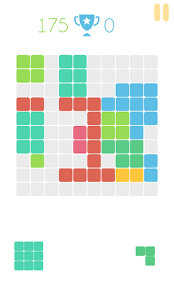


****

**שם הפרויקט:** !1010

**שם המתכנת:** עמית שטיין.

**תעודת זהות:** 207195280

**מורה מלווה:** יוסי זהבי

**בית ספר:** תיכון "הרצוג"

**כיתה:** י'4

**תוכן עניינים:**

שער ........................................................................................1

תוכן עניינים...............................................................................2

מבוא........................................................................................3

נושא העבודה............................................................................4

אופן ההפעלה............................................................................4

מטרת המשחק..........................................................................4

מהלך המשחק .........................................................................5

גרסאות המערכת......................................................................5

פרטים המאפשרים כניסה והפעלה...............................................5

תיעוד והסבר הפתרון.......................................................... 6 - 7

תרשימי זרימה.................................................................10 - 8

רשימת הפעולות..............................................................52 - 11

דוגמאות הרצה.......................................................................53

סיכום אישי............................................................................54

**מבוא:**

שם העבודה שלי הוא "!1010".

שם הקובץ הוא "1010Game".

למשחק יש לי קובץ אחד נלווה המחזיק את הניקוד הגבוה ביותר אי פעם: ""Scor.txt.

סביבת העבודה שלי היא ""Turbo Assembler וכדי ליצור את קובץ ההרצה צריך לרשום:

F:\ PROJECT \tasm /zi Game1010"" – תרגום הקובץ לשפת מכונה על ידי Tasm אשר מקבל קובץ הכתוב בשפת אסמבלי (קובץ asm) ומתרגם אותו לשפת מכונה (קובץ obj)

F: PROJECT \tlink /v Game1010"" – יצירת קובץ ההרצה (קובץ exe - executable) על ידי tlink (Turbo Linker) מקובץ הכתוב בשפת מכונה (obj).

כדי להריץ את הפרוייקט (לאחר הקומפילציה) צריך לרשום: Game1010"" והמשחק ירוץ.

מערכת ההפעלה שלי היא windows 8.

דיבגתי את הקובץ בעזרת התוכנה: TD – ""turbo debugger.

בכדי לדבג את הפרויקט צריך לקמפל אותו לפי ההוראות שלמעלה ובמקום לרשום F:\ PROJECT\ Game1010"", **לרשום: F:\ PROJECT\td Game1010""**.

**סביבת הפיתוח** שלי הייתה Notepad++"".

**סביבת ההרצה** הייתה סימולטור המדמה את המעבד הישן של אינטל: 8086 ששמו "DosBox" גרסא 0.74.

**נושא העבודה:**

נושא העבודה שלי הוא משחק לשחקן יחיד.

המשחק מושקע יחסית מהבחינה הגראפית וקל לשחק בו.

בנוסף המשחק מופעל על ידי העכבר ומאפשר שמירה של התוצאה הגבוהה ביותר גם לאחר היציאה מהמשחק.

**אופן ההפעלה:**

כאשר המשחק נפתח מופיע שם המשחק והוראה למשתמש ללחוץ על העכבר בכל מקום בכדי להמשיך.

לאחר שהמשתמש לחץ לחיצה שמאלית על העכבר יופיע מסך המשחק.

מסך המשחק מכיל מספר אפשרויות ומרכיבים:

אפשרויות (נמצאות בפינה הימנית עליונה) :

1. אפשרות ליציאה בכל רגע נתון מהתוכנית (גם במסך המשחק וגם במסך החוקים) – סימן איקס.
2. אפשרות לקריאת חוקי המשחק בכל רגע נתון - "Rules".
3. אפשרות להתחלת משחק חדש בכל רגע נתון.
4. בנוסף במסך החוקים קיימת האפשרות לחזור חזרה למסך המשחק - "Back" – האפשרות נמצאת מתחת לחוקי המשחק.

שאר המרכיבים:

* לוח משחק - 10\* 10 ריבועים – בצידו השמאלי של המסך.
* הניקוד העכשווי - ""Score – מתחת ל-3 האפשרויות.
* הניקוד הגבוה ביותר איי פעם – "High Score" – מעל הניקוד העכשווי ומתחת ל-3 האפשרויות.
* הצורה הנוכחית – מתחת לניקוד העכשווי.

**מטרת המשחק:**

לנצח את הניקוד הגבוה ביותר . כלומר – להגיע לניקוד גבוה יותר, לשבור שיא – להגיע לניקוד הכי גבוה שאפשר.

**מהלך המשחק:**

בתחילת המשחק כמו בכל תחילת תור, מופיעה צורה רנדומלית המורכבת מקוביות, שאותה צריך להכניס ללוח המשחק באמצעות לחיצה על הלוח.

בכל תור המשתמש יזהה איזו צורה הוא קיבל ולאחר מכן ילחץ על אחת מהתאים שעל הלוח.  
הצורה תצויר על הלוח בהתאם לתא עליו לחץ, כך שאותו התא עליו לחץ המשתמש, ייצג את הקובייה השמאלית עליונה של הצורה שקיבל.

בכל פעם שהמשתמש הכניס צורה ללוח המשחק, הצורה תצויר על הלוח והוא יקבל ניקוד בהתאם למספר הקובית המרכיבות את הצורה.

במידה ויש טור או שורה מלאים על לוח המשחק, הם יהבהבו וימחקו ויתווסף בונוס לניקוד בהתאם למספר הטורים והשורות שנמחקו. (יותר טורים ושורות = יותר ניקוד).

במידה ואין מקום על לוח המשחק לצורה הנוכחית, זהו הפסד.

**גרסאות המערכת:**

בגרסה זו עשיתי שהצורה תוכנס ללוח בלחיצה על הלוח.

בגרסאות הבאות ארצה שהצורה תיגרר על ידי העכבר על הלוח וכך המשתמש לא יצטרך לחשוב על הנקודה השמאלית עליונה של הצורה, אותה מסמל העכבר.

בנוסף, בגרסאות הבאות אוסיף צורות כך שיהיה קשה יותר שלא להיפסל.

**פרטים המאפשרים כניסה והפעלה:**

אין.

צריך להיכנס לפי הוראות ההפעלה שבמבוא.

