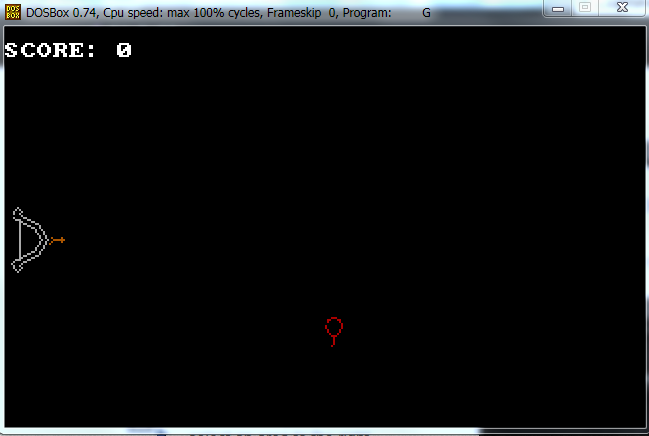
פרוייקט באסמבלי – Balloon Wars



**שם המגיש:** פימה פנחס מוצמכר

**ת.ז:** 325118693

**תאריך הגשה:** 3.6.2018

**מורה מלווה:** איזבלה טבלין

**סביבת עבודה:** notepad++,dosBox

**כיתה:** י'9

**בית ספר:** תיכון דה שליט רחובות

**שם הקובץ:** "BalloonWars.asm"

**קבצים נלווים:** "game.bmp" , "lose.bmp"

**תוכן עניינים**

מבוא......................................................3-4

אופן ההפעלה..........................................5-6

הסבר ההפעלה........................................7-9

דיאגרמת התכנית.....................................10

קוד התכנית ........................................11-43

סיכום אישי..............................................44

**מבוא**

**שם הפרוייקט –** מלחמות הבלונים

**שם הקובץ –** balloon.asm

**קבצים נלוים –** game.bmp, lose.bmp

**סביבת העבודה –** turbo assembler

**סביבת הפיתוח –** notepad++

**סביבת הרצה** **–** DosBox

**רקע למשחק**

המשחק אותו בחרתי להכין הוא למעשה משחק בו יורים בעזרת חץ וקשת

אל בלון הנע מעלה במסך. אם פוגעים, המשחק ממשיך לרוץ ואם מפספסים

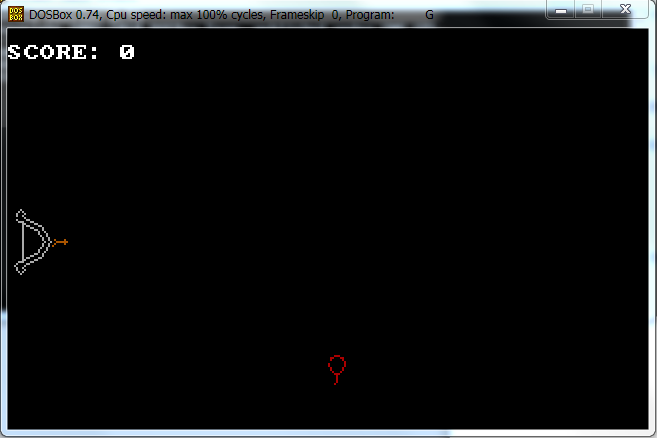
מפסידים. למשחק ישנה גרסאות רבות ברחבי האינטרנט , בעבר הוא היה

פופולרי מאד.

**מדוע בחרתי דווקא במשחק זה?**

כשהתחלנו ללמוד אסמבלי בכיתה י' ידעתי שאנסה לדמות משחק זה כי שחקתי

בו כשהייתי קטן יותר ומאד נהנתי מהחוויה.

**גרסה מהאינטרנט הגרסה שלי**

**אופן הפעלה**

מריצים את התכנית בעזרת התכנה DOSBOX.

