 2] ; Get red value.
    shr al, 2 ; Max. is 255, but video palette maximal
    ; value is 63. Therefore dividing by 4.
    out dx, al ; Send it.
    mov al, [si + 1] ; Get green value.
    shr al, 2
    out dx, al ; Send it.
    mov al, [si] ; Get blue value.
    shr al, 2
    out dx, al ; Send it.
    add si, 4 ; Point to next color.
    ; (There is a null chr. after every color.)

    loop PalLoop
    ret
endp copyPal

proc copyBitmap
    ; BMP graphics are saved upside-down.
    ; Read the graphic line by line (200 lines in VGA format),
    ; displaying the lines from bottom to top.
    mov ax, 0A000h
    mov es, ax
    mov cx, 200
PrintBMPLoop:
    push cx
    ; di = cx*320, point to the correct screen line

```

```

mov di,cx
shl cx,6
shl di,8
add di,cx
; Read one line
mov ah,3fh
mov cx,320
mov dx, offset ScrLine
int 21h
; Copy one line into video memory
cld ; Clear direction flag, for movsb
mov cx,320
mov si, offset ScrLine

rep movsb ; Copy line to the screen

pop cx
loop PrintBMPLoop
ret
endp copyBitmap

```

הסבר:

פרוצדורות אשר מטפלות בקבצי תמונות, מפרקות אותן, הופכות לפיקסלים ומציגות אותן.

משתנים:

- fileLose – שם הקובץ של תמונת ההפסד.
- fileWin – שם הקובץ של תמונת הניצחון.
- Filehandle – מחזיק את התמונה המפורקת.
- Header – מערך המייצג את השורות בתמונה.
- Palette – מייצג את השורות עם צבעים.
- ScrLine – משתנה עזר להחזקת שורה בתמונה.
- ErrorMsg – הודעת שגיאה אם התמונה לא נפתחה כמו שצריך.
- isOpenFileLose – מחזיק מידע איזו תמונה לפתוח. ניצחון או הפסד.

playCellSound

קוד הפרוצדורה:

```
proc playCellSound
    pusha

    ; open speaker
    in al, 61h
    or al, 00000011b
    out 61h, al
    ; send control word to change frequency
    mov al, 0B6h
    out 43h, al
    ; play frequency
    mov ax, [frequency]
    out 42h, al ; Sending lower byte
    mov al, ah

    out 42h, al ; Sending upper byte

    ;delay the sound
    mov cx, 02H
    mov dx, 4240H
    mov ah, 86h
    int 15h

    ; close the speaker
    in al, 61h
    and al, 11111100b
    out 61h, al

    popa
    ret
```

הסבר:

משמיע את הצליל בתדר השמור במשתנה (התו לה) למשך כמה רגעים.

משתנים:

frequency – התדר להשמעה.

askMine

קוד הפרוצדורה:

```
proc askMine
    pusha

    mov dx, offset easierlevelMSG
    mov ah, 9h
    int 21h

    mov ah, 08h
    int 21h
    sub al, '0'

    cmp al, 1
    jne notAskMine

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov dx, offset askMineMSG
    mov ah, 9h
    int 21h

    call inputToIndex
    mov bx, offset board
    add bx, [indexAfterCon]
    mov [byte ptr bx], MINE

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

notAskMine:
    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
```

```

mov ax, 13h
int 10h

popa
ret
endp askMine

```

הסבר:

הפרוצדורה שואלת אם המשתמש מעוניין ברמת משחק קלה יותר בכך שהוא ממקם מוקש במקום בו הוא רוצה. אם לא המשחק ממשיך כרגיל. אם כן הפרוצדורה שואלת היכן ברצונו למקם את המוקש וממקת אותו שם.

משתנים:

easierlevelMSG – הודעה השואלת אם ברצונו של המשתמש משחק קל יותר.

askMineMSG – מבקש מהמשתמש מיקום למוקש.

board – לוח התשובות, בו ימוקם המוקש.

indexAfterCon – האינדקס המוכן לאחר דרישת הקלט בו ימוקם המוקש.

game

קוד הפרוצדורה:

