ציורי פיקסלים – וגרפיקה **מורה**

בדף עבודה זה נתנסה בעבודה במצב גראפי. (בשונה למצב טקסט)

נצייר ביחד ריבוע, קווים אנכיים וקווים אופקיים.

# מעבר למצב גרפי

המעבד יכול לעבוד בשני מצבים הראשון מצב טקסט (בו עבדנו עד היום) והשני מצב גרפי בו משתמשים בגרפיקה (ציור ושרטוט על המסך) כדי לעבור למוד גראפי נשתמש בפסיקת ה bios 10h

mov ax, 13h

int 10h

צרו תכנית העוברת למוד גראפי. בדקו בעזרת ה turbo debugger (Alt -F5) מה קורה למסך ה – DosBox במוד זה?

**חזרה ממצב גרפי**:

mov ax, 2

int 10h

הוסיפו בסוף התוכנית שלכם את ה interrupt החוזר למצב text.

הריצו את התכנית ב – Turbo Debugger, כדי לצפות במסך המשתמש הקישו **Alt + F5**.

חזרה למסך ה – TD הקישו **Esc**.

# ציור במצב גרפי – ציור פיקסל (נקודה על המסך)

המסך מורכב מנקודות. כל נקודה נקראת פיקסל. במצב גרפי המסך מורכב מ 200 שורות ו 320 עמודות של פיקסלים. לכל פיקסל מיקום וצבע.

המיקום מורכב משורה ועמודה ⇦ row, column

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 319,0 |  |  |  |  |  |  | 0, 0 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 319, 199 |  |  |  |  |  |  | 0,199 |

**הצבע:**

בזיכרון המחשב יש טבלה בגודל 256 תאים ובכל תא שלה מצוי מספר המייצג צבע.

כדי לצייר פיקסל למסך, צריך לציין שלושה נתונים: מיקום על ציר x, מיקום על ציר y וצבע.

לדוגמה: שורה 0

עמודה 0

צבע 4

יצבע בצבע אדום את הפיקסל בפינה השמאלית העליונה

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 319,0 |  |  |  |  |  |  | 0, 0 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 319, 199 |  |  |  |  |  |  | 0,199 |

האינטרפט (פסיקה) המצייר פיקסל למסך, הוא פסיקת bois - **int 10h**

**הפרמטרים עבור אינטרפט זה:**

DATASEG

; --------------------------

; Your variables here

x\_coordinate dw 50 ; in column

y\_coordinate dw 20 ; in line

color dw 4

CODESEG

start:

mov ax, @data

mov ds, ax

; --------------------------

; Enter Graphic mod

mov ax, 13h

int 10h

; draw pixel

xor bh, bh

mov cx, [x\_coordinate] ;in column

mov dx, [y\_coordinate] ; in line

mov ax, [color]

mov ah, 0ch

int 10h

; Enter text mod

mov ax, 2

int 10h

; --------------------------

ah = 0ch – קוד הפסיקה

bh = 0 – איפוס רגיסטר bh

cx – עמודה

dx – שורה

ax - צבע

כתבו תכנית המציירת פיקסל אדום )תא 4 בפלטת הצבעים) בשורה 20 עמודה 50.

xor bh, bh

mov cx, [x\_coordinate]

mov dx, [y\_coordinate]

mov ax, [color]

mov ah, 0ch

int 10h

בהרצת התכנית ב – TD ניתן לראות את הציור על מסך (לאחר הפעלת הפסיקה) ע"י הקשה על המקשים **Alt+F5**

או בחירה בתפריט Window

ו - User screen.

# פרוצדורה

שנו את קטע הקוד לפרוצדורה העושה שימוש במשתנים לקואורדינטות x ו y של פיקסל ואת צבעו ומציירת אותו על המסך.

proc drawPixel

; draw pixel

xor bh, bh

mov cx, [x\_coordinate] ; place in column

mov dx, [y\_coordinate] ; place in line

mov ax, [color]

mov ah, 0ch

int 10h

ret

endp drawPixel

כדי לשמר את מצב הרגיסטרים בכל פעולה של פרוצדורה נבצע פעולות push ו – pop לכל הרגיסטרים.