המשחק הינו קל ונגיש מאוד. הערה אחת חשובה היא שהעכבר מייצג את הקובייה השמאלית והעליונה ביותר של הצורה שהוגרלה וכך גם הצורה תוכנס לטבלה (התא עליו לחץ המשתמש = הקובייה השמאלית עליונה של הצורה שהוגרלה).

**תיעוד והסבר הפיתרון:**

הפיתרון מתחיל מפעולות הממירות את המערך החד מימדי Board (לוח המשחק) לדו מימדי ולהפך, ומאותו רגע מתייחסים ללוח המשחק כמערך דו מימדי.

לאחר מכן כותבים פעולות הממירות מטריצה של תאים בזיכרון למטריצה שבה כל תא בה הוא ריבוע במסך.  
עושים זאת באמצעות קבוע המכפיל את X הנקודה וY הנקודה במטריצה שבזיכרון בקבוע הממיר מהמסך לזיכרון וחזרה (CONVERTER\_FACTOR) ולאחר מכן מציירים בעזרת פעולה ריבועים בגודל כך שיראו על המסך כמו מטריצה של ריבועים (מציירים ריבוע, מגדילים את X בקבוע, ושוב. כך עד סוף השורה ואז מגדילים את Y בקבוע, מחזירים את X להתחלת השורה ומגדילים את Y הנקודה בקבוע. ושוב יוצרים שורת ריבועים. כך עד שמגיעים לגודל המקסימלי של המטריצה בזיכרון (כל ריבוע שמייצרים, צבעו נקבע על פי מקומו בזיכרון בתוך המשתנה Board).

לאחר מכן כותבים פעולות הפוכות שממירות מטריצה במסך למטריצה בזיכרון.

כעת, לאחר שניתן להדפיס את המטריצה שבזיכרון על גבי המסך, נשאר רק לצייר את הצורות ליד המטריצה ולהתחיל בחלק הלוגי של הפרויקט.

תחילה כותבים פעולות קריאה מהמטריצה בזיכרון וכתיבה אליה (מטריצה של צבעים)  
לאחר מכן כותבים את הפעולות שמציירות את הצורות ליד המטריצה ואז מתחילים בחלק הלוגי:

ראשית כותבים פעולת Random שמחזירה מספר רנדומלי ב ShapeKind אליו מתייחסים כמספר של צורה ובכל פעם שרוצים לבדוק את הצורה הנוכחית, בודקים איזו צורה צריך לבדוק לפי המשתנה אליו הוכנס הערך הרנדומלי. (למשל המספר 2 מייצג צורה של שתי קוביות לאורך, אז אם המספר הוא 2, נצייר 2 קוביות למטה).

לאחר מכן כותבים את הפעולות המחשבות את התוצאות לתוך Score ואת הפעולות שקוראות וכותבות את הקובץ HighScor.txt בכדי לשמור על התוצאה של הניקוד הגבוהה ביותר גם לאחר היציאה מהתוכנית.  
כאשר יש שורות או טורים מלאים המשתנים Number\_Of\_Full\_Rows , Number\_Of\_Full\_Cols ו Number\_Of\_Full\_Lines משתנים בהתאם לפעולה שכן גם יוצרת את המשתנים השומרים את האינדקסים של השורות והטורים המלאים כך שיהיה אפשר למחוק אותם גם יחדיו. (Full\_Rows ו Full\_Cols).

המשתנה IsFull משמש לדעת האם השורה או הטור מלאים ואם כן יש להכניס אותם לתוך המשתנה השומר את האינדקסים.

לבסוף נשאר רק לכתוב פעולה המחכה ללחיצת עכבר ולאחר שקוראים לשאול איפה נלחץ העכבר ובהתאם לכך לפעול.

לאחר הלחיצה ישנם מספר בדיקות עם פעולות הבודקות האם הלחיצה הייתה במקום משמעותי בתוכנית ואם כן הפעולות בודקות על איזה מקום נלחץ ובהתאם לכך לפעול.

אם נלחץ על לוח המשחק Got\_Click\_On\_Mat)), יהיו בדיקות של פעולות האם ניתן לשים את הצורה במקום שנלחץ כלומר בדיקה האם המקום הוא ריק IsEmpty והאם לצורה יש מקום בכלל בלוח בכדי שעוד לפני שלוחצים יהיה אפשר להכריז בוודאות על הפסד CanPlace : הוא משתנה בתוך הפעולה הבודקת אם יש מקום במטריצה ו Mat\_Has\_Place הוא המשתנה המחזיר ערך של 1 או 0 (בוליאני) האם יש מקום לצורה במטריצה או לא.

לאחר הבדיקות, אם ניתן להכניס את הצורה למטריצה, היא תוכנס למטריצה בזיכרון בהתאם למשתנה ShapeKind וכן הScore יתעדכן, אם יש שורות או טורים מלאים יהייה הבהוב קצר שלהם ולבסוף המטריצה תצוייר על המסך מחדש וכן הScore ישתנה בהתאם.

אם נלחץ על כפתור ה "New Game" (Got\_Click\_On\_New\_Game), המשחק יאותחל ובהתאם הצורות ימחקו מהזיכרון ומהמטריצה במסך, הScore יתאפס ותוגרל צורה חדשה בתוך המשתנה ShaprKind.

אם נלחץ על לחצן ה "Rules" Got\_Click\_On\_Rules)), המשחק יעבור למסך החוקים וייתן אפשרות לצאת ממנו בעזרת הלחצן "Back" (Got\_Click\_On\_Back) .

אם נלחץ על לחצן היציאה (Got\_Click\_On\_Exit\_Program) התוכנית תסתיים ותצא.

**תרשימי זרימה:**

תרשים זרימה עבור: StartTurn""תרשים זרימה עבור:"Click on mat"

Draw new game option

Draw rules option

Print score

Draw shape

Draw board

Print high score

Get random shape

Check if can place the shape on click point

Yes

No

Input shape to memory

Calc score

Check if the score is bigger than the high score

Print score

Print high score

Clear board

Print board

Yes

Set high score

Write high score to file

No

תרשים זרימה עבור: "The Game 1010!"

No

Check click on mat

Yes

Make board empty

Read High Score file and save it in a variable

No

Check if game over

No

Yes

Check click on exit

"Start turn"

"Click on Mat"

New game page

No

Rules page

Yes

Check click on rules

Game over

Wait to left click

No

Check if have place for the shape in matrix

Yes

Yes

No

Yes

Check click on new game

**רשימת הפעולות:**

;========================================

**proc file\_open**

; Description: open HighScor file so we will be able to read or right.

