פרויקט משחק מבוך בשפת assembly

דניאל וולקוביץ'



# תוכן עניינים

[תוכן עניינים 2](#_Toc111853017)

[מבוא 3](#_Toc111853018)

[מדריך משתמש למוצר 4](#_Toc111853019)

[כללים בסיסיים: 4](#_Toc111853020)

[דוגמא: 4](#_Toc111853021)

[סיום המשחק: 5](#_Toc111853022)

[אמצעי הקלט 5](#_Toc111853023)

[סביבת עבודה 6](#_Toc111853024)

[סביבת פיתוח 6](#_Toc111853025)

[סביבת הרצה 6](#_Toc111853026)

[פונקציות שסופקו ע"י מנחה הפרויקט 6](#_Toc111853027)

[מבנה תוכנה 7](#_Toc111853028)

[תרשים זרימה ראשי (UML Activity diagram) 7](#_Toc111853029)

[עץ קריאות – מקרא 8](#_Toc111853030)

[8](#_Toc111853031)

[עץ קריאות ראשי 9](#_Toc111853032)

[תיאור פונקציות עיקריות 10](#_Toc111853033)

[maze\_getxy 10](#_Toc111853034)

[draw\_maze 10](#_Toc111853035)

[draw\_char\_at\_position 10](#_Toc111853036)

[wait\_for\_key\_pressed\_and\_then\_convert\_it 10](#_Toc111853037)

[Key\_to\_new\_position 10](#_Toc111853038)

[check\_legal\_move 10](#_Toc111853039)

[check\_win 10](#_Toc111853040)

[תיאור בעיות ואתגרים במהלך הפרויקט והתמודדות עימן 11](#_Toc111853041)

[המיפוי של המבוך בזיכרון 11](#_Toc111853042)

[יציאה מהמבוך 11](#_Toc111853043)

[הצעות להרחבה עתידית 13](#_Toc111853044)

[בניית מספר מבוכים (שלבים) 13](#_Toc111853045)

[בניית מבוך אקראי 13](#_Toc111853046)

[תזוזה באלכסון 13](#_Toc111853047)

[מעבר מצד לצד 13](#_Toc111853048)

[פתירת המבוך 13](#_Toc111853049)

[זמן מוקצב לפתירת המבוך 14](#_Toc111853050)

[תודות 15](#_Toc111853051)

[נספחים 16](#_Toc111853052)

[נספח קוד מקור 16](#_Toc111853053)

# מבוא

בחרתי לכתוב תוכנית אשר מייצרת משחק ובו על המשתמש לצאת מן מבוך נתון. בחרתי לכתוב תוכנית בנושא זה, מכיוון שאני חושב שמבוכים הם משחקים מאתגרים אשר מחדדים את המחשבה וכי רציתי להבין כיצד אוכל לייצר משחק שכזה בשפת סף בכוחות עצמי.

התוכנית נכתבה בשפת אסמבלי של 8086 עבור יחידת הבגרות במדמ"ח בשפת אסמבלי וכחלק מפרויקט "גבהים – מצוינות בסייבר".

מסמך זה מתאר את המוצר, מדריך משתמש למוצר, סביבת הפיתוח בה נעשה שימוש ואת מבנה התוכנית. כמו כן יצוינו הבעיות איתן התמודדתי במהלך הפרויקט.

# מדריך משתמש למוצר

הפרויקט הינו מימוש של משחק המורכב ממבוך, זאת כאשר מטרת המשתמש היא לצאת מן המבוך.

## כללים בסיסיים:

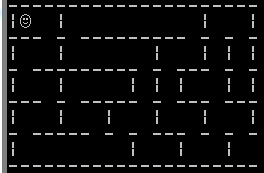
1. המשחק נועד לשחקן אחד בלבד, זאת כאשר השחקן מסומן במסך על ידי אימוג'י.
2. כל מהלך מבוצע על לוח המשחק בצורה אינטראקטיבית.
3. בכל תור לאחר שיחליט השחקן לאן תזוזתו הבאה תתבצע, ולאחר שיבצע את בחירתו ע"י לחיצה על אחד מכפתורי החיצים, האימוג'י יזוז בהתאם למקש המקלדת הנבחר.
4. אם נמצא קיר ביעד התזוזה של הסמיילי התזוזה לא תתבצע.

## דוגמא:

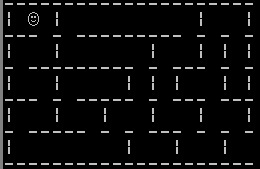
* האימוג'י:



* הקירות:
* תחילת המשחק (הסמיילי מוצב בעמדה ההתחלתית):



* תנוחת הסמיילי לאחר תזוזה ימינה:



במצב זה הסמיילי אינו יכול לנוע כלפי מעלה או מטה מכיוון שהקירות חוסמים אותו.