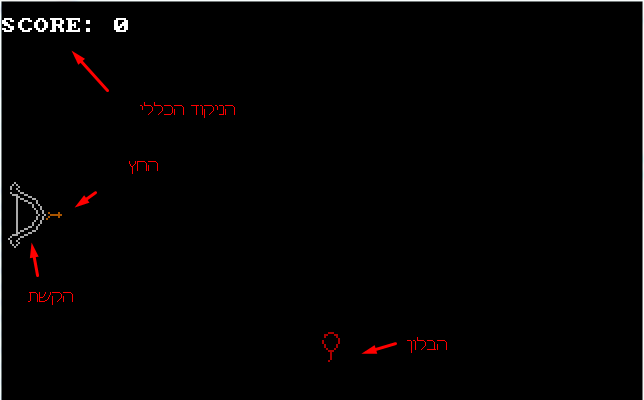
בתחילת ההרצה מגיעים למסך הפתיחה של המשחק:



יש ללחוץ על מקש כלשהו במקלדת בכדי להתחיל לשחק.

בהתחלה נראה במסך את הבלון, הקשת והחץ עומדים, לאחר שנייה

הבלון יתחיל לנוע במסך, בכדי לירות בבלון יש ללחוץ על המקש

השמאלי בעכבר. חשוב

לחשב את זמן השיגור

המומלץ בכדי לפגוע בבלון.

אם פוגעים בבלון, מקבלים

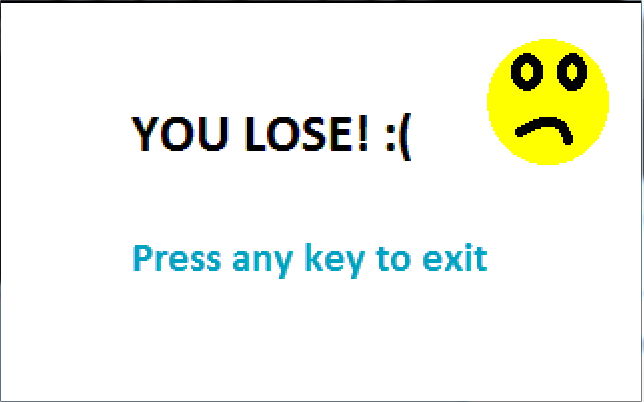
נקודה והבלון חוזר למקום

רנדומאלי.

את כמות הנקודות הכוללת ניתן לראות בצד השמאלי של המסך.

בעת הפסד במשחק, כאשר מפספסים את הבלון ישנו מסך סגירה

לתכנית:



**תיאור והסבר פתרון**

**פרוצדורות הקשורות לגרפיקה:**

* פרוצדורה המציירת את הבלון,החץ והקשת.
* פרוצדורה המציירת את הניקוד.

**פרוצדורות נלוות לעדכון ערכים:**

* פרוצדורה לשיגור החץ.
* פרוצדורה לשיגור הבלון.
* פרוצדורת RANDOM

**פרוצדורות הבודקות תנאים:**

* פרוצדורת בדיקת פגיעה.
* פרוצדורת RESET .

**פרוצדורות להדפסת תמונות:**

* פרוצדורה להדפסת תמונת התחלה.
* פרוצדורה להדפסת תמונת סיום.

**רשימת פעולות**

1. Print

פרוצדורה המדפיסה את החץ, הקשת והבלון.

1. ShowPictureStart

פרוצדורה המראה את תמונת ההתחלה של המשחק.

1. ShowPictureLose

פרוצדורה המראה את תמונת הסוף של המשחק.

1. EraseArrow

פרוצדורה המוחקת את הפיקלסים של החץ, צובעת אותם בשחור.

1. EraseBalloon

פרוצדורה המוחקת את הפיקסלים של הבלון, צובעת אותם בשחור.

1. ValueUpdater

הפרוצדורה המרכזית של התכנה, בודקת אם הבלון והחץ עפים,

מעדכנת את ערכיהם בהתאם ומדפיסה מחדש.

1. Delay

פרוצדורה של דיליי בתכנית של בערך 0.25 שניות.

1. BigDelay

פרוצדורה של דיליי ארוך בתכנית של בערך שנייה.

1. LaunchArrow

פרוצדורה המשגרת את הבלון כאשר נלחץ המקש בעכבר.