; Input: nothing.

; Output: nothing.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc file\_close**

; Description: Close HighScor file so we will not be able to read or right.

; Input: nothing.

; Output: nothing.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc Write\_to\_File**

; Description: Read from the HighScor file 4 bytes, every byte is a number in hexa.

; Input: nothing.

; Output: the numbers from the file in ReadBuffer variable.

; Registers Usage: ax, bx, cx, dx.

;========================================

;========================================

**proc Read\_From\_File**

; Description: Read from the HighScor file 4 bytes, every byte is a number in hexa.

; Input: nothing.

; Output: the numbers from the file in ReadBuffer variable.

; Registers Usage: ax, bx, cx, dx.

;========================================

;========================================

**proc Write\_High\_Score\_To\_File**

; Description: Write the high score to HighScor file.

; An explain for the method:

; the HighScor file has 4 ascii digits in hexa .

; every digit in the file is ascii one byte.

; the problem is that we need every nibble digit to be a byte so we will be able to insert the value to the file.

; the solution is that I did some masks and commands like shr, shl and logical comendands: or, and ,not and xor to make every nibble become a byte in the momory.

; Then i add rto the numbers 30h and to the letters 37h to convert the numbers in the memory to ascii cose.

; Finally, I copied the 4 bytes in the High Score from the memory to the file.

; Input: nothing (the file).

; Output: Variable High\_Score with the right values from the file

; Registers Usage: ax, bx, cx, dx.

;========================================

;========================================

**proc Read\_High\_Score**

; Description: Read the high score from HighScor file.

; An explain for the method:

; the HighScor file has 4 ascii digits in hexa .

; every digit in the file is ascii one byte.

; the problem is that we need every hexa digit to be a nibble so we will be able to insert the value to High\_Score variable.

; the solution is that i read al the ascii hexa digits into 4 bytes in the memory, converted them to a number (buy subtracting 30h from the numbers qnd 37h fron the letters)

; and then I did some masks and commands like shr, shl and logical comendands: or, and ,not and xor to make every byte become a nibble in a register.

; finally I copied the register to the High\_Score variable.

; Input: nothing (the file).

; Output: Variable High\_Score with the right values from the file

; Registers Usage: ax, bx, cx, dx.

;========================================

;========================================

**proc Check\_Click\_On\_Exit\_Program**

; Description: check if the click were at the Exit Program rectangle borders.

; Input: nothing.

; Output: Got\_Click\_On\_Exit\_Program variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Draw\_Exit\_Program\_Rect**

; Description: draw the rect of the Exit Program option.

; Input: nothing.

; Output: rectangle at the wanted place.

; Registers Usage: ax, cx, dx, si, si.

;========================================

;========================================

**proc Put\_Cursor\_On\_Exit\_Program**

; Description: put the cursor on the start of the Exit Program option.

; Input: nothing.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc Check\_Click\_On\_Back**

; Description: check if the click were at the Back rectangle borders.

; Input: nothing.

; Output: Got\_Click\_On\_Back variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Draw\_Back\_Rect**

; Description: draw the rect of the Back option.

; Input: nothing.

; Output: rectangle at the wanted place.

; Registers Usage: ax, cx, dx, si, si.

;========================================

;========================================

**proc Put\_Cursor\_On\_Back**

; Description: put the cursor on the start of the Back option.

; Input: nothing.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc Check\_Click\_On\_Rules\_Option**

; Description: check if the click were at the Rules rectangle borders.

; Input: nothing.

; Output:; Output: Got\_Click\_On\_Rules variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Draw\_Rules\_Option\_Rect**

; Description: draw the rect of the Rules option.

; Input: nothing.

; Output: rectangle at the wanted place.

; Registers Usage: ax, cx, dx, si, si.

;========================================

;========================================

**proc Put\_Cursor\_On\_Rules\_Option**

; Description: put the cursor on the start of the Rules option.

; Input: nothing.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc Put\_Cursor\_On\_Top\_Left\_Screen**

; Description: put the cursor on the top left corner of the screen to write the pragraph of the rules.

; Input: nothing.

; Output: cursor in the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc Check\_Click\_On\_New\_Game\_Option**

; Description: check if the click were at the New Game rectangle borders.

; Input: nothing.

; Output:; Output: Got\_Click\_On\_New\_Game variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Draw\_New\_Game\_Option\_Rect**

; Description: draw the rect of the New Game option.

; Input: nothing.

; Output: rectangle at the wanted place.

; Registers Usage: ax, cx, dx, si, si.

;========================================

;========================================

**proc Put\_Cursor\_On\_New\_Game\_Option**

; Description: put the cursor on the start of the New Game option place.

; Input: nothing.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc Set\_Board\_Empty\_In\_Memory**

; Description: Set the board empty in the memory - it will happen at the beginning and evey time the player will choose New Game option.

; Input: nothing.

; Output:; Output: the board in the memory and on the screen is cleared and.

; Registers Usage: cx, bx.

;========================================

;========================================

**proc Put\_Cursor\_On\_Game\_Over**

; Description: put the cursor on the start of the Game Over place.

; Input: nothing.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc ClearScrean**

; Description: clear the screan with rect method.

; Input: nothing.

; Output: the screan cleared - black.

; Registers Usage: si, di, cx, dx, ax.

;========================================

;========================================

**proc Print\_High\_Score**

; Description: print the High Score: first print the string High\_Score\_String variable and then the High Score in decimal numbers - Score variable.

; Input: the variable High Score.

; Output: cursor at the wanted place.

; Registers Usage: ax, dx.

;========================================

;========================================

**proc Put\_Cursor\_On\_High\_Score\_Position**

; Description: put the cursor on the start of the High\_Score place.

; Input: X\_Of\_The\_Start\_High\_Score\_Point and Y\_Of\_The\_Start\_High\_Score\_Point are consts.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;========================================

**proc PrintScore**

; Description: print the score: first print the string Score\_String variable and then the score in decimal numbers - Score variable.

; Input: the variable Score.

; Output: cursor at the wanted place.

; Registers Usage: ax, dx.