## סיום המשחק:

המשחק מסתיים כאשר השחקן מגיע אל היציאה מהמבוך או כאשר הוא מחליט לפרוש.

## אמצעי הקלט

* Q - מקש פרישה מהמשחק
* חצי ימינה, שמאלה, למעלה, למטה – תזוזה של הסמיילי בלוח המשחק



פרישה

מהמשחק

תזוזות

השחקן

# סביבת עבודה

## סביבת פיתוח

סביבת הפיתוח הינה התוכנה EMU8086.

התוכנה הינה אימולטור למחשב 8086, עם אוגרים בגודל של 16 ביט.

האימולטור הינו סביבת עבודה משולבת (IDE), הכוללת:

* עורך מסמכים
* אסמבלר
* דיבאגר
* אימולטור – הכולל מספר מוגבל של שירותי BIOS ו-DOS (Interrupts).

מודל הזיכרון הנבחר הוא: תוכנית COM, הכולל סגמנט אחד של זיכרון, בגודל 64K.

יש לציין כי לאימולטור קיים חיסרון משמעותי ביותר - הרצת התוכנית איטית מאוד.

## סביבת הרצה

סביבת ההרצה היא חלון ריצה של תוכנית ב-Emu8086.

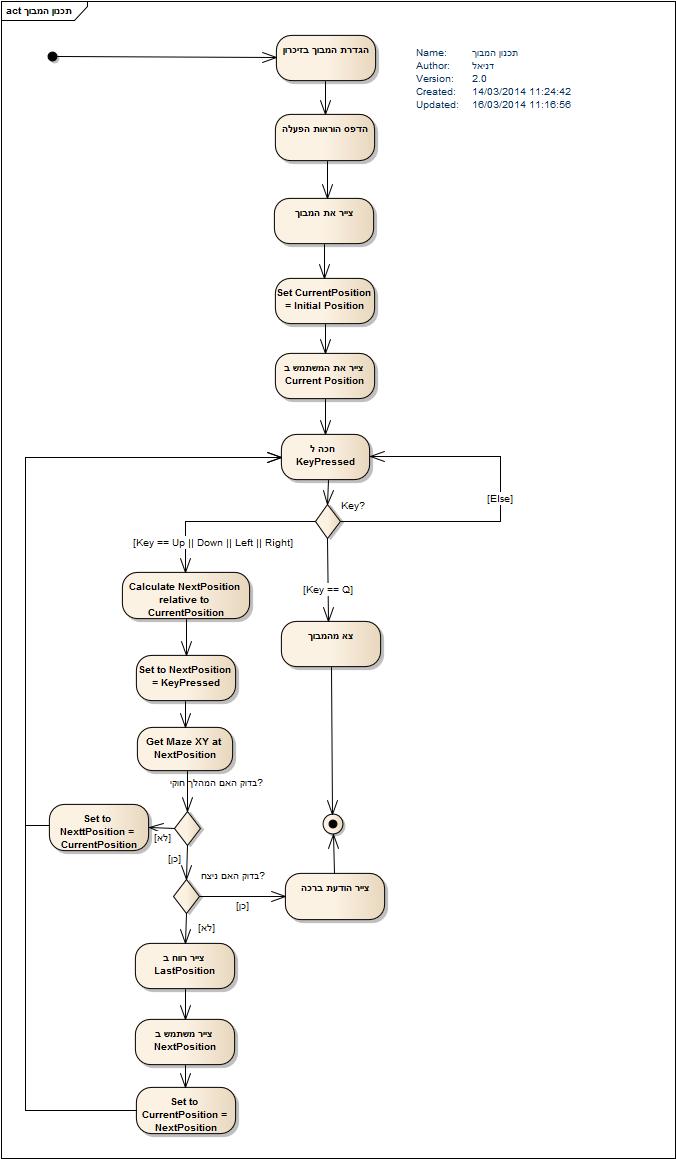
כאשר Emu8086 הינו אימולטור מלא של מחשב 8086, הכולל מערכת הפעלה תואמת ל-DOS.

## פונקציות שסופקו ע"י מנחה הפרויקט

הפונקציות שנמצאות בתוך קובץ ה-provided.asm סופקו ע"י המנחה מטעם בית הספר. מתוכם, השתמשתי בפונקציות **print\_str ,print\_char clear\_screen,**– כמצוין בהמשך.