1. CheckHit

פרוצדורה הבודקת האם החץ פוגע בבלון.

1. RandomNumber

פרוצדורה המחזירה מספר ראנדומלי במשתנה rnd

1. Reset

פרוצדורה המחזירה את עריכהם של הבלון והחץ למקורם (השמור

בdata SEGMENT) ומוסיפה ערך רנדומלי לערכי הX של הבלון.

1. DisplayScore

פרוצדורה המדפיסה את הניקוד של השחקן במשחק.

**הערה!**

בעבודתי השתמשתי בgraphics mode ולכן ישנם ערכים רבים

שמורים ב.DS ערכים אלה הם מיקומי הפיקסלים של החץ, הקשת

והבלון.

**דיאגרמת הפעולות של הפרוייקט**

**C:\Users\u110113\Downloads\diagrama.png**

**קוד התכנית**

**IDEAL**

**MODEL small**

**STACK 100h**

**DATASEG**

**;Welcome to The Balloon Game!**

**;All graphic design of the balloon,bow,arrow are stored as pixel maps in the data segment.**

**;M stands for memory data, I saved same values for having stored static values at all times in procedures.**

**mbowx dw 2 dup(3d),6 dup(4d),4 dup(5d),4 dup(6d),25 dup(7d),6 dup(8d),4 dup(9d),4 dup(10d),4 dup(11d),4 dup(12d),4 dup(13d),4 dup(14d),6 dup(15d),4 dup(16d),8 dup(17d),3 dup(18d),4 dup(19d),4 dup(20d),21d**

**mbowy dw 94d,118d,92d,93d,95d,117d,119d,120d,91d,96d,116d,121d,90d,96d,116d,122d,91d,93d,96d,97d,98d,99d,100d,101d,102d,103d,104d,105d,106d,107d,108d,109d,110d,111d,112d,113d,114d,115d,116d,119d,121d,92d,94d,97d,115d,118d,120d,95d,98d,114d,117d,95d,98d,114d,117d,96d,99d,113d,116d,96d,99d,113d,116d,97d,100d,112d,115d,97d,100d,112d,115d,98d,101d,102d,110d,111d,114d,98d,103d,109d,114d,99d,100d,104d,105d,107d,108d,112d,113d,101d,106d,111d,102d,103d,109d,110d,104d,105d,107d,108d,106d**

**mbowpixels dw 100d**

**mballoonx dw 3 dup(160d),4 dup(161d),2 dup(162d),3 dup(163d),6 dup(164d),2 dup(165d),2 dup(166d),4 dup(167d),3 dup(168d)**

**mballoony dw 188d,189d,190d,186d,187d,191d,192d,186d,193d,185d,194d,199d,185d,194d,195d,196d,197d,198d,185d,194d,186d,193d,186d,187d,191d,192d,188d,189d,190d**

**mballoonpixels dw 28d**

**marrowx dw 2 dup(22d),2 dup(23d),24d,25d,26d,27d,3 dup(28d),29d**

**marrowy dw 104d,108d,105d,107d,106d,106d,106d,106d,105d,106d,107d,106d**

**bowx dw 2 dup(3d),6 dup(4d),4 dup(5d),4 dup(6d),25 dup(7d),6 dup(8d),4 dup(9d),4 dup(10d),4 dup(11d),4 dup(12d),4 dup(13d),4 dup(14d),6 dup(15d),4 dup(16d),8 dup(17d),3 dup(18d),4 dup(19d),4 dup(20d),21d**

**bowy dw 94d,118d,92d,93d,95d,117d,119d,120d,91d,96d,116d,121d,90d,96d,116d,122d,91d,93d,96d,97d,98d,99d,100d,101d,102d,103d,104d,105d,106d,107d,108d,109d,110d,111d,112d,113d,114d,115d,116d,119d,121d,92d,94d,97d,115d,118d,120d,95d,98d,114d,117d,95d,98d,114d,117d,96d,99d,113d,116d,96d,99d,113d,116d,97d,100d,112d,115d,97d,100d,112d,115d,98d,101d,102d,110d,111d,114d,98d,103d,109d,114d,99d,100d,104d,105d,107d,108d,112d,113d,101d,106d,111d,102d,103d,109d,110d,104d,105d,107d,108d,106d**