ניתן לבצע push וpop לכל הרגיסטרים יחד.

נוסיף בתחילת התכנית את הפקודה p186 (לפני ה – DATASEGMENT).

# הוספת מקש

נוסיף לתכנית פסיקה אשר בהקשה על מקש היא יוצאת מהמצב הגרפי וחוזרת למצב טקסט.

הפסיקה הבודקת אם הוקש מקש במקלדת היא:

; Wait for key press

mov ah,00h

int 16h

start:

mov ax, @data

mov ds, ax

; --------------------------

; Enter Graphic mod

mov ax, 13h

int 10h

call drawPixel

**; Wait for key press**

**mov ah,00h**

**int 16h**

;Retern text mod

mov ax, 2

int 10h

; --------------------------

exit:

mov ax, 4c00h

int 21h

END start

הוסיפו פסיקה זו לאחר הקריאה של הפרוצדורה לציור פיקסל. ולפני הפקודה של חזרה למצב של טקסט.

במצב זה ניתן להריץ את התכנית ללא ה – Turbo Debugger.

הריצו את התכנית שוב ישירות

הריצו את הקובץ ב – tasm.

הריצו את הקובץ ב – tlink.

הריצו את הקובץ - רשמו את שמו בלבד ו – [Enter]

ליציאה מהתכנית הקישו על מקש (כל מקש).

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

# ציור קו אופקי

קו הוא אוסף של נקודות ............... על ציר ה – x

כתבו פרוצדורה המציירת קו על המסך (לרוחב). הפרוצדורה עושה שימוש בקואורדינטת תחילת

השורה אורך הקו, ובצבע הפיקסל ומציירת את השורה.

* לצורך ביצוע השרטוט הקו עלינו ליצור משתנה חדש שבו נשמור את ארוך הקו, [len]

**ניצור פרוצדורה חדשה המציירת קו למסך - drawHorizontalLine**

* בתחילת הפרוצדורה נבצע פקודת push לכל הרגיסטרים כדי לשמור את ערכם למחסנית.
* בתחילת התכנית (לפני ה – DATASEGMENT) נוסיף את ההוראה p186  
  בפרוצדורה זו יש שימוש בלולאה ולכן חשוב מאוד לשמר את הרגיסטרים בכלל ואת רגיסטר cx בפרט.
* בתחילת הפרוצדורה נכתב את הפקודה **pusha** - פקודה העושה push לכל הרגיסטרים.
* ניצור לולאה שערכה הוא כאורך הקו בפיקסלים (נעביר לרגיסטר cx את הערך של משתנה len).
* נזמן את הפעולה של ציור הפיקסל מתוך פרוצדורה זו.
* נגדיל את ערכו של [x\_coordinate]
* בסיום הפרוצדורה נבצע פעולת **popa** לערכים של כל הרגיסטרים.

proc drawHorizontalLine

; draw line of pixels

**pusha**

mov cx, [len]

drawLine:

call drawPixel

inc [x\_ coordinate]

loop drawLine

**popa**

ret

endp drawHorizontalLine

IDEAL

MODEL small

STACK 100h

**p186**

DATASEG

x\_coordinate dw 50 ; place in line

y\_coordinate dw 20 ; place in column

len dw 20

color dw 5

CODESEG

start:

mov ax, @data

mov ds, ax

; Enter Graphic mod

mov ax, 13h

int 10h

call drawHorizontalLine

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

# ציור קו אנכי

proc drawVerticalLine

; draw column of pixels

pusha

mov cx, [len]

drawColum:

call drawPixel

inc [y\_ coordinate]

loop drawColum

popa

ret

endp drawVerticalLine

כתבו פעולה המציירת קו אנכי.