```

proc game
pusha

mov dl, 0 ; Column
mov dh, 0 ; Row
mov bx, 0 ; Page number, 0 for graphics modes
mov ah, 2h
int 10h

push BOARD_LEN
push offset revealBoard
call printBoard

```

```

gameLoop:

push BOARD_LEN
push offset revealBoard
push offset board
call checkWin
cmp [iswin], 1
je winGame

call inputToIndex
call revealcell

cmp [isGameOver], 1
je gameOverr

mov dl, 0 ; Column
mov dh, 0 ; Row
mov bx, 0 ; Page number, 0 for graphics modes
mov ah, 2h
int 10h

push BOARD_LEN
push offset revealBoard
call printBoard
call playcellsound
jmp gameloop

gameOverr:
call gameOverP
jmp endddd

winGame:
call winGameP
jmp endddd

endddd:
popa
ret

endp game

```

הסבר:

הרצת המשחק והלולאה הראשית.

משתנים:

board – לוח התשובות.

revealBoard – הלוח להדפסה.

isWin – מחזיק מידע אם המשתמש ניצח.

isGameOver – מחזיק מידע אם המשתמש הפסיד.

runGame

קוד הפרוצדורה:

```
proc runGame
    pusha

    call gameFrame
    call askmine
    call setMines
    call initNumbers

    ;For check, you can remove the note and see the answer board
    ;#####
    ;push 64
    ;push offset board
    ;call printBoard
    ;#####