;========================================

;========================================

**proc Put\_Cursor\_On\_Score\_Position**

; Description: put the cursor on the start of the Score place.

; Input: X\_Of\_The\_Start\_Score\_Point and Y\_Of\_The\_Start\_Score\_Point are consts.

; Output: cursor at the wanted place.

; Registers Usage: ax, bx, dx.

;========================================

;================================================

**proc ShowAxDecimal**

; Description - Write on screen the value of ax (decimal)

; the practice :

; Divide AX by 10 and put the Mod on stack

; Repeat Until AX smaller than 10 then print AX (MSB)

; then pop from the stack all what we kept there.

; INPUT: AX

; OUTPUT: Screen

; Register Usage: AX, BX, CX, DX

;================================================

\*\*\*\*\*\*\*\*\*\*;========================================

**proc Get\_Rnd\_In\_ShapeKind**

; Description: put a random number of shape in the variable ShapeKind (every shape by its friquance).

; Input: nothing

; Output: the random number in the variable ShapeKind.

; Registers Usage: ax, bx.

;========================================

;===========================================================

**proc RandomByCs**

; Description : get RND between any bl and bh includs (0 - 255).

; Input : 1. Bl = min (from 0) , BH = Max (till 255).

; 2. RndCurrentPos a word variable, help to get good rnd number

; Declre it at DATASEG : RndCurrentPos dw ,0 .

; 3. EndOfCsLbl: is label at the end of the program, one line above END start.

; Output: Al = rnd num from bl to bh (example: 50 - 150)

; More Info:

; Bl must be less than Bh.

; In order to get good random value again and again the Code segment size should be

; at least the number of times the procedure called at the same second ...

; for example - if you call to this proc 50 times at the same second -

; Make sure the cs size is 50 bytes or more

; (if not, make it to be more)

;===========================================================

;========================================

**Proc MakeMask**

; Description: make mask acording to bh size.

; Input: nothing

; Output: Si = mask put 1 in all bh range

; example: if bh 4 or 5 or 6 or 7 si will be 7

; if Bh 64 till 127 si will be 127

; Registers Usage: nothing.

;========================================

;========================================

**proc Calc\_Score**

; Description: set the Score after the turn acording to the shape that were inserted and the full lines

; input: nothing

; Output: Score variable ready to be printed on the screen (the variable is with right values)

; Registers Usage: nothing.

;========================================

;========================================

**proc Add\_Score\_Full\_Lines\_Points**

; Description: add to the Score the points from the full lines.

; if the number of full lines is, for example: 3, the points for the full lines points are: (1\*MAT\_HIGHT\_IN\_MEMORY + 2\*MAT\_HIGHT\_IN\_MEMORY + 3\*MAT\_HIGHT\_IN\_MEMORY)

; input: nothing

; Output: Score variable plus the full lins points.

; Registers Usage: nothing.

;========================================

;========================================

**proc Clear\_And\_Count\_Full\_Lines**

; Description: Clear the the full lines from the memory (put BlackColor - empty) and count how much full lines we have to calculate the Score.

; input: nothing

; Output: the lines that were full will be empty and in the variable Number\_Of\_Full\_Lines will be the number of lines that are full.

; Registers Usage: bx, cx, dx.

;========================================

;========================================

**proc List\_Full\_Rows**

; Description: put the x of the full rows in Full\_Rows array so we will be able to know were are the rows that need to be cleared.

; if we would have cleared the full rows before we knew the places (y) of the full cols, the method that clear the rows might clear also

; some cells from the full cols and then they will not be full and will not be cleard at the time that the methode that clear cols will run.

; input: nothing

; Output: the x of the every full row in Full\_Rows array in the memory matrix

; Registers Usage: bx, cx, dx.

;========================================

;========================================

**proc Check\_If\_Row\_Full**

; Description: check if a row is full (if every cell in the row is not empty) .

; input: dx = y of the row

; Output: IsEmpty variable is true (1) if the row is full, else false (0).

; Registers Usage: cx, dx.

;=======================================

;========================================

**proc Clear\_Row**

; Description: clear one row from the memory array

; input: dx = y of the row

; Output: every cell in the row is empty (black)

; Registers Usage: ax, cx, dx.

;========================================

;========================================

**proc List\_Full\_Cols**

; Description: put the y of the full cols in Full\_Cols array so we will be able to know were are the cols that need to be cleared.

; if we would have cleared the full cols before we knew the places (x) of the full rows, the method that clear the cols might clear also

; some cells from the full rows and then they will not be full and will not be cleard at the time that the methode that clear rows will run.

; input: nothing

; Output: the y of the every full col in Full\_Cols array in the memory matrix

; Registers Usage: bx, cx, dx.

;========================================

;========================================

**proc Check\_If\_Col\_Full**

; Description: check if a column is full (if every cell in the column is not empty) .

; input: cx = x of the column

; Output: IsEmpty variable is true (1) if the column is full, else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Clear\_Col**

; Description: clear one column from the memory array

; input: cx = x of the col

; Output: every cell in the column is empty (black)

; Registers Usage: ax, cx, dx.

;========================================

;========================================

**proc DrawShape**

; Description: draw the shape according to ShapeKind variable .

; Input: cx = X of the start shapes point in the screen.

; dx = Y of the start shapes point in the screen.

; si = the hight of the cell on the screen.

; di = the width of the cell on the screen.

; Output: the shape on the the screen.

; Registers Usage: nothing.

;========================================

;========================================

**proc Check\_If\_Can\_Place\_shape\_In\_Mat**

; Description: check if can place the shape (if one of the squars are empty) .

; Input: nothing (use ShapeKind variable)

; Output: Mat\_Has\_Place variable is true (1) if there is place on the matrix, else false (0).

; Registers Usage: nothing.

;========================================

;========================================

**proc Check\_if\_there\_is\_One\_Square\_Shape\_Empty\_In\_Mat**

; Description: check if one of the squars are empty. this proc doesn't use x or y because it goes one time on the array

; Input: nothing

; Output: Mat\_Has\_Place variable is true (1) if there is place on the matrix, else false (0).

; Registers Usage: cx, bx.

;========================================

;========================================

**proc Check\_if\_there\_are\_two\_Square\_Down\_Shape\_Empty\_In\_Mat**

; Description: check if can place the shape - if the matrix has a place for two squares that goes down.