קו אנכי הוא אוסף נקודות על ציר y

כתבו פרוצדורה חדשה drawVerticalLine

אל תשכחו לפני הקריאה לפרוצדורה להשים ערכים למשתנים של תחילת ציור הקו על

ציר ה – x ועל ציר ה – y.

mov [x\_coordinate], 30

mov [y\_coordinate], 30

call drawVerticalLine

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

# ציור מסגרת למסך

mov [x\_coordinate], 6

mov [y\_coordinate], 6

mov [len], 308

call drawHorizontalLine

mov [x\_coordinate], 6

mov [y\_coordinate], 6

mov [len], 188

call drawVerticalLine

mov [x\_coordinate], 6

mov [y\_coordinate], 194

mov [len], 308

call drawHorizontalLine

mov [x\_coordinate], 314

mov [y\_coordinate], 6

mov [len], 189

call drawVerticalLine

גודל המסך הוא רוחב 320 פיקסלים וגובה 200 פיקסלים.

ציירו מסגרת למסך כל צבע שתבחרו.

השאירו שוליים צרים מכל צד של מסגרת ה – DoxBox.

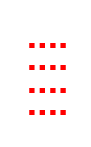
חשבו כך שגודל המסגרת יהי זוגי ויתחיל בנקודה זוגית בנקודות ציר

ה – x וציר ה – y.

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

# ציור ריבוע

ריבוע הוא אוסף של קווים 

בציור הריבוע נשתמש בטכניקה שונה. זה יהיה השחקן שלנו שיזוז בתוך המסגרת. לכן חשוב לנו לשמור את נקודת ה – x וה – y של מיקום תחילת ציור הריבוע.

תזוזה של הריבוע על פני המסך היא ציור ריבוע שוחר על הריבוע הקיים – מחיקת הריבוא וצייר ריבוע חדש בנקודה אחרת על המסך.

ניצור פרוצדורה חדשה לציור הריבוע. לצורך ציור הריבוע נשתמש בשתי לולאות מותנות.

נקבע 4 משתנים חשים:

מיקום תחילת הריבוע על ציר x,

מיקום תחילת הריבוע על ציר y,

משתנה שיהיה מונה הלולאה של ציור הקו האופקי

משתנה שיהיה מונה הלולאה על הציר האנכי (ציור הקווים)

x\_begin dw 100 ; Starting point on line

y\_begin dw 30 ; Starting point on column

x\_count db 10 ; loop count line draw

y\_count db 10 ; loop count column draw

פרוצדורה לציור ריבוע בשתי לולאות מותנות:

Proc DrawRectangle

; Draw square at x\_begin, y\_begin position, size 10\*10

mov [x\_count], 10

mov [y\_count], 10

קביעת גודל הריבוע והעברת הערכים של מיקם תחילת הריבוע ל משתנים x\_ coordinate ו – y\_ coordinate.

mov ax, [x\_begin]

mov [x\_coordinate], ax

mov ax,[y\_begin]

mov [y\_coordinate], ax

line\_loop:

; Print one dot on screen

call drawPixel

לאחר ציור נקודה אחת על המסך, מגדילים את הערך של מיקום הנקודה על ציר ה – x (מיקום על השורה),

ומקטינים את מונה הלולאה של ציור הקו האופקי ב – 1. כאשר מונה הלולאה מגיע ל – 0 יוצאים ממנה.

inc [x\_coordinate]

dec [x\_count]

cmp [x\_count], 0

jne line\_loop

mov ax, [x\_begin] ;reset line counters

mov [x\_coordinate], ax

mov [x\_count],10 ;reset column counters

inc [y\_coordinate]

לאחר ציור קו אחד למסך, מאפסים את תחילת ציור הקו על ציר x.