    call game

    popa
    ret
endp runGame
```

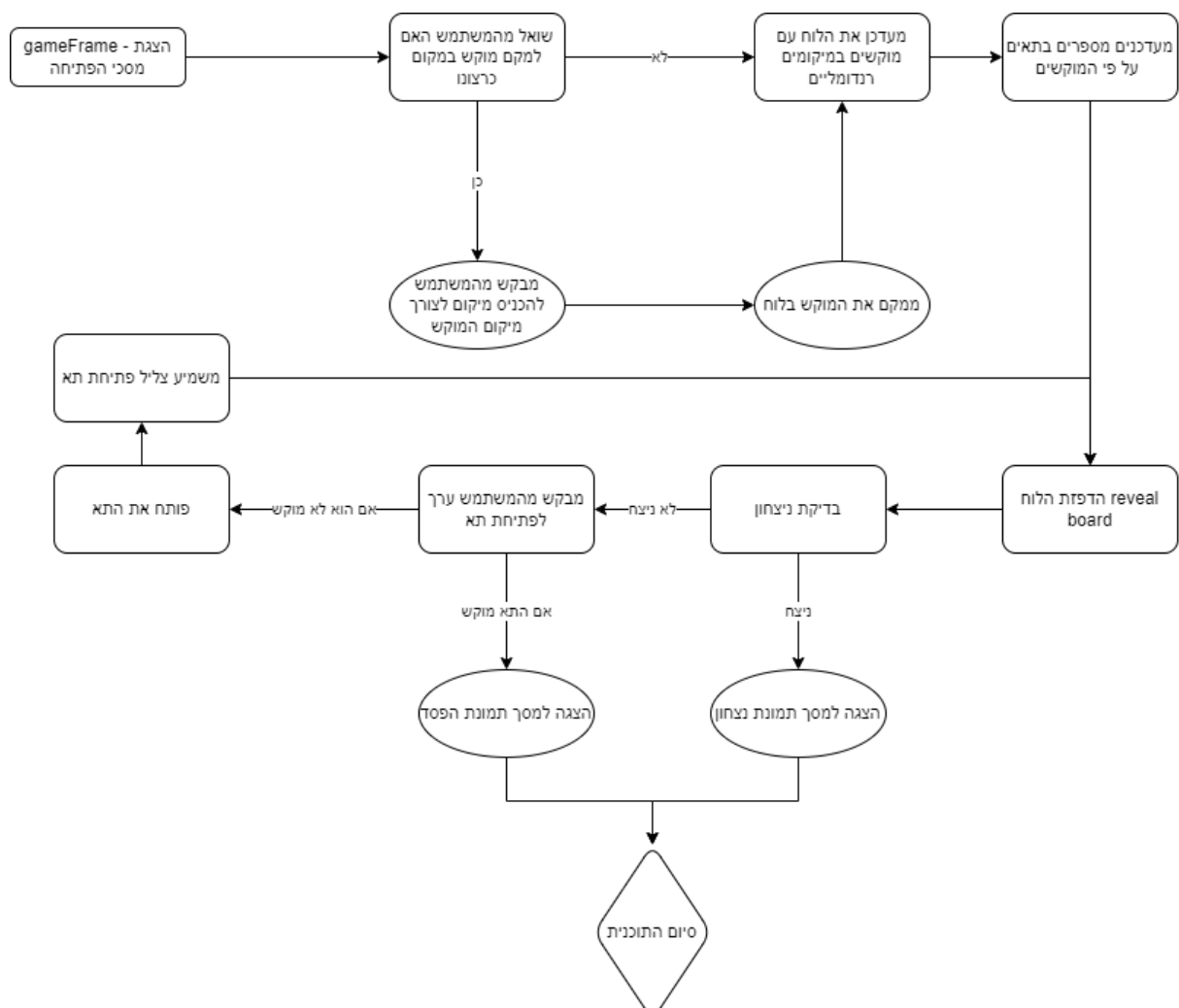
הסבר:

הרצת כל הפרוצדורות להפעלת המשחק: פיזור המוקשים, פיזור המספרים והפעלת לולאת המשחק הראשית.

משתנים:

אין.

אלגוריתם הפרויקט – תרשים זרימה



הפרויקט בשבילי

למה בחרתי בפרויקט זה

לפני תחילת העבודה חיפשתי באינטרנט רעיונות לפרויקטים, לא מצאתי הרבה, אך המשכתי לחפש וראיתי רעיון לבצע משחק שולה המוקשים, ראיתי שכדי לעשות את המשחק הזה מאוד מומלץ שימוש ברקורסיות. ידעתי רקורסיות כבר בכמה שפות ומאוד עניין אותי לממש את זה גם באסמבלי. לבסוף, לא השתמשתי (אפשרט על כך בקשיים) אך עדיין מאוד עניין אותי ביצוע הפרויט והלוגיקה מאחוריו.

הקשיים עימם התמודדתי בעת הכנת הפרויקט

כשהתחלתי לכתוב את הפרויקט לא התמודדתי עם הרבה קשיים. הקשיים הגיעו בהמשך:

ראשית, התעכבתי מאוד ולקח לי הרבה זמן להבין כיצד להגריל מוקשים אקראיים, גם כבר שהבנתי שעושים זאת בעזרת שעון נתקלתי בבעיה נוספת, בין כל הגרלת מוקשת עוברת מעט מאוד אם בכלל הקשות שעון, לכן המספרים האקראיים יצאו זהים. מצאתי כמה דרכים להתמודדות, אחת מהן הייתה השהיית הקוד אך זה יצר בעיה נוספת שללוח לוקח יותר מידי זמן לטעון. לבסוף מצאתי פתרון יעיל אש משתמש בשעון ומבצע קוד קצר נוסף אחרי כדי שיעברו כמה תקתוקי שעון.

קושי נוסף הוא בפיזור המספרים, ניסיתי בהתחלה לעבור ברקורסיה על כל תא שאין בו מוקש, לשאול כמה מוקשים סביבו ולפזר מספרים בהתאם אך הרקורסיה גרמה לבעיות רבות שגזלו לי זמן רב מהפרויקט לכן נאלצתי לוותר עליה. במקום זאת החלטתי לעבור על כל תא **שיש** בו מוקש ולפזר ממנו מספרים בהתאם לתאים הסובבים אותו.

קוד הפרויקט