; Input: nothing

; Output: Mat\_Has\_Place variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Check\_if\_there\_are\_two\_Square\_Right\_Shape\_Empty\_In\_Mat**

; Description: check if can place the shape - if the matrix has a place for for two squares that goes right.

; Input: nothing

; Output: Mat\_Has\_Place variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc Check\_if\_there\_are\_Two\_Squares\_Down\_Right\_Shape\_Empty\_In\_Mat**

; Description: check if can place the shape - if the matrix has a place for for two squares that goes right and down.

; Input: nothing

; Output: Mat\_Has\_Place variable is true (1) if there is place for the shape on the matrix , else false (0).

; Registers Usage: cx, dx.

;========================================

;========================================

**proc GetShapeColor**

; Description: put in al the color of the shape acording to type of the shae

; Input: nothing (use ShapeKind variable)

; Output: al = the color of the shape.

; Registers Usage: al.

;========================================

;========================================

**proc Check\_If\_Can\_Place\_shape\_On\_Click\_Point**

; Description: changr the color .

; Input: X of a point in the mat in memory, Y of a point in the mat in memory.

; the proc uses XClick And YClick.

; Output: CanPlace variable is true (1) if click was on the matrix, else false (0).

; Registers Usage: nothing.

;========================================

;========================================

**proc Check\_One\_Square\_Empty**

; Description: check if one square on the matrix is empty.

; Input: cx = X in memory

; dx = Y in memory

; Output: IsEmpty variable is true (1) if the square is empty, else false (0).

; Registers Usage: al, cx, dx.

;========================================

;========================================

**proc Check\_Two\_Squares\_Down\_Empty**

; Description: check if two squares that goes down on the matrix are empty.

; Input: cx = X in memory

; dx = Y in memory

; (1 = true = empty, else not empty)

; Output: IsEmpty variable is true (1) if the square is empty, else false (0).

; Registers Usage: al, dx, cx.

;========================================

;========================================

**proc Check\_Two\_Squares\_Right\_Empty**

; Description: check if two squares that goes right on the matrix are empty.

; Input: cx = X in memory

; dx = Y in memory

; (1 = true = empty, else not empty)

; Output: IsEmpty variable is true (1) if the square is empty, else false (0).

; Registers Usage: al, cx, dx.

;========================================

;========================================

**proc Check\_Two\_Squares\_Down\_Right\_Empty**

; Description: check if two squares that goes down and right on the matrix are empty.

; Input: cx = X in memory

; dx = Y in memory

; (1 = true = empty, else not empty)

; Output: IsEmpty variable is true (1) if the square is empty, else false (0).

; Registers Usage: al, cx, dx.

;========================================

;========================================

**proc Input\_Shape\_To\_Memory\_mat**

; Description: input the color of the shape in the memory matrix. .

; Input: X of a point in the matrix in memory, Y of a point in the matrix in memory.

; Output: the shape in the memory matrix.

; Registers Usage: nothing.

;========================================

;========================================

**proc Check\_Click\_On\_Mat**

; Description: check if there was click on the matrix - every shape have a different area

; where the mouse can be clicked according to the shape width and hight.

; Input: nothing (XClick and YClick is uses for this proc))

; Output: Check\_Click\_On\_Mat variable is true (1) if click was on the matrix, else false (0).

; Registers Usage: nothing.

;========================================

;========================================

**proc Draw\_One\_Square\_Shape**

; Description: draw the one square shape on the shapes point on the screen.

; Input: cx = X of the start shapes point in the screen.

; dx = Y of the start shapes point in the screen.

; si = the hight of the cell on the screen.

; di = the width of the cell on the screen.

; Output: the Shape on the screen.

; Registers Usage: cx, dx, si, di.

;========================================

;========================================

**proc Draw\_Two\_Squares\_Down\_Shapes**

; Description: draw two square shapes: start from the shapes point on the screen (Down Diraction).

; Input: cx = X of the start shapes point in the screen.

; dx = Y of the start shapes point in the screen.

; si = the hight of the cell on the screen.

; di = the width of the cell on the screen.

; Output: the Shape on the screen.

; Registers Usage: cx, dx, si, di.

;========================================

;========================================

**proc Draw\_Two\_Squares\_Right\_Shapes**

; Description: draw two square shapes: start from the shapes point on the screen (right Diraction).

; Input: nothing.

; Output: the Shape on the shapes point on the screen.

; Registers Usage: cx, dx, si, di.

;========================================

;========================================

**proc Draw\_Two\_Squares\_Down\_Right\_Shape**

; Description: draw two squares down and right shape: start from the shapes point on the screen.

; Input: nothing.

; Output: the Shape on the shapes point on the screen.

;; Registers Usage: cx, dx, si, di.

;========================================

;========================================

**Proc PrintBoard**

; Description: print the board with the colors on the screen

; Input: nothing

; Output: the Board with the colors on the screen but without the borders.

; Registers Usage: ax, bx, cx ,dx , si, di.

;========================================

;========================================

**Proc Clear\_Board\_From\_Screen**

; Description: Erase the board from the screen

; Input: nothing

; Output: the Board on the screen without the borders but all the colors are black so it earase the board from the screen but do not change any thing in the memory.

; Registers Usage: ax, bx, cx ,dx , si, di.

;========================================

;========================================

**proc Draw\_Board\_Borders**

; Description: Drawing the borders of the Board on the screen considering all the properties of the board

; top and left of the screen, hight and width of Board in memory and on screen (giving names and not numbers so we can change any time).

; Input: nothing

; Output: Borders at the Board place.

; Registers Usage: ax, bx, cx.

;========================================

;========================================

**proc Convert\_Screen\_Y\_To\_Memory\_Y**

; Description: convert a y of a point on the screen to a y of a point in the memory matrix.

; Input: ax = a y of a point in the screen

; OutPut: dx = the y of the point in the memory

; Registers Usage: ax, dx.

;========================================

;========================================

**proc Convert\_Screen\_X\_To\_Memory\_X**

; Description: convert a x of a point on the screen to a x of a point in the memory matrix.

; Input: ax = a x of a point in the screen

; OutPut: dx = the x of the point in the memory

; Registers Usage: ax, cx.

;========================================

;========================================

**proc DivFromScreenToMemory**

; Description: div ax by 17 because every cell in the memory mat is 17 pixels on the screen.

; Input: ax - the x or the y of a point on the screen.