מאפסים את מונה הלולאה המותנית של ציור קו אופקי.

מגדלים את מיקום ציור הנקודה על ציר ה – y (ציר אופקי, עמודה)

מקטינים את מונה הלולאה של ציור הקווים .(ציר אופקי).

כאשר מונה הלולאה של ציור הקווים מגיע לאפס, יוצאים משתי הלולאות.

dec [y\_count]

cmp [y\_count], 0

jne line\_loop

ret

endp DrawRectangle

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

# הזזת השחקן על פני המסך בעזרת מקשים

לצורך הזזת הריבוע על המסך עלינו תחילה לצייר ריבוע שחור על הריבוע הצבעוני ולאחר מכן לצייר את הריבוע החדש במיקום הבא.

Scan codes

כל לחיצה או שחרור של מקש במקלדת מייצרים קוד מיוחד – **scan code**.

המספר שנוצר נשמר בזיכרון מיוחד של המעבד שנקרא זיכרון **I/O** (Input/Output)

תהליך ההקלדה

* המשתמש לוחץ או משחרר מקש
* **בזיכרון ה I/O נכתב ה scan code המתאים**
* ה־PIC מקבל אינטרפט מהמקלדת.
* ה־ PIC שולח למעבד אינטרפט (int 9), שאומר למעבד שיש מידע בזיכרון ה I/O של המקלדת
* כתגובה לפסיקה, המעבד מריץ ISR שמטפל בפסיקה 9. ה־ISR "מעתיק" ה־code scan אל מקום מוגדר בזיכרון שנקא Type Ahead Buffer

Type Ahead Buffer

מקום בזיכרון בו המעבד שומר את המידע מהמקלדת.

עבור כל תו המעבד שומר את ה Scan Code ואת התרגום שלו ל ASCII

כלומר לכל תו נשמרים 2 בתים

גודל הבאפר הוא 32 בתים. **כלומר יש מקום ל 16 תווים.**

לכן עלינו לדאוג לנקות את הבאפר

יש לנו בתכנית כבר פסיקה הממתינה למקש, וכאשר מקישים על מקש (כל מקש) אנחנו יוצאים מתצוגה גרפית וחוזרים לתצוגת טקסט רגילה.

נוסף לפסיקה הוראה שתבדוק האם הקישו על מקש Esc אם הקישו על מקש Esc נצא למצב טקסט.

נוסיף לפני הפסיקה של ההמתנה למקש תווית:

**WaitForKeyPress:**

אם המשתמש הקיש על מקש הבאפר (זיכרון) המקלדת מיכל את הערך של המקש.

הפסיקה:

mov ah, 1h

int 16h

מדליקה את ה zero flag– אם יש תו מוכן לקריאה(zf = 1) , ומכבה אם אין תו מוכן (zf = 0).

אם יש תו מוכן, al ו-ah יקבלו את ערכי ה־ASCII וה־code scan של התו.

(הפסיקה אינה "מנקה" את התו מהבאפר של המקלדת).

## האם מקש Esc נלחץ?

ניצור רצף פקודות שאם הקישו על מקש Esc נצא מהתכנית.

כדי לבדוק האם הקישו תו במקלדת נבדקו את דגל האפס ⇦ zf = 1.

אם דגל האפס = 0 לא הוקש מקש במקלדת, נמשיך למתקין למקש.

אחרת:

נעביר לרגיסטר ah את ערך המקש.

**נבדק אם הערך של ah = 1h (נלחץ Esc) ⇦ נצא מהלולאה.**

; Wait for key press

WaitForKeyPress:

mov ah,1 ;Check Type Ahead Buffer 1 = no character

int 16h

jz WaitForKeyPress

mov ah,0 ; Read key pressed

int 16h

cmp ah,1h ; if Esc

je TheEnd

jmp WaitForKeyPress

TheEnd:

;Retern to text mod

**הריצו ובדקו.**

**שמרו את הקובץ, בשם חדש והמשיכו בקובץ החדש.**

mov ax, 2

int 10h

## הזזת השחקן למעלה

ניצור פעולה של הזזת הריבוע למעלה: UpKeyPress

תחילה נמחק את הריבוע הקיים.