```
P186
IDEAL
MODEL small
STACK 100h
jumps

;-----

;DEFINES
;-----
ONE_ASCII equ 31h
MINE equ 2ah
EMPTY_IN_BOARD equ 2dh
EMPTY_IN_REVEAL_BOARD equ 23h
ROW_COL_LEN equ 8
BOARD_LEN equ 64
NUMBER_OF_MINE equ 10
HEADER_LEN equ 54
;-----

DATASEG
;-----

;THE-GAME-BOARDS
;-----
board db 64 dup(2dh)
revealBoard db 64 dup(23h)
;-----

;RINT-PICTURE-DATA
;-----
fileLose db 'LOSE.bmp',0
fileWin db 'WIN.bmp',0
filehandle dw 0
Header db 54 dup (0)
Palette db 256*4 dup (0)
ScrLine db 320 dup (0)
ErrorMsg db 'ERROR', 10, 13
isOpenFileLose db 0
;-----

;GAME-MESSAGES
;-----
```


[illegible]

```

inputRow db 0
inputCol db 0
indexAfterCon dw 0
validInput db 0
cnt db 0
;-----

;SOUND-DATA
;-----
frequency dw 0a98h
;-----

CODESEG
;-----

proc gameFrame
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; set the resolution to (640 x 480) with 256 colors
    mov ax, 4F02h
    mov bx, 101h
    int 10h

    ; print game frame screen
    mov dx, offset gameframeMSG
    mov ah, 9h
    int 21h

    ; wait for key press
    mov ah, 0h
    int 16h

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; print opening screen
    mov dx, offset openScreenMSG
    mov ah, 9h
    int 21h

    ; wait for key press

```

```

    mov ah, 0h
    int 16h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    popa
    ret
endp gameFrame

proc printBoard

    push bp
    mov bp, sp
    pusha

    call printnumbersuptheboard

    ;Inserts the values from the stack into registers
    mov bx, [bp + 4] ;board
    mov cx, [bp + 6] ;board length
    printArrayLoop:

    mov al, [byte ptr bx]
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    ;-----
    mov ax, cx
    dec ax
    mov dl, ROW_COL_LEN
    div dl
    cmp ah, 0
    je printLine
    jmp notPrintLine
    ;-----

    printLine:
    call printLineInBoard

```

```

notPrintLine:
inc bx
loop printArrayLoop

mov dx, offset newlineMSG
mov ah, 9h
int 21h

popa
pop bp
ret 4

endp printBoard

proc printNumbersUpTheBoard
pusha

mov [cnt], 31h
xor cx, cx
mov cl, ROW_COL_LEN

mov dx, offset spaceMSG
mov ah, 9h
int 21h

mov dx, offset spaceMSG
mov ah, 9h
int 21h
numLoop:

mov al, [cnt]
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset spaceMSG
mov ah, 9h
int 21h

inc [cnt]
loop numloop

mov dx, offset newlineMSG
mov ah, 9h
int 21h

```

```

mov dx, offset spaceMSG
mov ah, 9h
int 21h

mov al, '_'
mov dx, ax
mov ah, 2h
int 21h

mov al, '_'
mov dx, ax
mov ah, 2h
int 21h

xor cx, cx
mov cl, 22
numLoop2:

mov al, '_'
mov dx, ax
mov ah, 2h
int 21h

loop numloop2

mov dx, offset newlineMSG
mov ah, 9h
int 21h

mov al, ' '
mov dx, ax
mov ah, 2h
int 21h

mov al, '|'
mov dx, ax
mov ah, 2h
int 21h

mov dx, offset newlineMSG
mov ah, 9h
int 21h

mov [cnt], 31h
mov al, [cnt]
mov dx, ax

```

```

    mov ah, 2h
    int 21h

    inc [cnt]

