****

**פרויקט סיום באסמבלר**

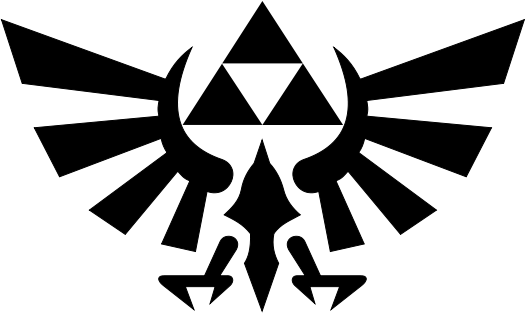
**Legend of Zelda**

**נושא העבודה:** משחק מלחמה נגד מפלצות

**שם המגיש:** עידו גרזון

**ת"ז של המגיש:** 323011825

**שם המורה:** גלית קלאוס



תוכן עניינים

[נושא הפרויקט ולמה בחרתי בו 3](#_Toc482644635)

[סביבת העבודה 4](#_Toc482644636)

[הוראות המשחק 5](#_Toc482644637)

[תרשים זרימה 6](#_Toc482644638)

[פרוצדורות 7](#_Toc482644639)

[משתנים 8](#_Toc482644640)

[מסכי התכנית 9](#_Toc482644641)

[קוד התכנית 10](#_Toc482644642)

[סיכום ורפלקציה 11](#_Toc482644643)

[תודות 11](#_Toc482644644)

# נושא הפרויקט ולמה בחרתי בו

כאשר שמעתי שעלינו לתכנת פרוייקט סיום בסביבת אסמבלר אז חשבתי שמכיוון שאנחנו מתכנתים בסביבה ישנה לכן עלינו לתכנת משחקים ישנים ישר חשבתי על משחק שאני מאוד אוהב Legend of Zelda נוצרה בשנת 1987 על ידי שיגרו מיאמוטו וגם המשחק הזה נכתב בסביבה ישנה לכן רציתי לתכנת משחק דומה

# סביבת העבודה

את הקוד כתבתי בnotepad++ תוכנה נוחה לכתיבת טקסט שמאכלסת בתוכה את סביבת אסמבלר ומסוגלת לזהות פקודות באסמבלר על מנת להריץ את הקוד נאלצתי לעבוד כמה שלבים ועל מנת לעשות אותם היה עליי להוריד קובץ שנקרא tasm שם תחילה השתמשתי בפקודה tasm הפקודה פעלה על קבצי asm בדקה באגים ואם היא לא מצאה היא הפכה אותם לקבצים מסוג obj לאחר מכן השתמשתי בפקודה tlink כדי להכין את הקובץ להפיכה לקובץ הרצה tlink הפכה obj לexe ולאחר שהיה לי קובץ מסוג זה הרצתי אותו בdosbox . דוסבוקס היא תוכנה המריצה קבצי exe אם היה ברצוני לעבור על הקוד שלי ולמצוא טעויות בדוסבוקס פניתי לפקודת td שבעזרתה בדקתי את הקוד ואת ההשפעה שלו על המשתנים וכו... למעשה בעזרת כל הצעדים הללו דימינו סביבת עבודה על מעבד 8086. בנוסף לתוכנות הללו השתמשתי בgithub תוכנה זו עזרה לעקוב אחרי גרסאות של הפרויקט ובכך לעשות מעקב של ההתקדמות שלי לכיוון מטרת הסיום נעזרתי גם בגוגל דרייב שכל פעם שעשיתי שינוי לא משנה איזה הוא גיבה את הפרויקט וככה דאג שלא אאבד מידע.