# מבנה תוכנה

## תרשים זרימה ראשי (UML Activity diagram)



## עץ קריאות – מקרא

(ראו עץ הקריאות עצמו בעמוד הבא)

פונקצית עזר נתונה מראש ע"י המנחה

פונקציה שנכתבה על ידי

## עץ קריאות ראשי

## תיאור פונקציות עיקריות

### maze\_getxy

הפונקציה מחזירה את תו ה-ascii במיקום X,Y מתוך המערך אשר מייצג את המבוך.

### draw\_maze

הפונקציה מציירת את המבוך.

### draw\_char\_at\_position

הפונקציה מציירת אימוג'י במיקום X,Y על גבי המבוך.

### wait\_for\_key\_pressed\_and\_then\_convert\_it

הפונקציה מחכה לקבלת קלט מהמשתמש וממירה את הקלט למספר על פי תו ה-ascii או קוד הסריקה שלו, ומחזירה את המספר המומר בתור פלט. היה והמספר המומר הוא חמש, אשר מסמל מקש שאיננו נמצא בחוקי המשחק, אזי הפונקציה תקרא לעצמה מחדש עד לקבלת קלט תקין.

### Key\_to\_new\_position

הפונקציה מקבלת את המיקום הנוכחי של השחקן ואת הכיוון שאליו רוצה השחקן לזוז, בהתאם היא מציבה ב-[next x] וב-[next y] את המיקום שאליו רוצה לזוז השחקן.

### check\_legal\_move

הפונקציה מקבלת את המיקום שאליו השחקן רוצה לזוז ומחזירה האם מותר לשחקן לזוז למיקום זה.

### check\_win

הפונקציה מקבלת X,Y שאליו השחקן רוצה לזוז ובודקת האם זה מיקום ניצחון (כלומר מיקום יציאה מן המבוך), ומחזירה אמת או שקר בהתאם למיקום.

# תיאור בעיות ואתגרים במהלך הפרויקט והתמודדות עימן

## המיפוי של המבוך בזיכרון

* תיאור הבעיה:

בפרויקט יש צורך בהקצאת זיכרון אשר יכיל את המבוך (כלומר את הקירות והמעברים שבו). הנטייה הראשונית והאינטואיטיבית הייתה למפות את המבוך בזיכרון כמערך דו-מימדי, אך יישום זאת ב-Assembly הוא מאתגר יותר עקב תצורת הזיכרון הנתונה. מצד אחד מטעמי נוחות היה לי עניין להתייחס למערך דו-מימדי במהלך כתיבת התוכנית, ומצד שני על מנת לממש את הפרויקט באסמבלי היה עלי לשטח את הזיכרון. לכן, לא היה ברור לי כיצד ניתן להתייחס למקום במבוך ע"י קוארדינטות X ו-Y ועדיין לגשת לזיכרון במבנה שטוח.

בנוסף, על מנת להדפיס את המבוך כפלט על המסך היה נח להשתמש בפונקציה קיימת של הדפסת String אך פוקנציה זו מצפה ל-CR/LF בסוף כל שורה, דבר שחייב להתבטא בזיכרון אשר יכיל את מיפוי המבוך.

* התמודדות / פתרון:

בחרתי לבנות מערך חד-ממדי עם הוספת סיום שורה (CR /LF) בסוף הערך האחרון של כל שורה במבוך, כך יצרתי מצב בו מיפוי המבוך בזיכרון כולל בתוכו גם את סיום השורות. יתרון תצורה זו הוא שניתן בקלות להדפיס את המערך למסך, היות שלכל סוף שורה קיים CR/LF.

על מנת לגשת לקואורדינטת X,Y במערך, ולהתגבר על כך שהזיכרון הוא מערך חד-ממדי בלבד, מכפילים את Y במספר העמודות במבוך (Size Of X שכולל 2 תווים ל-CR/LF). מוסיפים את X, וניגשים לנקודה בזיכרון. היתרון ברור, במהלך הכתיבה ניתן להתייחס למבוך כדו-ממדי, על אף שהוא ממומש בתצורה חד-ממדית.

## יציאה מהמבוך

* תיאור הבעיה:

עבור כל תזוזה של השחקן יש לבדוק האם האימוג'י נמצא בנקודת יציאה מן המבוך. לשם כך יש צורך בפונקציה ייעודית אשר תפקידה לבדוק האם המשתמש ניצח וסיים את המשחק. בתחילה, לא היה ברור לי מה תהיה הבדיקה, כלומר כיצד התוכנית תוכל "לדעת" האם השחקן סיים את המשחק בהצלחה.