    mov al, '|'
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

    popa
    ret
endp printnumbersuptheboard

proc printLineInBoard
    pusha

    mov dx, offset newlineMSG
    mov ah, 9h
    int 21h

    cmp [cnt], 39h
    je enddddd

    mov al, ' '
    mov dx, ax
    mov ah, 2h
    int 21h

    mov al, '|'
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset newlineMSG
    mov ah, 9h
    int 21h

    mov al, [cnt]
    mov dx, ax
    mov ah, 2h
    int 21h

```

```

    inc [cnt]

    mov al, '|'
    mov dx, ax
    mov ah, 2h
    int 21h

    mov dx, offset spaceMSG
    mov ah, 9h
    int 21h

enddddd:
    popa
    ret
endp printLineInBoard

proc setMines
    pusha

    mov cl, NUMBER_OF_MINE
setMinesRndloop:
    mov ah, 2Ch
    push cx ;preserve value of cx
    int 21h ;dl stores milliseconds
    pop cx
    xor ax, ax
    mov al, dl
    mov bl, BOARD_LEN
    div bl ;ah stores the remainder
    mov bx, offset board
    add bl, ah
    cmp [byte ptr bx], MINE ;is there a bomb already?
    je setMinesRndloop
    ; avoid the 4 center cells
    cmp bl, 27
    je setMinesRndloop
    cmp bl, 28
    je setMinesRndloop
    cmp bl, 35
    je setMinesRndloop
    cmp bl, 36
    je setMinesRndloop
    mov [byte ptr bx], MINE

    loop setMinesRndloop
    popa

```

```

    ret
endp setMines

proc addNumber
    add bx, offset board
    add bx, cx
    cmp [byte ptr bx], MINE ;is there a mine?
    je ADDNUM_continue

    cmp [byte ptr bx], EMPTY_IN_BOARD
    je isEmpty
    add [byte ptr bx], 1
    jmp ADDNUM_continue

isEmpty:
    mov [byte ptr bx], ONE_ASCII

    ADDNUM_continue:
        ret
endp addNumber

proc checkLeft
    mov ax, cx
    mov bl, 8
    div bl
    cmp ah, 0
    ret
endp checkLeft

proc checkRight
    mov ax, cx
    mov bl, 8
    div bl
    cmp ah, 7
    ret
endp checkRight

proc initNumbers
    pusha

    mov cx, 0
MAP_loop:
    mov bx, offset board
    add bx, cx
    cmp [byte ptr bx], MINE ;is there a mine?

```



```

    jne MAP_continue
    ; check for free space above
    cmp cl, 8
    jl MAP_left
    ; check for free space to the left
    call checkLeft
    je MAP_top
    ; add number
    mov bx, -9
    call addNumber
MAP_top:
    ; add number
    mov bx, -8
    call addNumber
MAP_topright:
    ; check for free space to the right
    call checkRight
    je MAP_left
    ; add number
    mov bx, -7
    call addNumber
MAP_left:
    ; check for free space to the left
    call checkLeft
    je MAP_right
    ; add number
    mov bx, -1
    call addNumber
MAP_right:
    ; check for free space to the right
    call checkRight
    je MAP_bottomleft
    ; add number
    mov bx, 1
    call addNumber
MAP_bottomleft:
    ; check for free space below
    cmp cl, 55
    jg MAP_continue
    ; check for free space to the left
    call checkLeft
    je MAP_bottom
    ; add number
    mov bx, 7
    call addNumber
MAP_bottom:
    ; add number
    mov bx, 8

```

```

        call addNumber
MAP_bottomright:
        ; check for free space to the right
        call checkRight
        je MAP_continue
        ; add number
        mov bx, 9
        call addNumber
MAP_continue:
        inc cx
        cmp cx, BOARD_LEN
        jne MAP_loop

        popa
        ret
endp initNumbers

```

```

proc inputToIndex
    pusha

    mov dx, offset newLineMSG
    mov ah, 9
    int 21h

    mov dx, offset inputRowMSG
    mov ah, 9
    int 21h

    call input
    mov al, [validInput]
    mov [inputRow], al
    dec [inputRow]

