תיק הפרויקט

שם הפרויקט: התחמקות.

שם התלמיד: טל ברילנט.

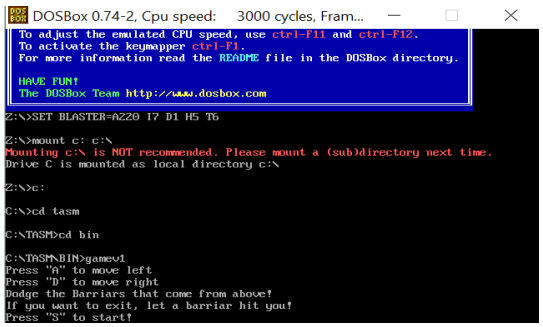
שם המנחה: יינון ברנע.

הנחיות:

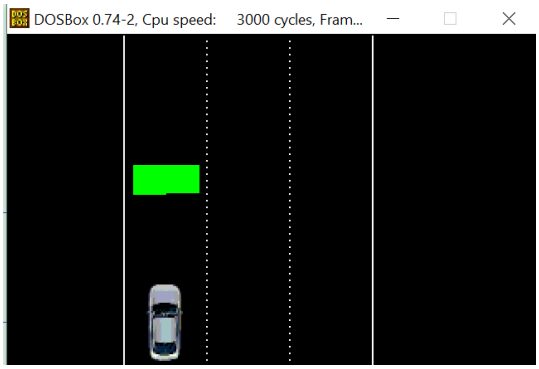
מטרת המשחק: מטרת המשחק היא להתחמק מהמכשולים שבאים מלמעלה.

במשחק המשתמש משחק בתור המכונית, ישנם שלושה מסלולים, עם המקשים על המקלדת A(שמאלה) ו – D(ימינה) השחקן יעבור בין מסלולים בכדי להתחמק מהמכשולים שבאים מלמעלה. במהלך משחק, ככל שמתחמקים מיותר מכשולים, מהירותם עולה, צבעם מתחלף ואפילו בסוף גודלם גדל. אם אחד מהמכשולים פוגע במכונית, השחקן נפסל.

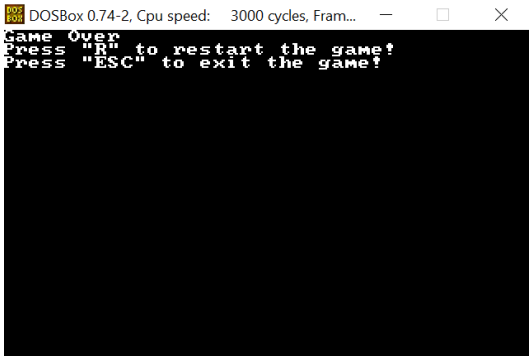
בתמונה הזו ניתן לראות את מסך ההתחלה של המשחק, כאשר לוחצים "S" המשחק יתחיל!



בתמונה הזו ניתן לראות את המחסום שיורד לכיוון המכונית, השחקן יצתרך להתחמק ממנו!



כפי שניתן לראות בתמונה הזו, השחקן כבר נפסל, המסך התנקה, אם "R" נלחץ, המשחק מתחיל מחדש, אם "ESC" נלחץ, התוכנה מפסיקה.



אלגוריתם:

במהלך התוכנית, קודם כל התוכנה "מציירת" את המסך האחורי – את מסלולי הכביש. לאחר מכן, מתחיל הלופ הראשי של המשחק שבו היא תחילה מפעילה פרוצדורת RANDOM שמגרילה מספר בין 0 ל 2 , לפי המספר שיוצא התוכנה מדפיסה וכל פעם בלופ מורידה את המחסום את כמות הפיקסלים למטה לפי כמה שהשחקן מתקדם ולפי כמה זמן הוא שרד. לאחר כל פיקסל שהתוכנה מורידה את המחסום, היא בודקת האם יש קלט מהמקלדת, אם יש, המכונית תזוז לפי הקלט, אם אין קלט, הלופ יחזור לתחילתו שבו התוכנה בודקת האם המחסום הגיע לאיזור שבו הוא יכול להיתקע במכונית. התוכנה בודקת אם הוא והמכונית באותו מסלול, אם כן, המשחק נגמר. המסך מתנקה ומופיע הודעה שבה אם השחקן לוחץ "R" התוכנה חוזרת לתחילת המשחק בעזרת קלט מהמקלדת ואם השחקן לוחץ "ESC" התוכנה הולכת ל EXIT.

פרוצדורות:

- Draw_stone – מדפיסה את המחסום על המסך בצבע לפי מה שניתן לה לפני שקוראים לה.
- makeScreen – מדפיסה על המסך את הרקע, הכביד, המסלולים.
- RANDOM – מגרילה מספר רנדומלי בין 0 – 2.
- OpenShowBmp,OpenBmpFile,CloseBmpFile,ReadBmpHeader,ReadBmpPallate,CopyBmpPallate,ShowBmp - כל הפרוצדורות האלה

הן הפרוצדורות שמדפיסות את התמונה על המסך – התמונה של המכונית.