**bowpixels dw 100d**

**balloonx dw 3 dup(160d),4 dup(161d),2 dup(162d),3 dup(163d),6 dup(164d),2 dup(165d),2 dup(166d),4 dup(167d),3 dup(168d)**

**balloony dw 188d,189d,190d,186d,187d,191d,192d,186d,193d,185d,194d,199d,185d,194d,195d,196d,197d,198d,185d,194d,186d,193d,186d,187d,191d,192d,188d,189d,190d**

**balloonpixels dw 28d**

**arrowx dw 2 dup(22d),2 dup(23d),24d,25d,26d,27d,3 dup(28d),29d**

**arrowy dw 104d,108d,105d,107d,106d,106d,106d,106d,105d,106d,107d,106d**

**arrowpixels dw 11d**

**waskeypressed dw 00h**

**balloonflying dw 00h**

**hit dw 00h**

**rnd dw ?**

**score DB 10, 13, 'SCORE: ','$'**

**count dw 0**

**;================================**

**filename1 db 'game.bmp',0**

**filename2 db 'lose.bmp',0**

**filehandle dw ?**

**Header db 54 dup (0)**

**Palette db 256\*4 dup (0)**

**ScrLine db 320 dup (0)**

**ErrorMsg db 'Error', 13, 10,'$'**

**;================================**

**CODESEG**

**proc OpenFile1**

**; Open file**

**mov ah, 3Dh**

**xor al, al**

**mov dx, offset filename1**

**int 21h**

**jc Openerror1**

**mov [filehandle], ax**

**ret**

**Openerror1: ;jkjhkjhkjh**

**mov dx, offset ErrorMsg**

**mov ah, 9h**

**int 21h**

**ret**

**endp OpenFile1**

**;========================================**

**proc OpenFile2**

**; Open file**

**mov ah, 3Dh**

**xor al, al**

**mov dx, offset filename2**

**int 21h**

**jc Openerror2**

**mov [filehandle], ax**

**ret**

**Openerror2:**

**mov dx, offset ErrorMsg**

**mov ah, 9h**

**int 21h**

**ret**

**endp OpenFile2**

**;========================================**

**proc ReadHeader**

**; Read BMP file header, 54 bytes**

**mov ah,3fh**

**mov bx, [filehandle]**

**mov cx,54**

**mov dx,offset Header**

**int 21h**

**ret**

**endp ReadHeader**

**proc ReadPalette**

**; Read BMP file color palette, 256 colors \* 4 bytes (400h)**

**mov ah,3fh**

**mov cx,400h**

**mov dx,offset Palette**

**int 21h**

**ret**

**endp ReadPalette**

**;========================================**

**proc CopyPal**

**; Copy the colors palette to the video memory**

**; The number of the first color should be sent to port 3C8h**

**; The palette is sent to port 3C9h**

**mov si,offset Palette**

**mov cx,256**

**mov dx,3C8h**

**mov al,0**

**; Copy starting color to port 3C8h**

**out dx,al**

**; Copy palette itself to port 3C9h**

**inc dx**

**PalLoop:**

**; Note: Colors in a BMP file are saved as BGR values rather than RGB.**

**mov al,[si+2] ; Get red value.**

**shr al,2 ; Max. is 255, but video palette maximal**

**; value is 63. Therefore dividing by 4.**

**out dx,al ; Send it.**

**mov al,[si+1] ; Get green value.**

**shr al,2**

**out dx,al ; Send it.**

**mov al,[si] ; Get blue value.**

**shr al,2**

**out dx,al ; Send it.**

**add si,4 ; Point to next color.**

**; (There is a null chr. after every color.)**

**loop PalLoop**

**ret**

**endp CopyPal**

**;========================================**

**proc CopyBitmap**

**; BMP graphics are saved upside-down.**

**; Read the graphic line by line (200 lines in VGA format),**

**; displaying the lines from bottom to top.