    mov dx, offset newLineMSG
    mov ah, 9
    int 21h

    mov dx, offset inputColMSG
    mov ah, 9
    int 21h

    call input
    mov al, [validInput]
    mov [inputCol], al
    dec [inputCol]
    call convertRowCol

    popa

```

```

    ret

endp inputToIndex

proc input
    pusha

    checkInputt:
    mov ah, 08h
    int 21h
    sub al, '0'

    cmp al, 0
    je checkInputt

    cmp al, 8
    ja checkInputt
    jmp successCheck

    successCheck:

    mov [validInput], al

    add [validinput], '0'

    mov dl,[validinput]
    mov ah,2
    int 21h

    sub [validinput], '0'

    popa
    ret
endp input

proc revealCell
    pusha

    mov si, offset revealBoard
    mov di, offset board

    add si, [indexAfterCon]
    add di, [indexAfterCon]

    cmp [byte ptr di], MINE
    je mineCell

```

```

notMineCell:
mov al, [byte ptr di]
mov [byte ptr si], al
jmp endd

mineCell:
mov [isGameOver], 1
jmp endd

endd:
popa
ret
endp revealCell

proc convertRowCol
pusha

xor ax, ax
mov al, ROW_COL_LEN
mul [inputRow]
add al, [inputCol]
mov [indexAfterCon], ax

popa
ret
endp convertRowCol

proc checkWin
push bp
mov bp, sp
pusha

;Inserts the values from the stack into registers
mov si, [bp + 4] ;board
mov di, [bp + 6] ;reavel board
mov cx, [bp + 8]

xor bx, bx
checkWinLoop:
mov al, [di + bx]
cmp al, 23h ;Check if is a cell that not opened
je checkMine
jmp continueLoop

checkMine:

```

```

    mov al, [si + bx]
    cmp al, 2ah
    jne notWin

    continueLoop:
    inc bx
    loop checkWinLoop
    jmp win

win:
    mov [iswin], 1
    jmp enddd

notWin:
    mov [iswin], 0

enddd:
    popa
    pop bp
    ret 6
endp checkwin

proc gameOverP
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov [isOpenFileLose], 1
    call OpenFile
    call ReadHeader
    call ReadPalette
    call CopyPal
    call CopyBitmap

    ; wait for key press
    mov ah, 0h
    int 16h

    ; returns to text mode
    mov ah, 0h
    mov al, 03h

```

```

    int 10h

    popa
    ret
endp gameOverP

proc winGameP
    pusha

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov [isOpenFileLose], 0
    call openFile
    call readHeader
    call readPalette
    call copyPal
    call copyBitmap

    ; wait for key press
    mov ah, 0h
    int 16h

    ; returns to text mode
    mov ah, 0h
    mov al, 03h
    int 10h

    popa
    ret
endp winGameP

proc openFile
    mov ah, 3Dh
    xor al, al

    cmp [isOpenFileLose], 1
    je openLose
    jne openWin

openLose:

```

```

    mov dx, offset fileLose
    jmp continueOpen

openWin:
    mov dx, offset fileWin
    jmp continueOpen

continueOpen:
    int 21h
    jc openererror
    mov [filehandle], ax

    ret
openererror:
    mov dx, offset ErrorMsg
    mov ah, 9h
    int 21h
    ret
endp openFile

proc readHeader
    ; Read BMP file header, 54 bytes
    mov ah, 3fh
    mov bx, [filehandle]
    mov cx, HEADER_LEN
    mov dx, offset Header
    int 21h
    ret
endp readHeader

proc readPalette
    ; Read BMP file color palette, 256 colors * 4 bytes (400h)
    mov ah, 3fh
    mov cx, 400h
    mov dx, offset Palette
    int 21h
    ret
endp readPalette

proc copyPal
    ; Copy the colors palette to the video memory
    ; The number of the first color should be sent to port 3C8h
    ; The palette is sent to port 3C9h
    mov si, offset Palette
    mov cx, 256
    mov dx, 3C8h

```