להלן מספר אפשרויות שנשקלו:

1. הגדרה מראש של קואורדינטות X, Y אשר מסמלות את נקודת היציאה מן המבוך, זאת בעזרת משתנה נפרד אשר יכיל את הנקודה הספציפית.
2. תו ascii מיוחד במבוך אשר יסמל את היציאה במיפוי המבוך.
3. בדיקה האם האימוג'י נמצא על גבול (כלומר קיר חיצוני) במבוך, הפתרון עובד בהנחה שכלל הגבולות החיצוניים של המבוך הם אכן קירות.

* התמודדות / פתרון:

לאחר מספר ניסיונות ודרכי חשיבה, הצלחתי למצוא את פתרון פשוט ויעיל. היציאה מסומנת על המפה על ידי מקום ריק בקצה המבוך, קרי תו ascii של רווח.

כך הפונקציה תבדוק קודם האם המהלך הינו חוקי, כלומר שאין קיר או הפרעה אחרת, ואם זה נכון, האם ה-NextPosition נמצא על גבול המבוך. אם ה-NextPosition נמצא על הגבול ועבר את בדיקת החוקיות, אז וודאי שהשחקן הגיע ליציאה.

היתרון הוא שפתרון זה מאפשר לתמוך במספר יציאות מהמבוך ע"י יצירת מפת מבוך שתכלול מספר רווחים על גבולות המבוך.

# הצעות להרחבה עתידית

## בניית מספר מבוכים (שלבים)

* הסבר: המשחק יכיל מספר מבוכים שונים, ברמת קושי עולה. כך המשחק יהפוך לממושך יותר והשחקן יוכל לפצח את דרכו במשחק עד לסיום המבוך האחרון.
* מטרה: להפוך את המשחק ליותר אטרקטיבי ומאתגר.
* הצעת יישום תמציתית: יצירת תאי זיכרון נוספים המכילים מבוכים שונים.

## בניית מבוך אקראי

* + הסבר: בניית המבוך בתוכנית באופן אקראי בכל ריצה של התוכנית.
  + מטרה: להפוך את המשחק לאטרקטיבי גם לאחר סיום המשחק. הגרסא של המבוך היא בסיסית יחסית, מכיוון שלאחר סיום המשחק השחקן יכיר את המשחק על בוריו, כיוון שהמבוך נוצר באופן קבוע ולא אקראי.
  + הצעת יישום קצרה: יש להשתמש בזמן הנתון בעת ריצת התוכנית על מנת לייצר קירות ומעברים באופן אקראי, כמוכן יש לייצר את המבוך ע"י רקורסיה על מנת לוודא כי אכן ישנה דרך בה ניתן לסיים את המשחק (כלומר יש יציאה שניתן להגיע אליה מנקודת ההתחלה).

## תזוזה באלכסון

* הסבר: השחקן יוכל לזוז גם באלכסון, ולא רק בתנועות ישרות.
* מטרה: להפוך את המשחק ליותר מעניין ומיוחד.
* הצעת יישום קצרה: שינוי ייצוג המבוך כך שיאפשר תנועות באלכסון, בחירת מקשים אשר ייצגו תנועה באלכסון, ויצירת הוראה לתזוזה באלכסון.

## מעבר מצד לצד

* הסבר: השחקן יוכל לעבור מקצה אחד של הלוח אל הקצה השני.
* מטרה: להפוך את המשחק ליותר מעניין ומיוחד.
* הצעת יישום קצרה: זהו פיצ'ר מורכב אשר תצורת המימוש הנוכחית מקשה על אופן יישומו, אמנם ישנם דרכים ליישם את הפיצ'ר, אך יש לתת על כך את הדעת ולא ניתן להסביר את היישום בקצרה.

## פתירת המבוך

* הסבר: בהתאם לבקשת השחקן ובמידה שהוא איננו מצליח לפתור את המבוך בכוחות עצמו, התכונית תוכל להדפיס למסך את הדרך שבה ניתן לפתור את המבוך הנתון.
* מטרה: שחקן שאיננו מצליח את המבוך, יוכל לראות את דרך הפתרון וכך ישכיל ולא ייצא מתוסכל מניסיונות חוזרים ונשנים אשר נדונו לכישלון.
* הצעת יישום קצרה: שימוש באלגוריתם backtracking שפותר את המבוך, והדפסת תוצאות האלגוריתם.

## זמן מוקצב לפתירת המבוך

* הסבר: בהינתן שיעבור זמן מסוים המשחק יגמר בתבוסה של השחקן.
* מטרה: יצירת עניין ותחרותיות נגד הזמן.
* הצעת יישום קצרה: שימוש ב interrupt שיחכה זמן רב ולאחר תום הזמן יציג הודעת תבוסה.