# הוראות המשחק

בעזרת המקשים: w,a,s,d ניתן להזיז את הדמות הכחולה על המסך בעזרת מקש e ניתן לתקוף את הדמויות האדומות לא ניתן לעבור לשטחים הירוקים ולא ניתן לצאת מהזירה דרך השבילים

# תרשים זרימה



# פרוצדורות

**Print\_black\_board:**this procedure is not in the game but I used it beta tests to print empty  
screen   
this is in text mode so to print pixel I used the ascii code 219 (also called DB) the character is a full text slot size so you can print pixel in colors using this char  
Input: none  
output: none  
goal: print black screen

**print\_game\_board:**this procedure prints the game board with the main character in the middle in the start position using another procedure that prints squares in colors  
inputs: the offset of the player x, the offset of the player   
outputs: none  
goal: prints game start board with the player

**Print\_square:**this procedure prints squre in selected color starting in the starting pixel that is given by the programmer and ending the square in a selected position pushed by the programmer  
it print using another function that prints 1 pixel  
inputs: starting x, starting y, ending x, ending y, and color to print the square  
outputs: none  
goal: print square in color

**Check\_for\_press:**this procedure is one of the main procedures of the program using int 16h ah=1 interrupt this procedure read the last value that entered the keyboard buffer  
so using that the player move and attack  
inputs: the offset of x player, the offset of y player, is legal offset (a helper var, see above),is near offset (a helper var, see above),array containing x values of monsters, array containing y values of monsters  
array containing hp values of monsters, num position offset (a helper var, see above),counter var of live monster  
outputs: updated x and y values, live monster counter updated  
goal: respond to player clicking on the keyboard by action that suits the press

**Mov\_up:**this procedure prints brown pixel in previous position and colors pixel in updated location the color is chosen by the user the updated position is up to the previous  
inputs: x offset, y offset, color of the pixel  
outputs: none   
goal: "moving" pixel place by replacing the last position with the new

**Mov\_down:**this procedure prints brown pixel in previous position and colors pixel in updated location the color is chosen by the user the updated position is down to the previous  
inputs: x offset, y offset, color of the pixel  
outputs: none   
goal: "moving" pixel place by replacing the last position with the new

**Mov\_right:**this procedure prints brown pixel in previous position and colors pixel in updated location the color is chosen by the user the updated position is right to the previous  
inputs: x offset, y offset, color of the pixel  
outputs: none   
goal: "moving" pixel place by replacing the last position with the new

**Mov\_left:**this procedure prints brown pixel in previous position and colors pixel in updated location the color is chosen by the user the updated position is left to the previous  
inputs: x offset, y offset, color of the pixel  
outputs: none   
goal: "moving" pixel place by replacing the last position with the new

**Check\_array:**this procedure is checking if a number is in a array if it's in the array in returns it's position in a var if it's not in the array it returns 70 (high impossible value) in the var  
inputs: number to search, the array offset, array length, and a num position offset (a helper var, see above)  
outputs: number position in array if it's not in the array returns 70 (returns using var)

**Print\_char:**this procedure prints the ascii 219 char which is a full text slot (also called DB) in a given value and color  
inputs: x, y, color  
outputs: none  
goal: print a pixel in a given location and color

**Timer:**this procedure is waiting a given time of milliseconds using the ah 2ch interrupt to get system time and moving it to a register and calling it again until the equality is broken and then a millisecond passed  
and then doing that a given time  
inputs: milliseconds to wait  
output: none   
goal: waiting a milliseconds

**Timer1:**same as timer procedure but here there is the check\_for\_press procedure so that even when it waits time the player could move  
inputs: position x of player offset, position y of player offset, is legal (helper var, see above),is near (healper var, see above),monster x array offset, monster y array offset, monster health offset  
num position offset (helper var, see above),live monster counter  
outputs: none  
goal: wait time and let the player move in that time

**Random\_num:**  
 this procedure randomize a number zero to give number (not included) it calls system time and divide the milliseconds in a number that can create one of your random options  
for example: the divider sent-25 and we divide it by system milliseconds for exmple:46 46/25=1 (and ignoring the reminder) we get our number 1 (which is from 0-3 4 options)  
input: random num offset (helper var, see above),divider which is the number to divide  
output: using the rand\_num\_offset pushed value we output the result through  
goal: randomize num using divider

**Play\_sound:**this procedure play sound from given number that number can be achieved by dividing a constant number with the frequency of the note and outputting that number to ports using in and out   
inputs: note after dividing, time to play that note