**

**mov ax, 0A000h**

**mov es, ax**

**mov cx,200**

**PrintBMPLoop:**

**push cx**

**; di = cx\*320, point to the correct screen line**

**mov di,cx**

**shl cx,6**

**shl di,8**

**add di,cx**

**; Read one line**

**mov ah,3fh**

**mov cx,320**

**mov dx,offset ScrLine**

**int 21h**

**; Copy one line into video memory**

**cld**

**; Clear direction flag, for movsb**

**mov cx,320**

**mov si,offset ScrLine**

**rep movsb**

**; Copy line to the screen**

**;rep movsb is same as the following code:**

**;mov es:di, ds:si**

**;inc si**

**;inc di**

**;dec cx**

**;loop until cx=0**

**pop cx**

**loop PrintBMPLoop**

**ret**

**endp CopyBitmap**

**;========================================**

**proc ShowPictureStart**

**push ax**

**; Graphic mode**

**mov ah, 0 ; set display mode function.**

**mov al, 13h ; mode 13h = 320x200 pixels, 256 colors.**

**int 10h ; set it!**

**; Process BMP file**

**call OpenFile1**

**call ReadHeader**

**call ReadPalette**

**call CopyPal**

**call CopyBitmap**

**mov ah,1**

**int 21h**

**pop ax**

**ret**

**endp ShowPictureStart**

**;========================================**

**proc ShowPictureLose**

**push ax**

**; Graphic mode**

**mov ah, 0 ; set display mode function.**

**mov al, 13h ; mode 13h = 320x200 pixels, 256 colors.**

**int 10h ; set it!**

**; Process BMP file**

**call OpenFile2**

**call ReadHeader**

**call ReadPalette**

**call CopyPal**

**call CopyBitmap**

**mov ah,1**

**int 21h**

**pop ax**

**ret**

**endp ShowPictureLose**

**;========================================**

**proc Print**

**push ax**

**push bx**

**push cx**

**push dx**

**mov cx,[bowpixels] ;Inserts number of bow pixels in order to print them.**

**Printbow:**

**mov bx,cx**

**push cx**

**shl bx,1 ; all pixels are stored as words so the gap between each pixel is 2 bytes.**

**mov cx, [bowx+bx] ; column**

**mov dx, [bowy+bx] ; row**

**mov al, 07h ; gray**

**mov ah, 0ch ; put pixel**

**int 10h**

**pop cx**

**loop Printbow ;loop for printing the bow**

**xor ax,ax**

**xor bx,bx**

**xor cx,cx**

**mov cx,[arrowpixels] ;Inserts number of arrow pixels in order to print them.**

**Printarrow:**

**mov bx,cx**

**push cx**

**shl bx,1 ; all pixels are stored as words so the gap between each pixel is 2 bytes.**

**mov cx, [arrowx+bx] ; column**

**mov dx, [arrowy+bx] ; row**

**mov al, 06h**

**mov ah, 0ch ; put pixel**

**int 10h**

**pop cx**

**loop Printarrow ;loop for printing the arrow**

**xor ax,ax**

**xor bx,bx**

**xor cx,cx**

**mov cx,[balloonpixels]**

**Printballoon:**

**mov bx,cx**

**push cx**

**shl bx,1**

**mov cx, [balloonx+bx] ; column**

**mov dx, [balloony+bx] ; row**

**mov al, 04h**

**mov ah, 0ch ; put pixel**

**int 10h**

**pop cx**

**loop Printballoon**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp Print**

**;========================================**

**proc EraseArrow**

**push ax**

**push bx**

**push cx**

**push dx**

**mov cx,[arrowpixels]**

**Erasearrowloop:**

**mov bx,cx**

**push cx**

**shl bx,1**

**mov cx, [arrowx+bx] ; column**

**mov dx, [arrowy+bx] ; row**

**mov al, 00h**

**mov ah, 0ch ; put pixel**

**int 10h**

**pop cx**

**loop Erasearrowloop**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp EraseArrow**

**;========================================**

**proc EraseBalloon**

**push ax**

**push bx**

**push cx**

**push dx**

**mov cx,[balloonpixels]**

**Eraseballoonloop:**

**mov bx,cx**

**push cx**

**shl bx,1**

**mov cx, [balloonx+bx] ; column**

**mov dx, [balloony+bx] ; row**

**mov al, 00h**

**mov ah, 0ch ; put pixel**

**int 10h**

**pop cx**

**loop Eraseballoonloop**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp EraseBalloon**

**;========================================**

**proc ValueUpdater**

**push ax**

**push cx**

**push dx**

**push bx**

**Checkifballoonfly:**

**cmp [balloonflying],00h**

**je Returntocode**

**call EraseBalloon**

**mov cx,[balloonpixels]**

**Decreaseballoony:**

**mov bx,cx**

**shl bx,1**

**sub [balloony+bx],2d**

**loop Decreaseballoony**

**call Print**

**xor ax,ax**

**xor bx,bx**

**xor cx,cx**

**xor dx,dx**

**Checkifkeypressed:**

**cmp [waskeypressed],00h**

**je Returntocode**

**call EraseArrow**

**mov cx,[arrowpixels]**

**Increasearrowx:**

**mov bx,cx**

**shl bx,1**

**add [arrowx+bx],6d**

**loop Increasearrowx**

**call Print**

**Returntocode:**

**pop bx**

**pop dx**

**pop cx**

**pop ax**

**ret**

**endp ValueUpdater**

**;========================================**

**proc Delay**

**push ax**

**push bx**

**push dx**

**push cx**

**mov di, 1**

**mov ah, 0**

**int 1Ah**

**mov bx,dx**

**DelayLoop:**

**mov ah, 0**

**int 1Ah**

**sub dx, bx**

**cmp di, dx**

**ja DelayLoop**

**pop cx**

**pop dx**

**pop bx**

**pop ax**

**ret**

**endp Delay**

**;========================================**

**proc LaunchBalloon**

**call BigDelay**

**mov [balloonflying],01h**

**ret**

**endp LaunchBalloon**

**;========================================**

**proc LaunchArrow**

**push ax**

**push bx**

**push cx**

**push dx**

**mov ax,0003h**

**int 33h**

**cmp bx,1h**

**jne Nokeywaspressed**

**mov [waskeypressed],01h**

**Nokeywaspressed:**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp LaunchArrow**

**;;;;;;;;;;;;;;;;;;**

**proc BigDelay**

**push ax**

**push bx**

**push dx**

**push cx**

**mov di, 30**

**mov ah, 0**

**int 1Ah**

**mov bx,dx**

**DelayLoop1:**

**mov ah, 0**

**int 1Ah**

**sub dx, bx**

**cmp di, dx**

**ja DelayLoop1**

**pop cx**

**pop dx**

**pop bx**

**pop ax**

**ret**

**endp BigDelay**

**;========================================**

**proc CheckHit**

**push ax**

**push bx**

**push cx**

**push dx**

**mov ah,0Dh**

**mov cx,[arrowx+22d]**

**mov dx,[arrowy+22d]**

**int 10h**

**cmp al,04h**

**jne Nohit1**

**mov [hit],1h**

**jmp Nohitatall**

**Nohit1:**

**mov ah,0Dh**

**mov cx,[arrowx+20d]**

**mov dx,[arrowy+20d]**

**int 10h**

**cmp al,04h**

**jne nohit2**

**mov [hit],1h**

**jmp Nohitatall**

**Nohit2:**

**mov ah,0Dh**

**mov cx,[arrowx+16d]**

**mov dx,[arrowy+16d]**

**int 10h**

**cmp al,04h**

**jne nohitatall**

**mov [hit],1h**

**Nohitatall:**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp CheckHit**

**;========================================**

**proc Reset**

**push ax**

**push bx**

**push cx**

**push dx**

**mov al, 13h ; mode 13h = 320x200 pixels, 256 colors.