# תודות

תודה למנחי הפרויקט: **מר** **שאול שמולה, מר יאיר מירסקי ומר חגי לוי,** שבדקו את תיק הפרויקט זה, והעירו את הערותיהם הנבונות לגבי העריכה והתוכן.

# נספחים

## נספח קוד מקור

org 100h

name "TheMaze"

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

; the great maze!

;

; By: Daniel Wolkowicz

;

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

;

; Constants

;

space\_ascii EQU 20h

ascii\_CapQ EQU 51h

ascii\_SmallQ EQU 71h

ascii\_left EQU 4B00h

ascii\_right EQU 4D00h

ascii\_up EQU 4800h

ascii\_down EQU 5000h

quit EQU 0h

left\_arrow EQU 1h

right\_arrow EQU 2h

up\_arrow EQU 3h

down\_arrow EQU 4h

ignore EQU 5h

false EQU 0h

true EQU 1h

THE\_USER EQU 1h ; the emoji

SIZE\_OF\_X EQU 17h ; twenty one columns, 15h+0Dh,0Ah

SIZE\_OF\_Y EQU 0Ah ; ten rows

SIZE\_Xborder EQU 14h ; twenty one columns

MINIMUM\_Xborder EQU 0h ;column=0

SIZE\_Yborder EQU 0Ah

MINIMUM\_Yborder EQU 0h

;

; Main Function

;

call hide\_cursor\_and\_draw\_welcome\_message

call draw\_maze ; draw the maze

mov dl,[current\_x]

mov dh,[current\_y]

mov al,THE\_USER

call draw\_char\_at\_position ; draw the emoji at current position

main\_loop:

call wait\_for\_key\_pressed\_and\_then\_convert\_it ; wait for user

cmp al,quit ; 'q' for quit

je too\_bad\_player\_is\_a\_looser

call Key\_to\_new\_position ; set next position

call check\_legal\_mov ; check if new position is a legal one

cmp bl,false

je main\_loop

call check\_win ;check if new position is the exit

cmp bl,1

je draw\_congrats\_and\_exit

mov dl,[current\_x]

mov dh,[current\_y]

mov al,space\_ascii

call draw\_char\_at\_position ; draw space at current position

mov dl,[next\_x]

mov dh,[next\_y]

mov al,THE\_USER

call draw\_char\_at\_position ; draw emoji to next position

call set\_current\_position\_to\_next\_position

jmp main\_loop

draw\_congrats\_and\_exit:

call draw\_congratulations\_message

too\_bad\_player\_is\_a\_looser:

ret

;

; Memory variables

;

next\_x db 0h

next\_y db 0h

current\_x db 1h

current\_y db 1h

the\_maze db '---------------------',0Dh, 0Ah,

db '| | | |', 0Dh, 0Ah,

db '--- - --------- - - -', 0Dh, 0Ah,

db '| | | | | |', 0Dh, 0Ah,

db '- --------- - --- - -', 0Dh, 0Ah,

db '| | | | | | |', 0Dh, 0Ah,

db '--- - ----- - --- ---', 0Dh, 0Ah,

db '| | | | | |', 0Dh, 0Ah,

db '- ----- - - --- - - -', 0Dh, 0Ah,

db '| | | | ', 0Dh, 0Ah,

db '---------------------', 0Dh, 0Ah, 0

welcome\_message db ' \_\_ \_\_ \_\_ \_\_ ', 0Dh, 0Ah,

db ' \_ \_\_\_\_\_ / /\_\_\_\_\_\_\_\_ \_\_\_\_ \_\_\_ \_\_\_ / /\_\_\_\_\_ / /\_/ /\_ \_\_\_ ', 0Dh, 0Ah,

db '| | /| / / \_ \/ / \_\_\_/ \_\_ \/ \_\_ `\_\_ \/ \_ \ / \_\_/ \_\_ \ / \_\_/ \_\_ \/ \_ \', 0Dh, 0Ah,

db '| |/ |/ / \_\_/ / /\_\_/ /\_/ / / / / / / \_\_/ / /\_/ /\_/ / / /\_/ / / / \_\_/', 0Dh, 0Ah,

db '|\_\_/|\_\_/\\_\_\_/\_/\\_\_\_/\\_\_\_\_/\_/ /\_/ /\_/\\_\_\_/ \\_\_/\\_\_\_\_/ \\_\_/\_/ /\_/\\_\_\_/ ', 0Dh, 0Ah,

db ' ', 0Dh, 0Ah,

db ' \_\_ \_\_\_\_\_\_ ', 0Dh, 0Ah,

db ' \_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_ \_/ /\_ \_\_\_\_ \_\_\_ \_\_\_\_ \_\_\_\_\_ \_\_\_ / / / / ',0Dh, 0Ah,