**Legal\_place:**this procedure check if the place is legal using a position given by the programmer it also gets a direction from the programmer to know the movement   
inputs: x, y, is legal offset (helper var, see above),direction the direction of the move   
outputs: it outputs in is\_legal 1 if the move is legal and 0 if it's illegal   
goal: check if a move is legal or illegal and do "beep" if illegal

**Spawn\_mons:**this procedure randomize a spawn from 4 roads and in every road it randomize specific slot in the road  
inputs: player x offset, player y offset, is legal offset (helper var,see above),is near offset (helper var, see above), monster x array offset, monster y array offset, monster health array  
num position offset (helper var,see above),live monster counter, random number offset (helper var, see above), monster counter offset  
outputs: updating the x and y monster array, updating monster health when spawning (3)  
goal: spawn a monster in a randomized position

**Mov\_mons:**  
;this procedure make the monster move after the player using his position and the legal\_place procedure check for press is also here to give the player a fast response  
inputs: player x offset, player y offset, is legal offset (helper var,see above),is near offset (helper var, see above), monster x array offset, monster y array offset, monster health array  
monster counter offset, x helper offset (helper var, see above), y helper offset (helper var, see above), num position offset (helper var, see above),live monster counter  
outputs: updated monster position  
goal: monster that follows the player

**Check\_near:**  
this procedure gets a position and it checks the pixels near that position if the color match the color is given by the programmer  
inputs: x, y, is near offset (helper var, see above),color  
outputs: using is near offset array it moves 1 to every direction where the color that is given by the programmer is equal to the color near that pixel  
goal: check if there is a monster or a player near a given position

**Mons\_attack:**this procedure checks if a user is near her and the she attacks him   
inputs: player x offset, player y offset, is legal offset (helper var,see above),is near offset (helper var, see above), monster x array offset, monster y array offset, monster health array  
monster counter offset, num position offset (helper var,see above),hp player offset, attacked offset (helper var, see above),live monster offset  
outputs: it's updating the hp of the player if attack  
goal: to attack the player and beep

**Clear\_array:**  
this procedure changes all the values of an array to zero  
inputs: array offset, array length  
outputs: none  
goal: reset an array

**Manage\_game:**this procedure is running one game using all procedures and vars  
inputs: every var except for var of printing a bmp (see above)  
outputs: none  
goal: run one game

**OpenFile:**opens a file  
input: offset file name  
output: file's handle

**ReadHeaderPalette:**;Reads Header and Palette  
input: file handle, offset to put header, offset to put Palette  
output: changes the given places the the header and palette  
moves the reading pointer in the file to the start of the actual image

**Copypal:**  
;chnanges the colors from BGR (assembly color format) to RGB (BMP file color format)  
input: offset to read Palette from  
output: the colors in the ports are changed

**Copybitmap:**prints to the graphic screen the BMP file (after opening file, reading palette and copying it)  
input: file handle  
output: copying the BMP file from the file to the data segment to the A000 segment, the graphics screen

**Closefile:**  
closes an open file  
input: file handle  
output: none

**Printbmp:**this procedure prints to the screen a BMP file, doing all the things needed  
opening the file, reading header, reading palette, copying the palette, copying the BMP, and closing the file  
input: offset of file's name  
output: moving to graphics mode and chaning the screen to the given BMP file

# משתנים

x\_player dw 40 ;it remembers the x of the player

y\_player dw 12 ;it remembers the y of the player

hp\_player db 3 ;it remembers the health of the player

rand\_num db ? ;after the randomize num procedure this num is put in this var

is\_legal db ? ;after checking the place in legal\_place procedure it puts 1 or 0 in this if the move is legal or illegal

x\_mons\_arr dw ?,?,?,?,?,?,?,?,?,?,?,?,?,?,? ;array of all the x of the monsters

y\_mons\_arr dw ?,?,?,?,?,?,?,?,?,?,?,?,?,?,? ;array of all the y of the monsters

hp\_mons\_arr db ?,?,?,?,?,?,?,?,?,?,?,?,?,?,? ;array of the health of every monster

mons\_num dw 0 ;it's the monster counter now

mons\_x\_mov\_helper dw ? ;it helps with x values in the mons\_mov procedure

mons\_y\_mov\_helper dw ? ;it helps with y values in the mons\_mov procedure

is\_near db ?,?,?,? ;it helps in the check\_near procedure if the first place is 1 = there's a painted pixel on the left if the second one is 1 = there is a pixel down and etc...