; Output: al - the x or the y of a point in the memory.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc Convert\_Memory\_Y\_To\_Screen\_Y**

; Description: convert a Y of a point in the memory matrix to a Y of a point on screen.

; Input: ax = a Y of a point in the memory

; OutPut: dx = the Y of the point in the screen

; Registers Usage: ax, dx.

;========================================

;========================================

**proc Get\_Memory\_Y\_In\_Memory\_Place\_In\_Arr**

; Description: putting into ax the Y in memory "matrix" of a place in the memory oneD Array (dividing by mat hight and getting the result.

; Input: ax - a place (not offset) in the oneD Mat Array.

; Output: ax - the memory Y in the matrix of the memory place in oneD arr.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc Convert\_Memory\_X\_To\_Screen\_X**

; Description: convert a x of a point in the memory matrix to a x of a point on screen.

; Input: ax = a x of a point in the memory

; OutPut: dx = the x of the point in the screen

; Registers Usage: ax, cx.

;========================================

;========================================

**proc Get\_Memory\_X\_In\_Memory\_Place\_In\_Arr**

; Description: putting into ax the X in memory "matrix" of a place in the memory oneD Array (dividing by mat hight and getting the mod.

; Input: ax - a place (not offset) in the oneD Mat Array.

; Output: ax - the memory X in the matrix of the memory place in oneD arr.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc MulFromMemoryToScreen**

; Description: mul ax by 17 because every cell in the memory mat is 17 pixels on the screen.

; Input: ax - the x or the y of a point in the memory.

; Output: ax - the x or the y of a point on the screen.

; Registers Usage: ax, bx.

;========================================

;========================================

**proc Read\_Color\_From\_One\_Memory\_Cell**

; Description: Read one color from 2D array cell.

; Input: dx = row (y)

; cx = col (x)

; Output: al = Color

; Registers Usage: ax, bx, cx, dx, si

;========================================

;========================================

**proc Input\_Color\_To\_One\_Memory\_Cell**

; Description: input a color to one cell in the memory matrix using the oneD array.

; Input: cx = X in the memory

; dx = Y in the memory

; al = color

; Output: the colour in its place in the memory matrix

; Registers Usage: ax, bx, cx, dx, si

;========================================

;========================================

**proc Input\_Color\_To\_Two\_Right\_Memory\_Cells**

; Description: input a color to two cells (Right Diraction) in the memory matrix.

; Input: cx = X in the memory

; dx = Y in the memory

; al = color

; Output: the two colours in their place in the memory matrix

; Registers Usage: cx

;========================================

;========================================

**proc Input\_Color\_To\_Two\_Down\_Memory\_Cells**

; Description: input a color to two cells (down Diraction) in the memory matrix.

; Input: cx = X in the memory

; dx = Y in the memory

; al = color

; Output: the two colours in their place in the memory matrix

; Registers Usage: dx

;========================================

;========================================

**proc Rect**

; Description: draw a rectangle on the screen using DrawVerticalLine method.

; Input: si - hight

; di - width

; cx,dx (col,row) - start point of the Rectangle (going right and down)

; al - the color of the rectangle

; Output: a rectangle on the screen

; Registers Usage: ax, si, di, dx, cx

;========================================

;========================================

**proc DrawHorizontalLine**

; Description: draw one Horizontal Line on the screen using int 10,c (put pixel)

; Input: si - how much pixels to draw

; cx,dx (col,row) - start point of the Horizontal line - going right

; al - the color

; Output: one Horizontal Line on the screen

; Registers Usage: ax, si, cx (dx doesn't change).

;========================================

;========================================

**proc DrawVerticalLine near**

; Description: draw one Vertical Line on the screen using int 10,c (put pixel)

; Input: si - how much pixels to draw

; cx,dx (col,row) - start point of the Vertical line - going Down

; al - the color

; Output: one Vertical Line on the screen

; Registers Usage: ax, si, dx (cx doesn't change).

;========================================

;========================================

**proc Wait\_To\_Left\_Click\_Press**

; Description: wait until there is a left click Press on the mouse

; Input: nothing.

; Output: Xclick and Yclick variables have the point of the click on the screen.

; Registers Usage: ax, bx, cx, dx, si

;========================================

;========================================

**proc SetGraphic**

; Description: Set the screen to Mode 13h - an IBM VGA BIOS mode. It is the specific standard 256-color mode. 320 X 200 pixels.

; Input: nothing

; Output: mode change to graphic

; Registers Usage: ax

;========================================

;========================================

**proc SetText**

; Description: Set the screen to text mode 80 X 45 pixels.

; Input: nothing

; Output: mode change to graphic

; Registers Usage: ax

;========================================

;========================================

**proc ShowMouse**

; Description: Show the mouse on the screen.

; Input: nothing

; Output: mouse on the screen

; Registers Usage: ax

;========================================

;========================================

**proc HideMouse**

; Description: Hide the mouse from the screen.