- SetGraphics – מעבירה את הDOS למצב גרפי.
- Delete_car – מחיקת המכונית מהמקום שהיא נמצאת בו באותו הרגע.
- CLS – מחיקת המסך.

משתנים:

שם	סוג	ערך התחלתי	שימוש
Akey	byte	1Eh	סקן קוד של A
Dkey	byte	20h	סקן קוד של D
Amsg	Byte	String	הודעה
Dmsg	Byte	String	הודעה
Smsg	Byte	String	הודעה
instructions	byte	String	הודעה
loseMsg	Byte	String	הודעה
restart	Byte	String	הודעה
escape	byte	String	הודעה
ifExit	byte	String	הודעה
saveKey	byte	0	שומר את הקלט מהמקלדת
Delete_pic_x	word	?	משתנה למחיקת המכונית
delete_pic_y	word	?	משתנה למחיקת המכונית
picX	Word	0	משתנה למיקום המכונית

picY	word	0	משתנה למיקום המכונית
X	word	0	Count
y	word	0	Count
increasingStone	word	20	count
count	byte	0	משתנה ספירה
Xsize	Byte	20	לא משנה כלום
Ysize	byte	20	לא משנה כלום
Screen_color	byte	255	צבע לבן
OneBmpLine	byte	Max_bmp_width dup(0)	משתנים של תמונת המכונית
ScreenLineMax	byte	Max_bmp_width dup(0)	משתנים של תמונת המכונית
fileHandle	word	0	משתנים של תמונת המכונית
header	byte	54 dup(0)	משתנים של תמונת המכונית
Palatte	Byte	400h dup(0)	משתנים של תמונת המכונית
SmallPicName	byte	‘carpic.bmp’,0	משתנים של תמונת המכונית
BmpFileErrorMsg	byte	Error massage	משתנים של תמונת המכונית

BB	Byte	'BB','$'	הודעה
BmpLeft	Word	?	מרחק של המכונית מהקצה השמאלי של המסך
BmpTop	Word	?	מרחק המכונית מהלמעלה של המסך
BmpColSize	Word	?	גודל המכונית
BmpRowSize	word	?	גודל המכונית
startposX	word	125	משתנה של מיקום המחסומים
startposY	word	5	משתנה של מיקום המחסומים
Road_color	byte	255	צבע המסך
Stone_color	byte	?	צבע המחסומים
Score_counter	byte	0	סופר את הניקוד
route	word	?	מספר בין 0 ל 2

קשיים במהלך הפרויקט:

במהלך כתיבת הפרויקט נתקלתי בהמון קשיים. בתחילת הכתיבה הייתי צריך להבין כיצד להדפיס תמונה על המסך וכיצד להזיז אותה, נתקלתי בבעיה שבה האסמבלר הדפיס את תמונה שבה הפיקסלים לא היו בסדר הנכון וזה היה נראה כמו קוביה מטושטשת, פטרתי את הבעיה הזו בכך שהבנתי שזה קרה בגלל שעשיתי לתמונה המקורית

RESIZE וכנראה שזה בלבל את הפיקסלים. פתרתי את זה בעזרת עוד תלמיד אחר שעזר לי למצוא תוכנה שבה אני אוכל לעשות RESIZE מבלי להתעסק עם הסדר של הפיקסלים בתמונה. עוד בעיה שנתקלתי בה לאחר מכן הייתה הכנת פונקציית RANDOMIZING שתעבוד, חיפשתי באינטרנט דוגמאות לפונקציית RANDOM באסמבלי ומצאתי תוכנה שעושה את זה עם השעון כאשר היא מגרילה מספר בין 1-9, אך בגלל שבמשחק שלי השתמשתי בה בלי הפרשי זמן גדולים, יצא אותו מספר הרבה פעמים ברצף ורק אז הוא התחלף, אז בעזרת רעיון של חבר, שילבתי את מספר המאיות עם מספר השניות בעזרת הפעולה XOR במקום להשתמש רק במאיות. עוד בעיה שנתקלתי בה הייתה שלא הבנתי איך אני אמור להזיז את המכונית ורק אחרי כמה זמן הבנתי שאני אמור למחוק את התמונה ולהדפיס אותה מחדש במקום אחר. בעיה נוספת שנתקלתי בה הייתה מהירות המחסומים שבאים מלמעלה, בכלל שלא רציתי להשתמש בפונקציה שסופרת שניות, לקח לי קצת זמן לחשוב על הרעיון של להשתמש במרחק הפיקסלים בין כל הזזה. בעיה קטנה יותר שנתקלתי בה הייתה שלאחר שהמחסום עובר את המכונית, המכונית יכלה פשוט לעבוד דרך המחסום בלי שיקרה לה כלום, כדי לפתור את זה אני בודק כל 10 פיקסלים אם המכונית והמחסום נפגשים. הבעיה הכי גדולה שנתקלתי בה בהכנת הפרויקט הייתה לשלב בין שני הקבצים שהכנתי - במהלך הכנת הפרויקט בחרתי להכין קובץ אחד שבו יש את המכונית ואת האפשרות להזיז אותה ובקובץ אחר הכנתי את המחסומים שנופלים בצורה רנדומלית, היה לי ממש קשה לחשוב ולפענח איך אני אשלב את שני האלגוריתמים בלי שהם יפריעו לזרימה אחד של השני. לאחר חשיבה רבה פיענחתי לבד מה תהיה הדרך הכי טובה - לאחר שהבנתי שכל הזזה של המחסום בפיקסל אחד למטה קורית במילישניות, הבנתי שאני יכול כל פעם להזיז אותה קצת ואז לחכות לאינפוט מהמקלדת, אם לא קיבלתי אינפוט, הזזתי אותה עוד קצת. לסיכום, בפרויקט נתקלתי בבעיות רבות אך לאחר מחשבה רבה ולפעמים עזרה מחברים, הצלחתי לפתור את על הבעיות.