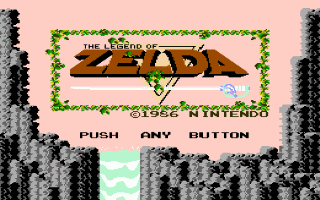
num\_position dw ? ;it helps in check\_array procedure using this the procedure returns the value with this var

live\_mons db 1 ;how many monsters to print this round

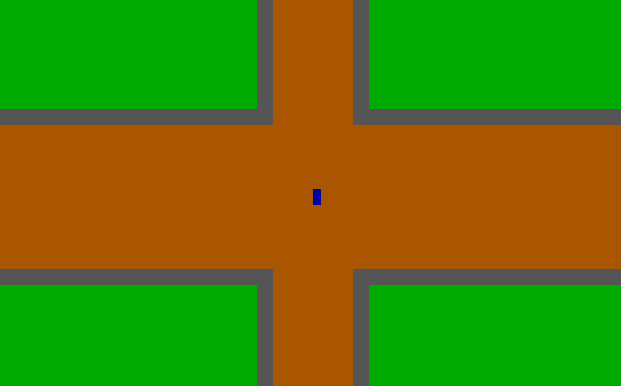
attacked db ? ;it helps in the mons\_attack if the mons\_attacked the value is changed to 1

# C:\Users\Ido-Gaming-PC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.bmpמסכי התכנית בשביל המסך פתיחה יש 6 תמונות שרצות מהר אחת אחרי השנייה ונותנות את ההרגשה של אנימציה: תמונה 1:

# תמונה 6:

****

התמונה האחרונה היא ההתחלה של הראשונה דבר שיוצר את ההרגשה  
בנוסף ישנו את לוח המשחק שבנוי מארבעה שבילים ו4 אזורי דשא שלא ניתן לעבור בהם



# קוד התכנית

# סיכום ורפלקציה

העבודה הזו הייתה מאוד מעניינת וכיפית. היא דרשה ממני השקעה וזמן על מנת להגיע למוצר המוגמר שהגעתי אליו כיום, הייתי צריך לחשוב הרבה בשביל ליצור את המשחק ולהגיע לאלגוריתם הטוב ביותר, בעקבות כך חלק שהיה לי קשה במיוחד היה התנועה של המפלצות אחרי השחקן, איך לגרום לדבר לא אמיתי, בלי שחקן מאחוריו לתת הרגשה שאכן יושב מישהו שמנסה להתקרב אליך אחרי ההצלחה הראשונה שלי בענף זה הייתי שמח והדברים וההצלחות הלוו היו אלו שהיוו את ההנאה, הדברים שהביאו את עניין היו המחקר לבד שהייתי צריך לעשות כדי להגיע לפרויקט כמו שהגעתי, למידה לבד שזה משהו שתמיד הייתי עושה אך לא ברמה הזאת שעשיתי למען הפרויקט הזה, אך לבסוף עברתי את זה ולמדתי מה שהייתי צריך בשביל המשימה.  
לסיכום הפרויקט הזה היה מאוד משמעותי לדעתי אפשר להגיד שהוא זה שבאמת פתח אותנו לתכנות אמיתי ומתמשך בנוסף עם הצורך לניהול גרסאות כאשר בזכות כל אלו למדנו איך זה באמת לתכנת פרויקט.

# תודות ברצוני להודות לחברי שתרמו לי כל פעם שלא הצלחתי משהו הם עזרו לי בניהם, ניר דפנאי, ניר ארואטי ובן קירשנבאום ברצוני להודות למורה שלי שגם היא תרמה לי כשלא הצלחתי גלית קלאוס ולבסוף אודה לאורי עוזר ההוראה שגם תרם רבות להצלחת הפרויקט שלי