; Input: nothing

; Output: hide mouse from the screen

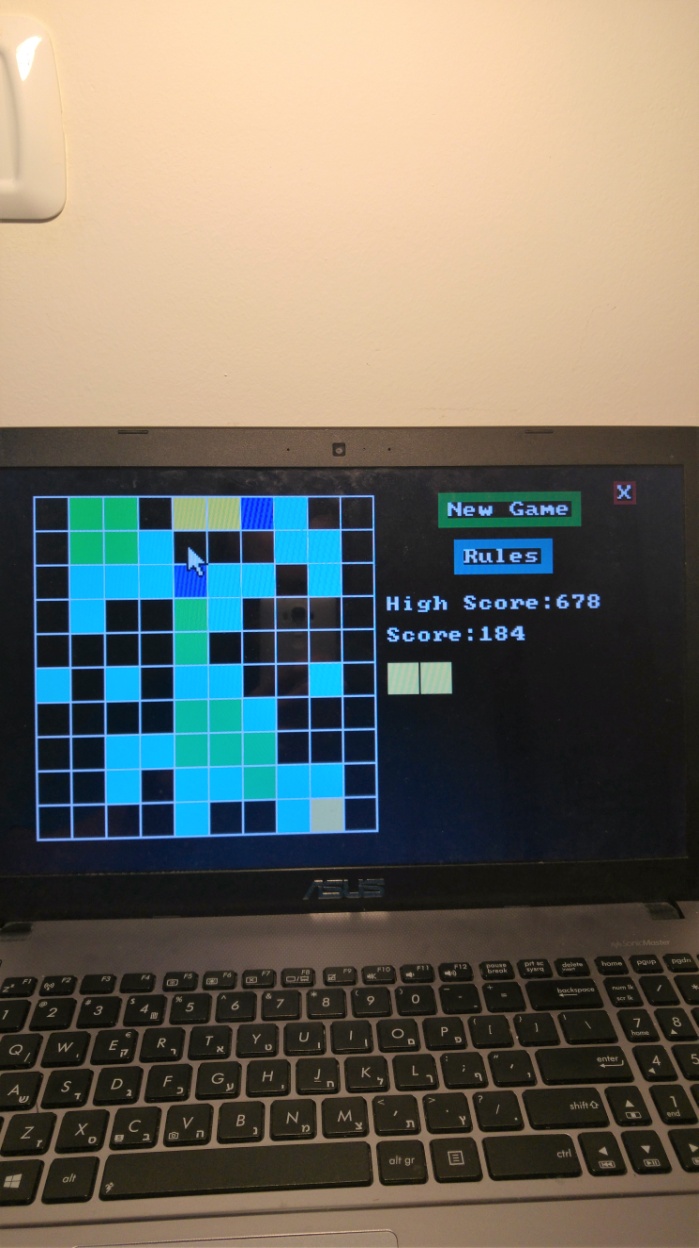
; Registers Usage: ax

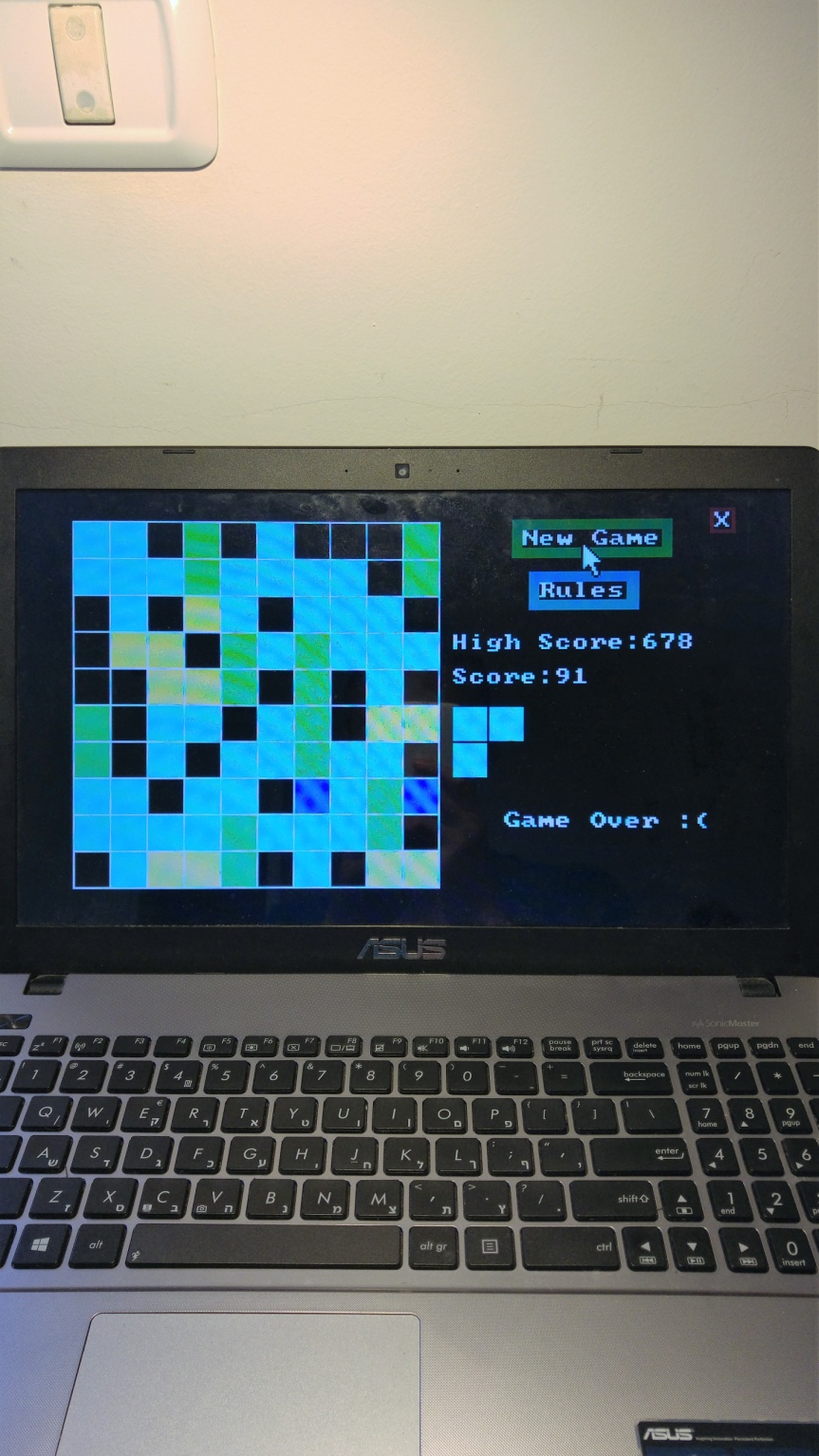
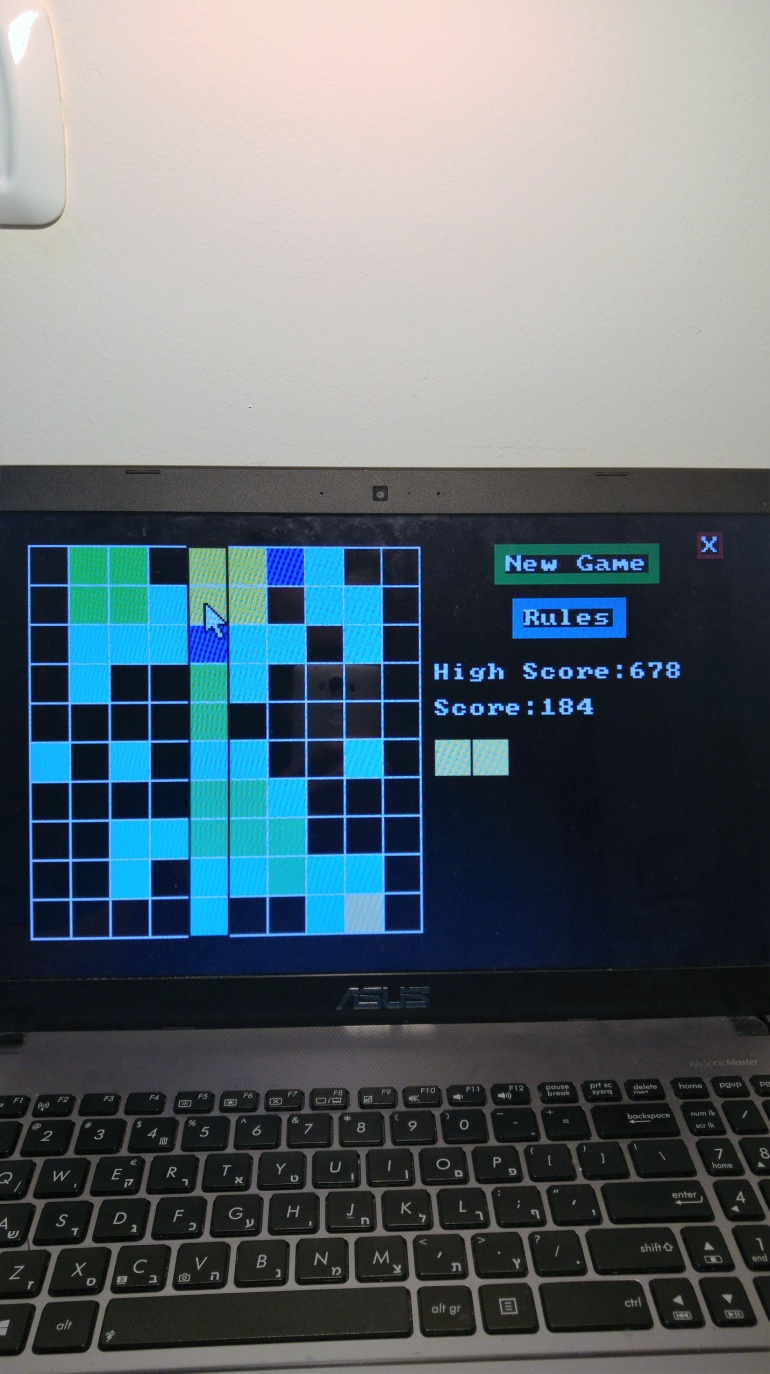
;========================================