```

mov al,0
; Copy starting color to port 3C8h
out dx,al
; Copy palette itself to port 3C9h
inc dx
PalLoop:
; Note: Colors in a BMP file are saved as BGR values rather than RGB.
mov al,[si + 2] ; Get red value.
shr al,2 ; Max. is 255, but video palette maximal
; value is 63. Therefore dividing by 4.
out dx,al ; Send it.
mov al,[si + 1] ; Get green value.
shr al,2
out dx,al ; Send it.
mov al,[si] ; Get blue value.
shr al,2
out dx,al ; Send it.
add si,4 ; Point to next color.
; (There is a null chr. after every color.)

loop PalLoop
ret
endp copyPal

proc copyBitmap
; BMP graphics are saved upside-down.
; Read the graphic line by line (200 lines in VGA format),
; displaying the lines from bottom to top.
mov ax, 0A000h
mov es, ax
mov cx,200
PrintBMPLoop:
push cx
; di = cx*320, point to the correct screen line
mov di,cx
shl cx,6
shl di,8
add di,cx
; Read one line
mov ah,3fh
mov cx,320
mov dx, offset ScrLine
int 21h
; Copy one line into video memory
cld ; Clear direction flag, for movsb
mov cx,320
mov si, offset ScrLine

```



```

    rep movsb ; Copy line to the screen

    pop cx
    loop PrintBMPLoop
    ret
endp copyBitmap

proc playCellSound
    pusha

    ; open speaker
    in al, 61h
    or al, 00000011b
    out 61h, al
    ; send control word to change frequency
    mov al, 0B6h
    out 43h, al
    ; play frequency
    mov ax, [frequency]
    out 42h, al ; Sending lower byte
    mov al, ah
    out 42h, al ; Sending upper byte

    ;delay the sound
    mov cx, 02H
    mov dx, 4240H
    mov ah, 86h
    int 15h

    ; close the speaker
    in al, 61h
    and al, 11111100b
    out 61h, al

    popa
    ret
endp playcellsound

proc askMine
    pusha

    mov dx, offset easierlevelMSG
    mov ah, 9h
    int 21h

    mov ah, 08h
    int 21h

```

```

    sub al, '0'

    cmp al, 1
    jne notAskMine

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    mov dx, offset askMineMSG
    mov ah, 9h
    int 21h

    call inputToIndex
    mov bx, offset board
    add bx, [indexAfterCon]
    mov [byte ptr bx], MINE

    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

notAskMine:
    ; clear DOS-Box window
    mov ax, 3h
    int 10h

    ; switch to graphic mode (40 x 25)
    mov ax, 13h
    int 10h

    popa
    ret
endp askMine

proc game
    pusha

    mov dl, 0 ; Column
    mov dh, 0 ; Row

```

```

mov bx, 0 ; Page number, 0 for graphics modes
mov ah, 2h
int 10h

push BOARD_LEN
push offset revealBoard
call printBoard

gameLoop:

push BOARD_LEN
push offset revealBoard
push offset board
call checkWin
cmp [iswin], 1
je winGame

call inputToIndex
call revealcell

cmp [isGameOver], 1
je gameOverr

mov dl, 0 ; Column
mov dh, 0 ; Row
mov bx, 0 ; Page number, 0 for graphics modes
mov ah, 2h
int 10h

push BOARD_LEN
push offset revealBoard
call printBoard
call playcellsound
jmp gameloop

gameOverr:
call gameOverP
jmp endddd

winGame:
call winGameP
jmp endddd

endddd:
popa
ret

```

```

endp game

proc runGame
    pusha

    call gameFrame
    call askmine
    call setMines
    call initNumbers

    ;For check, you can remove the note and see the answer board
    ;#####
    ;push 64
    ;push offset board
    ;call printBoard
    ;#####

    call game

    popa
    ret
endp runGame

;-----
start:
    mov ax, @data
    mov ds, ax

    call rungame
;-----
-
exit:
    mov ax, 4c00h
    int 21h
END start
;-----

```