נצייר את הריבוע בצבע שחור.

נזיז את תחיל ציור הריבוע במינו שני פיקסלים על ציר ה – y [y\_begin].

proc UpKeyPress

mov [color], 0

call DrawRectangle

sub [y\_begin], 2

mov [color], 15

call DrawRectangle

ret

endp UpKeyPress

נחזור לתכנית הראשית ונוסיף בדיקה האם הקישו על מקש חץ למעלה ⇧ - **048h**

; Wait for key press

WaitForKeyPress:

לב התכנית:

ממתינים להקשת תו

אם הוקש Esc

יציאה מהתכנית

אם הוקש - חץ מעלה

קפיצה לתווית חץ מעלה

חזרה להמתנה למקש.

mov ah,1

int 16h

jz WaitForKeyPress

mov ah,0

int 16h

cmp ah,1h

je TheEnd

cmp ah, 048h

je up

jmp WaitForKeyPress

up:

call UpKeyPress

jmp WaitForKeyPress

TheEnd:

;Retern to text mod

mov ax, 2

int 10h

הרחיבו את התכנית, היספו תנאים ופרוצדורות למקשים של שאר החיצים.

050h – חץ מטה ⇩

**בדקו אחר כל הוספת תנאי ופרוצדורה והריצו את התכנית.**

**שמרו את התכנית בשם חדש והמשיכו לשלב הבא.**

04bh – חץ שמאלה ⇦

04dh - חץ ימינה ⇨

# האם נוגע בצבע

במשק שיצרנו עם השחקן עולה על המסגרת הוא צבוע אותה בשחור ויכול לצאת ממסגרת המסך ולחזור בצדו השני. אנחנו רוצים למנוע מצב זה.

לשם כך לפני תזוזת השחקן נבדוק מהו הצבע של הפיקסלים במיקום הבא שאליו השקן צריך להגיע אליו. אם הצבע כמו צבע הרקע השחקן ינוע למיקום הבא אם הצבע הוא צבע כצבע המסגרת לא נאפשר את תנועת השחקן.

**פסיקת bios ah = 0dh** מאפשרת לנו לקרא ערך צבע של פיקסל מהמסך. הפסיקה מחזירה את ערך הצבע בנקודה שמבצעים עליה לרגיסטר al.

הפסיקה:

mov bh, 0h

mov cx, [x]

mov dx, [y]

mov ah, 0dh

int 10h

ערך הצבע נשמר לרגיסטר al.

כתבו פעולה הבודקת את ערך הצבע בנקודה מסוימת על פני המסך ושומרת את הצבע למשתנה [check].

נוסיף משתנים [x] ו – [y] שיצביעו על מיקום הבדיקה של הצבע.

proc CheckColor

; get color of a single pixel --> return it to al

mov bh, 0h

mov cx, [x]

mov dx, [y]

mov ah, 0dh

int 10h

mov [check], al ; save color of pixel

ret

endp checkColor

לפני כל תזוזה של השחקן (הריבוע)

עלינו להעביר ל - [x] ול -[y] את המיקום הפיקסל שנרצה לבדוק את צבעו.

מיקום הפיקסל שנרצה לבדוק הוא המיקום הבא שאליו יזוז השחקן.

נוסיף לכל פרוצדורה את הפקודות שיעזרו לנו לבדוק את הצבע בצעד הבא של השחקן.

proc UpKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

נעביר את הערך של תחילת ציור הריבוע למשתנים החדשים [x] [y].

אנו רוצים להזיז את השחקן מעלה על כן נפחית 2 פיקסלים מהמיקום בעמודה (משתנה y).