**דוגמאות הרצה:**

המשחק כבר החל ויש צורות על הלוח.

התחלת המשחק: התוכנית מחכה ללחיצת עכבר

****

****

תמונה לאחר לחציה, בזמן הבהוב שורה לפני היעלמותה מן המסך

השחקן הפסיד ובאפשרותו לשחק משחק חדש, להציץ בחוקים או לצאת מהתוכנית

**סיכום אישי:**

אני זוכר שכשהתחלנו לדבר בכיתה על הפרויקט הייתי מאוד נרגש.

רציתי כבר לחשוב על רעיון לפרויקט ורציתי כבר לתכנן ולעשות אותו, אולם עד מהרה הבנתי כי אין לי מה למהר וכי עליי לתכנן את שלבי הפרויקט עוד לפני כתיבתו בכדי לא להסתבך באמצע כתיבת הקוד.

כך למדתי כל פעם דבר חדש על הטכניקות איתן אני צריך להשתמש כאשר אני כותב קוד.

לאט לאט התגבש לו המשחק שיצרתי (שהיה אמור להיות טטריס בהתחלה) והתחלתי לעבוד.

בתחילת הכתיבה מצאתי את עצמי עבוד ולא הבנתי כיצד אני פותר את הבעיות השונות אך בעזרת יוסי וקצת חשיבה הצלחתי לבנות את הפרויקט.

כתיבת הקוד הרבה פעמים הייתה מסובכת, ארוכה ובעיקר לא מובנת אך כאשר התחלתי לתעד כל דבר, הכול החל להיות ברור יותר ויכולתי לעצור מתי שרק רציתי ולחזור לכתוב את הקוד ביום אחר.

במהלך כתיבת הפרויקט באסמבלי למדתי המון דברים על תכנות, לדוגמא כמה חשוב לתעד כל דבר בתוכנית או לדעת להפסיק לכתוב ולא להיות שקוע במחשב הרבה זמן (יש לי אימוני טניס).

בנוסף, הפרויקט באסמבלי סייע לי להבין הרבה יותר טוב את החומר שנלמד בכיתה ובעיקר אני מרגיש שיפור הרמת הידע שלי בפעולות הלוגיות or, and, xor, not בהן השתמשתי בכדי לקרוא ולכתוב אל קבצים.

בנוסף, אני מרגיש שיפור משמעותי גם במתן שמות המשתנים שלי בתוכנית ובבהירות ויעילות הקוד שכן דאגתי לכתוב קוד קצר כל עוד יכולתי וחילקתי פעולות לתתי פעולות וכו.

אני זוכר שבחופש הגדול שלפני תחילת הלימודים, כאשר עוד לא ידעתי מה פירוש המילה מתכנת, עניין אותי מאוד לדעת מה אדע בסוף השנה והאם בכלל אשרוד את השנה.  
היום אני יודע כמה שהתקדמתי ואני אפילו כמעט לא מאמין שכתבי משחק מחשב ועוד בשפת התכנות המורכבת ביותר אסמבלי.

לסיכום, נהניתי מאוד לכתוב את הפרויקט הרבה יותר ממה שחשבתי ואני שמח שיצא לי להיחשף לשפת אסמבלי שכן היא הבסיס לכל השפות וכעת אני מרגיש ביטחון רב ויתרון על פני מתכנתים אחרים שלא יודעים כיצד המחשב באמת פועל.

הדבר שלפי דעתי הכי הלהיב אותי בפרויקט הוא בעיקר יישום הרעיונות שעברו בראשי פיזית, בעזרת המחשב, דבר שלא הייתי מאמין שאהיה מסוגל לעשות לפני כתחילת הלימודים וכך הצלחתי ליצור את הפרויקט שדמיינתי ולכן אני מאוד מרוצה מעצמי וגאה בפרויקט שלי.