db ' / \_\_ `/ \_\_\_/ \_ \/ \_\_ `/ \_\_/ / \_\_ `\_\_ \/ \_\_ `/\_ / / \_ \/ / / / ', 0Dh, 0Ah,

db ' / /\_/ / / / \_\_/ /\_/ / /\_ / / / / / / /\_/ / / /\_/ \_\_/\_/\_/\_/ ', 0Dh, 0Ah,

db ' \\_\_, /\_/ \\_\_\_/\\_\_,\_/\\_\_/ /\_/ /\_/ /\_/\\_\_,\_/ /\_\_\_/\\_\_\_(\_|\_|\_) ', 0Dh, 0Ah,

db ' /\_\_\_\_/ ', 0Dh, 0Ah,

db ' ', 0Dh, 0Ah,

db 'to continue please press any key ', 0Dh, 0Ah,

db 'if you wanna quit in the middle of the game just press Q ', 0Dh, 0Ah,

db 'to move use the arrows on the keyboard',0

congratulations\_message db ' \_\_ ', 0Dh, 0Ah,

db ' \_ \_\_\_\_\_\_ \_ \_\_/ / ', 0Dh, 0Ah,

db ' | | /| / / \_\_ \ | /| / / / ', 0Dh, 0Ah,

db ' | |/ |/ / /\_/ / |/ |/ /\_/ ', 0Dh, 0Ah,

db ' |\_\_/|\_\_/\\_\_\_\_/|\_\_/|\_\_(\_) ', 0Dh, 0Ah,

db ' ', 0Dh, 0Ah,

db ' \_ \_ \_\_ ', 0Dh, 0Ah,

db ' (\_)\_\_\_\_\_\_\_ \_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_(\_) \_\_\_\_\_ / / \_\_ \_\_\_\_\_\_ \_\_ \_\_', 0Dh, 0Ah,

db ' / / \_\_ \_\_ \/ \_\_ \/ \_\_\_/ \_ \/ \_\_\_/ \_\_\_/ / | / / \_ \/ / / / / / \_\_ \/ / / /', 0Dh, 0Ah,

db ' / / / / / / / /\_/ / / / \_\_(\_\_ |\_\_ ) /| |/ / \_\_/\_/ / /\_/ / /\_/ / /\_/ / ', 0Dh, 0Ah,

db ' /\_/\_/ /\_/ /\_/ .\_\_\_/\_/ \\_\_\_/\_\_\_\_/\_\_\_\_/\_/ |\_\_\_/\\_\_\_(\_) \\_\_ /\\_\_\_\_/\\_\_\_\_/ ', 0Dh, 0Ah,

db ' /\_/ /\_\_\_\_/ ', 0Dh, 0Ah,

db ' \_\_\_\_\_ \_ \_\_ \_\_ \_\_ \_\_ ', 0Dh, 0Ah,

db ' / \_\_(\_)\_\_\_ (\_)\_\_\_\_/ /\_ \_\_\_ \_\_\_\_/ / / /\_/ /\_ \_\_\_ ', 0Dh, 0Ah,

db ' / /\_/ / \_\_ \/ / \_\_\_/ \_\_ \/ \_ \/ \_\_ / / \_\_/ \_\_ \/ \_ \ ', 0Dh, 0Ah,

db ' / \_\_/ / / / / (\_\_ ) / / / \_\_/ /\_/ / / /\_/ / / / \_\_/ ', 0Dh, 0Ah,

db ' /\_/ /\_/\_/ /\_/\_/\_\_\_\_/\_/ /\_/\\_\_\_/\\_\_\_\_/ \\_\_/\_/ /\_/\\_\_\_/ ', 0Dh, 0Ah,

db ' \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_ ', 0Dh, 0Ah,

db ' / \_\_ / \_\_ / \_\_ \_\_ \/ \_ \ ', 0Dh, 0Ah,

db ' / /\_/ / /\_/ / / / / / / \_\_/ ', 0Dh, 0Ah,

db ' \\_\_ /\\_\_\_\_/\_/ /\_/ /\_/\\_\_\_/ ', 0Dh, 0Ah,

db ' /\_\_\_\_/ ', 0Dh, 0Ah,

db ' ', 0Dh, 0Ah,

db 'to exit the game please press any key',0

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

; draw\_welcome\_message() - welcoming the player

;

; uses print\_str()

; wait\_for\_key\_pressed()

; clear\_screen()

;

; Parameters:

; none

;

; Returns:

; none

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

proc draw\_welcome\_message

push ax

lea ax,[welcome\_message] ;set the address of [welcome\_message] in ax

call print\_str ;prints [welcome\_message]

call wait\_for\_key\_pressed ; receives a key

call clear\_screen ; clears the console

pop ax

ret

endp

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

; maze\_getxy() - Load maze character from maze array

;

; Parameters:

; dl - x

; dh - y

;

; Returns:

; al - the char in the place of [current\_x] and [current\_y]

; ah will stay as before

;

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

proc maze\_getxy

push bx

;set ax to the start of the current row

mov al,dh

mov bl,SIZE\_OF\_X

mul bl

;find column

add al,dl

;mov the char to ax

lea bx, [the\_maze]

add bx, ax

mov al, [bx]

pop bx

ret

endp

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

;draw\_maze() - Draws the maze

;

; (uses PRINT\_STR)

;

; Parameters:

; none

;

; Returns:

; none

;

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

proc draw\_maze

push ax

lea ax,[the\_maze] ;ax contains now the address of the maze

call PRINT\_STR

pop ax

ret

endp

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

;draw\_char\_at\_position() - draws char at position xy

;

;uses print\_char()

;uses set\_cursor()

;

;Parameters:

; al - ascii char to write

; dl - x

; dh - y

;

; Returns:

; none

;

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

proc draw\_char\_at\_position

call set\_cursor

call print\_char

ret

endp

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

;set\_cursor() - move the cursor to xy

;

;Parameters:

; dh - y

; dl - x

;

; Returns:

; none

;

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

proc set\_cursor

push ax

mov ah,2

int 10h

pop ax

ret

endp

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

;wait\_for\_key\_pressed() - waiting for key

;

;Parameters:

; none

;

;Returns:

; ah - BIOS scan code

; al - ascii char

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

proc wait\_for\_key\_pressed

mov ax,0

int 16h

ret

endp

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

;convert\_key\_pressed() - converts the key that was pressed to a number

;

;Parameters:

; ah - BIOS scan code

; al - ascii char

;

;Returns:

; al - switch:left=1, right=2, up=3, down=4, Q=0, q=0, another key (ignore)=5

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

proc convert\_key\_pressed

cmp al,ascii\_CapQ ;does the player wants to quit?

jne do\_not\_quit ;maybe

mov al,quit ;yes he wants to quit

jmp convert\_done

do\_not\_quit:

cmp al,ascii\_SmallQ ;does the player wants to quit?

jne do\_not\_quit\_its\_an\_arrow

mov al,quit ;yes he wants to quit

jmp convert\_done

do\_not\_quit\_its\_an\_arrow:

cmp ax,ascii\_left ;was the left arrow pressed?

jne its\_not\_left ;no its not

mov al,left\_arrow ;yes it does

jmp convert\_done

its\_not\_left:

cmp ax,ascii\_right ;was the right arrow pressed?

jne its\_not\_right ;no its not

mov al,right\_arrow ;yes it does

jmp convert\_done

its\_not\_right:

cmp ax,ascii\_up ;was the up arrow pressed?

jne its\_not\_up ;no its not

mov al,up\_arrow ;yes it does

jmp convert\_done

its\_not\_up:

cmp ax,ascii\_down ;was the down arrow pressed?

jne another\_key ;no its not

mov al,down\_arrow ;yes it does

jmp convert\_done

another\_key:

mov al,5

convert\_done:

ret

endp

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

;wait\_for\_key\_pressed\_and\_then\_convert\_it()- waiting for key and converts it to num if the

;key that pressed equals to ignore(=5), the function will wait for another key to be pressed

;

; uses wait\_for\_key\_pressed()

; convert\_key\_pressed()

;

;Parameters:

; none

;

;

;Returns:

; al - switch:left=1, right=2, up=3, down=4, Q=0, q=0, another key (ignore)=5

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

proc wait\_for\_key\_pressed\_and\_then\_convert\_it

the\_key\_that\_was\_pressed\_isnt\_relevant:

call wait\_for\_key\_pressed ;wait for a key to be pressed

call convert\_key\_pressed ; convert the key

cmp al,ignore ; if the key isnt relevant wait again for a key else do nothing

je the\_key\_that\_was\_pressed\_isnt\_relevant

ret

endp

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

;Key\_to\_new\_position()-receives the current position with the direction to move

; returns the next position

;

;Parameters:

; [current\_x] - x

; [current\_y] - y

; al - switch:left=1, right=2, up=3, down=4

;Returns:

; [next\_x] - x of new position

; [next\_y] - y of new position

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

proc Key\_to\_new\_position

push dx

mov dl,[current\_x]

mov dh,[current\_y]

cmp al,left\_arrow ;does the player wants to move left

jne left\_arrow\_wasnt\_pressed ;if not continue else jump

dec dl ;move the x to the left (x-1)

jmp new\_position\_done ;arrange the registers and finish

left\_arrow\_wasnt\_pressed:

cmp al,right\_arrow ;does the player wants to move right

jne right\_arrow\_wasnt\_pressed ;if not jump else continue

inc dl ;move the x to the right (x+1)

jmp new\_position\_done ;arrange the registers and finish

right\_arrow\_wasnt\_pressed:

cmp al,up\_arrow ;does the player wants to move up

jne up\_arrow\_wasnt\_pressed ;if not jump else continue

dec dh ;move the y up (y-1)

jmp new\_position\_done ;arrange the registers and finish

up\_arrow\_wasnt\_pressed: ;if you arrived here the player wants to move down

inc dh ;move the y down (y+1)

jmp new\_position\_done

new\_position\_done:

mov [next\_x],dl ;set next x

mov [next\_y],dh ;set next y

pop dx

ret

endp

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

;check\_legal\_move() -check if the user is allowed to move to xy (there is a space)

;

;Parameters:

; [next\_x] - x of the position to check

; [next\_y] - y of the position to check

;

;returns:

; bl- is it a leagal movement? switch: 0=false 1=true

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

proc check\_legal\_move

push ax

push dx

mov dl,[next\_x]

mov dh,[next\_y]

call maze\_getxy ;what's the ascii char at xy?

cmp al,space\_ascii ;the movement is legal?

jne ilegal\_movement

mov bl,true ;legal\_movement=true

jmp finish\_check\_legal\_move

ilegal\_movement:

mov bl,false ;ilegal\_movement=false

mov ah,[current\_x] ;reset [next\_x] to [current\_x]

mov [next\_x],ah

mov ah,[current\_y] ;reset [next\_y] to [current\_y]

mov [next\_y],ah

finish\_check\_legal\_move:

pop dx

pop ax

ret

endp

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

;check\_win()- checks if x, y is the exit

;

;Parameters:

; [next\_x] - x

; [next\_y] - y

;

;Returns:

; bl - is it the exit? switch: 0=false 1=true

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

proc check\_win

cmp [next\_x],SIZE\_Xborder ;is x of the exit?

je its\_an\_exit

cmp [next\_x],MINIMUM\_Xborder ;is x of the exit?

je its\_an\_exit

cmp [next\_y],SIZE\_Yborder ;is y of the exit?

je its\_an\_exit

cmp [next\_y],MINIMUM\_Yborder ;is y of the exit?

jne the\_player\_isnt\_moving\_to\_the\_exit

its\_an\_exit:

mov bl,1 ;it is the exit, the player has won

jmp check\_win\_done

the\_player\_isnt\_moving\_to\_the\_exit:

mov bl,0 ;the player didnt win yet

check\_win\_done:

ret

endp

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

;hide\_cursor()-hide blinking text cursor

;

;Parameters:

; none

;

;returns:

; none

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

proc hide\_cursor

push ax

push cx

mov ch, 32

mov ah, 1

int 10h

pop cx

pop ax

ret

endp

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

;draw\_congratulations\_message() - congrats the player

;

; uses print\_str()

; wait\_for\_key\_pressed()

; clear\_screen()

;

; Parameters:

; none

;

; Returns:

; none

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

proc draw\_congratulations\_message

push ax

call clear\_screen ; clears the console

lea ax,[congratulations\_message] ;set the address of [welcome\_message] in ax

call print\_str ;prints [welcome\_message]

call wait\_for\_key\_pressed ; receives a key

pop ax

ret

endp

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

;set\_current\_position\_to\_next\_position() - sets [current\_x] to [next\_x] & [current\_y] to [next\_y]

;

; Parameters:

; x - [current\_x]

; y - [current\_y]

; x - [next\_x]

; y - [next\_y]

;

; Returns:

; the changed [current\_x]&[current\_y] & [next\_x],[next\_y]

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

proc set\_current\_position\_to\_next\_position

push ax

mov al,[next\_x] ;set [current\_x]

mov [current\_x],al

mov al,[next\_y] ;set [current\_y]

mov [current\_y],al

pop ax

ret

endp

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

;hide\_cursor\_and\_draw\_welcome\_message() -hide blinking text cursor and welcoming the player

;

; uses hide\_cursor()

; draw\_welcome\_message()

;

; Parameters:

; none

;

; Returns:

; none

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

proc hide\_cursor\_and\_draw\_welcome\_message

call hide\_cursor

call draw\_welcome\_message

ret

endp

;

; Functions provided to us

;

include "..\MySource\provided.asm"