נזמן את הפרוצדורה של בקידת הצבע

נשווה את הצבע שהתקבל לצבע המסגרת. אם אותו צבע נצא מהפרוצדורה.

אחרת נתקדם.

mov [x], ax

sub [y], 2 ; sub 2 for 2 pixel move

call CheckColor

cmp [check], 5

je endUp

mov [color], 0

call DrawRectangle

sub [y\_begin], 2

mov [color], 15

call DrawRectangle

endUp:

ret

endp UpKeyPress

תקנו את הפרוצדורות של שאר הכיוונים. שימו לב באיזה מיקום לבדוק את הפיקסל.

proc DownKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

mov [x], ax

**add [y], 12 ; add 2 for 2 pixel move & 10 for rec size**

call CheckColor

cmp [check], 5

je endDown

mov [color], 0

call DrawRectangle

add [y\_begin], 2

mov [color], 15

call DrawRectangle

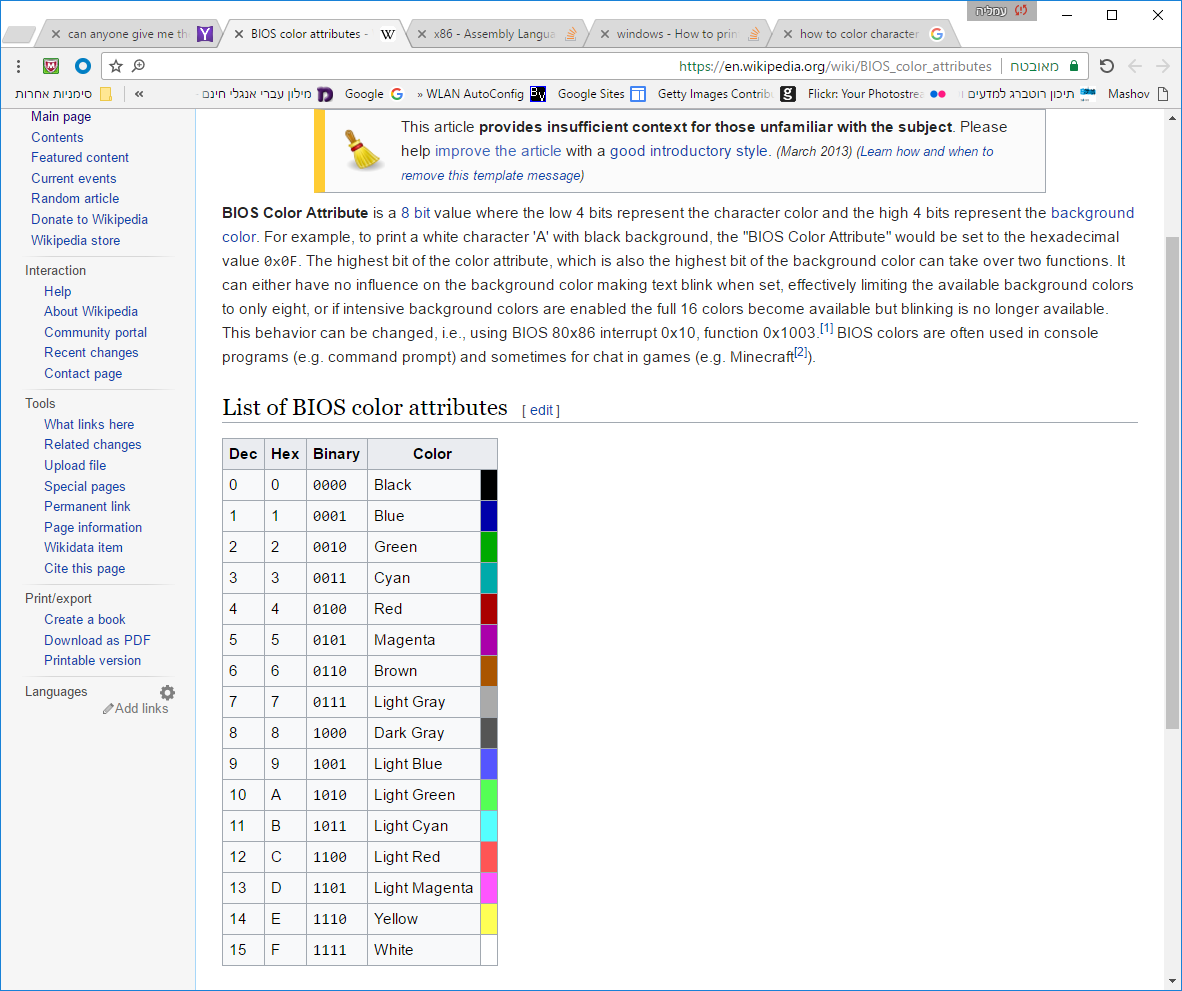
endDown:

ret

endp DownKeyPress

|  |  |
| --- | --- |
| proc LeftKeyPress  mov ax, [y\_begin]  mov [y], ax  mov ax, [x\_begin]  mov [x], ax  sub [x], 2 ; sub 2 for 2 pixel move  call CheckColor  cmp [check], 5  je endLeft  mov [color], 0  call DrawRectangle  sub [x\_begin], 2  mov [color], 15  call DrawRectangle  endLeft:  ret  endp LeftKeyPress | proc RightKeyPress  mov ax, [y\_begin]  mov [y], ax  mov ax, [x\_begin]  mov [x], ax  add [x], 12 ; add 2 for 2 pixel move & 10 for rec size  call CheckColor  cmp [check], 5  je endRight  mov [color], 0  call DrawRectangle  add [x\_begin], 2  mov [color], 15  call DrawRectangle  endRight:  ret  endp RightKeyPress |

# הרחבות

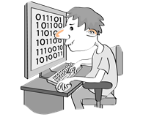
שפרו את התכנית.

הוסיפו קווים לציור מבוך, או "הפרעות" אחרות לציור מסלול מכשולים.

אפשר לשנות את השחקן (ציור דמות)

אפשר להוסיף ניקד.

ועד כיד הדמיון הטובה עליכם.



# קודה התכנית:

פתיח + משתנים

IDEAL

MODEL small

STACK 100h

**p186**

; תרגיל מודרך ציור מבוך

; ציור מסגרת למסך, ציור שחקן - ריבוע

; הזזת הריבוע

; נוגע בצבע

DATASEG

; --------------------------

; Your variables here

x\_coordinate dw ? ; place in line

y\_coordinate dw ? ; place in column

len dw ?

color dw 5

x\_begin dw 100 ; Starting point on line

y\_begin dw 30 ; Starting point on column

x\_count db 10 ; loop count line draw

y\_count db 10 ;loop count column draw

x dw ? ; x coordinate for check if tach color on next step

y dw ? ; y coordinate for check if tach color on next step

check db ? ; save cheac coolor, color

פעולות

CODESEG

proc drawPixel

pusha

xor bh, bh

mov cx, [x\_coordinate] ; place in line

mov dx, [y\_coordinate] ; place in column

mov ax, [color]

mov ah, 0ch

int 10h

popa

ret

endp drawPixel

proc drawHorizontalLine

; draw line of pixels

pusha

mov cx, [len]

drawLine:

call drawPixel

inc [x\_coordinate]

loop drawLine

popa

ret

endp drawHorizontalLine

Proc DrawRectangle

; Draw square at x\_begin, y\_begin position, size 10\*10

mov [x\_count],10

mov [y\_count],10

mov ax,[x\_begin]

mov [x\_coordinate], ax

mov ax,[y\_begin]

mov [y\_coordinate], ax

line\_loop:

; Print one dot on screen

call drawPixel

inc [x\_coordinate]

dec [x\_count]

cmp [x\_count], 0

jne line\_loop

mov ax,[x\_begin] ;reset line counters

mov [x\_coordinate], ax

mov [x\_count],10 ;reset column counters

inc [y\_coordinate]

dec [y\_count]

cmp [y\_count], 0

jne line\_loop

ret

endp DrawRectangle

proc drawVerticalLine

; draw column of pixels

pusha

mov cx, [len]

drawColum:

call drawPixel

inc [y\_coordinate]

loop drawColum

popa

ret

endp drawVerticalLine

proc UpKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

mov [x], ax

sub [y], 2 ; sub 2 for 2 pixel move

call CheckColor

cmp [check], 5

je endUp

mov [color], 0

call DrawRectangle

sub [y\_begin], 2

mov [color], 15

call DrawRectangle

endUp:

ret

endp UpKeyPress

proc DownKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

mov [x], ax

add [y], 12 ; add 2 for 2 pixel move & 10 for rec size

call CheckColor

cmp [check], 5

je endDown

mov [color], 0

call DrawRectangle

add [y\_begin], 2

mov [color], 15

call DrawRectangle

endDown:

ret

endp DownKeyPress

proc LeftKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

mov [x], ax

sub [x], 2 ; sub 2 for 2 pixel move

call CheckColor

cmp [check], 5

je endLeft

mov [color], 0

call DrawRectangle

sub [x\_begin], 2

mov [color], 15

call DrawRectangle

endLeft:

ret

endp LeftKeyPress

proc RightKeyPress

mov ax, [y\_begin]

mov [y], ax

mov ax, [x\_begin]

mov [x], ax

add [x], 12 ; add 2 for 2 pixel move & 10 for rec size

call CheckColor

cmp [check], 5

je endRight

mov [color], 0

call DrawRectangle

proc CheckColor

; get color of a single pixel --> return it to al

mov bh, 0h

mov cx, [x]

mov dx, [y]

mov ah, 0dh

int 10h

mov [check], al ; save color of pixel

ret

endp checkColor

add [x\_begin], 2

mov [color], 15

call DrawRectangle

endRight:

ret

endp RightKeyPress

תכנית ראשית

start:

mov ax, @data

mov ds, ax

; --------------------------

; Enter Graphic mod 320 \* 200 pixels, 256 colors

mov ax, 13h

int 10h

ציור מסגרת ושחקן

mov [x\_coordinate], 6

mov [y\_coordinate], 6

mov [len], 308

call drawHorizontalLine

mov [x\_coordinate], 6

mov [y\_coordinate], 6

mov [len], 188

call drawVerticalLine

mov [x\_coordinate], 6

mov [y\_coordinate], 194

mov [len], 308

call drawHorizontalLine

mov [x\_coordinate], 314

mov [y\_coordinate], 6

mov [len], 189

call drawVerticalLine

mov [color], 15

call DrawRectangle

לב המשחק, הקשת מקשים

; Wait for key press

WaitForKeyPress:

mov ah,1 ; check for keystroke in the keyboard buffer

int 16h

jz WaitForKeyPress ; if keystroke is not available

mov ah, 0 ; get keystroke from keyboard --> ah = BIOS scan code

int 16h

cmp ah, 1h ; Esc key

je TheEnd

cmp ah, 048h ; up key

je up

cmp ah, 050h ; down key

je down

cmp ah, 04bh ; left key

je left

cmp ah, 04dh ; right key

je right

jmp WaitForKeyPress

up:

call UpKeyPress

jmp WaitForKeyPress

down:

call DownKeyPress

jmp WaitForKeyPress

left:

call LeftKeyPress

jmp WaitForKeyPress

right:

call RightKeyPress

jmp WaitForKeyPress

TheEnd:

;Retern to text mod

mov ax, 2

exit:

mov ax, 4c00h

int 21h

END start

int 10h