**

**int 10h ; set it!**

**call EraseArrow**

**call EraseBalloon**

**call RandomNumber**

**mov cx,[balloonpixels]**

**Resetingballoon:**

**mov bx,cx**

**shl bx,1d**

**mov ax,[mballoonx+bx]**

**add ax,[rnd]**

**mov [balloonx+bx],ax**

**mov ax,[mballoony+bx]**

**mov [balloony+bx],ax**

**loop Resetingballoon**

**mov cx,[arrowpixels]**

**Resetingarrow:**

**mov bx,cx**

**shl bx,1d**

**mov ax,[marrowx+bx]**

**mov [arrowx+bx],ax**

**mov ax,[marrowy+bx]**

**mov [arrowy+bx],ax**

**loop Resetingarrow**

**call Print**

**mov [hit],0h**

**mov [waskeypressed],0h**

**mov [balloonflying],0h**

**inc [count]**

**call DisplayScore**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**ret**

**endp Reset**

**;=========================**

**proc DisplayScore ;Beginning of procedure**

**push ax**

**push bx**

**push cx**

**push dx**

**MOV AH, 09H**

**LEA DX, [score]**

**INT 21H ;Calls MS DOS to display message**

**mov ax,[count]**

**MOV BX, 10 ;Initializes divisor**

**MOV DX, 0000H ;Clears DX**

**MOV CX, 0000H ;Clears CX**

**;Splitting process starts here**

**Dloop1: MOV DX, 0000H ;Clears DX during jump**

**div BX ;Divides AX by BX**

**PUSH DX ;Pushes DX(remainder) to stack**

**INC CX ;Increments counter to track the number of digits**

**CMP AX, 0 ;Checks if there is still something in AX to divide**

**JNE Dloop1 ;Jumps if AX is not zero**

**Dloop2: POP DX ;Pops from stack to DX**

**ADD DX, 30H ;Converts to it's ASCII equivalent**

**MOV AH, 02H**

**INT 21H ;calls DOS to display character**

**LOOP Dloop2 ;Loops till CX equals zero**

**pop dx**

**pop cx**

**pop bx**

**pop ax**

**RET ;returns control**

**ENDP DisplayScore**

**;======================**

**proc RandomNumber**

**push ax**

**push dx**

**RANDSTART:**

**mov ah,00h ; interrupts to get system time**

**int 1Ah ; CX:DX now hold number of clock ticks since midnight**

**mov ax, dx**

**xor dx, dx**

**mov cx, 10**

**div cx ; here dx contains the remainder of the division - from 0 to 9**

**mov ax,13d**

**mul dx**

**mov [rnd],ax**

**pop dx**

**pop ax**

**ret**

**endp RandomNumber**

**;==========================================================**

**start:**

**mov ax, @data**

**mov ds, ax**

**mov [count],0**

**call ShowPictureStart**

**mov ah, 0 ; set display mode function.**

**mov al, 13h ; mode 13h = 320x200 pixels, 256 colors.**

**int 10h ; set it!**

**call Print**

**call DisplayScore**

**Gameloop:**

**cmp [balloonflying],01h**

**je Skip1**

**call LaunchBalloon**

**Skip1:**

**cmp [waskeypressed],01h**

**je Skip2**

**call LaunchArrow**

**Skip2:**

**call ValueUpdater**

**call CheckHit**

**cmp [hit],1h**

**jne Skip3**

**call Reset**

**jmp gameloop**

**Skip3:**

**cmp [arrowx+22],310d**

**ja exit**

**call Delay**

**jmp Gameloop**

**exit:**

**call ShowPictureLose**

**mov ax, 4c00h**

**int 21h**

**END start**

**סיכום אישי**

אני מרגיש שהפרוייקט היה משמעותי מאד בשבילי. למדתי לשלוט

בשפה אסמבלי בצורה טובה, אני מאד מרוצה מהפרוייקט שלי.

בתחילת השנה התייחסתי לאסמבלי כשפה שולית ולא שימושית,

היום אני מחשיב את אסמבלי כשפה שימושית שהייתי משתמש בה

שוב בעתיד.

**ביבליוגרפיה**

ספר גבהים של אסמבלי