*אני רוצה להודות במיוחד לאורי שמיר מי7, ליהלי גלבוע מי3 ולאגם סגל שמאוד עזרו לי בפרויקט!

קוד התוכנית:

```
jumps
IDEAL
MODEL small
STACK 0f500h
MAX_BMP_WIDTH = 320
MAX_BMP_HEIGHT = 200

SMALL_BMP_HEIGHT = 40
SMALL_BMP_WIDTH = 40

DATASEG
startposX___ dw 0
startposY___ dw 0
;winmsg dw 'you won the game!','$'
loseMsg db 'Game Over',13,10,'$'
restart db 'Press "R" to restart the game!',13,10,'$'
escape db 'Press "ESC" to exit the game!','$'
Amsg db 'Press "A" to move left',13,10,'$'
Dmsg db 'Press "D" to move right',13,10,'$'
instructions db 'Dodge the Barriars that come from above!',13,10,'$'
ifExit db 'If you want to exit, let a barriar hit you!',13,10,'$'
Smsg db 'Press "S" to start!','$'
; picture and moving variables
Akey db 1Eh ; 'a' key on ascii
Dkey db 20h ; 'd' key on ascii
saveKey db 0
;erasing picture vars
delete_pic_x dw ?
delete_pic_y dw ?
picX dw 0
picY dw 0
;rock vars
x dw 0
y dw 0
increasingStone dw 20
count db 0
;startposX dw ?
;startposY dw ?
;color db ?
;Xsize db 20
;Ysize db 20
;start screen vars
screen_color db 255
;picture vars
OneBmpLine db MAX_BMP_WIDTH dup (0)  ; One Color line read buffer
```

```
ScreenLineMax db MAX_BMP_WIDTH dup (0)  ; One Color line read buffer
;BMP File data
FileHandle dw ?
Header    db 54 dup(0)
Palette db 400h dup (0)
SmallPicName db 'carPic.bmp',0
BmpFileErrorMsg     db 'Error At Opening Bmp File .', 0dh, 0ah,'$'
ErrorFile           db 0
BB db "BB..","$"
BmpLeft dw ?
BmpTop dw ?
BmpColSize dw ?
BmpRowSize dw ?
; barriers variables
;-----------------
;first row
;startposX dw 75
;startposY dw 5
;second row
startposX dw 125
startposY dw 5
;third row
;startposX dw 175
;startposY dw 5

road_color db 255
stone_color db ?

;score counter
score_counter db 0
;-----------------
route dw ?

CODESEG
start:
mov ax,@data
mov ds, ax
;-------- code here

lea dx, [Amsg] ; print msg
mov ah,09h
int 21h

lea dx,[Dmsg] ; print msg
mov ah,09h
int 21h
```

```
lea dx,[instructions] ; print msg
mov ah,09h
int 21h

lea dx,[ifExit] ; print msg
mov ah,09h
int 21h

lea dx,[Smsg] ; print msg
mov ah,09h
int 21h


input__:
xor ax,ax
in al,64h
cmp al,10b
je input__
;
in al,60h
cmp al,1Fh
je start___
jmp input__
start___:
call SetGraphic
;mov ah,13h
;int 21
;cls
draw_row___:
; compare and check if x is equal to 40
cmp [x], 300
; if x == 40 : the row is finished.
je finished_row___
; else: we need to keep drawing
inc [x]

mov bh, 0h

; add x position
mov cx, [startposX___]
add cx, [x]

; add y position
mov dx, [startposY___]
add dx, [y]

;put color in al
```

```
xor ax,ax
mov al,0

mov ah, 0ch
int 10h
jmp draw_row___

finished_row___:
; compare and check if y == 20
cmp [y], 50
je finished_square___

; else
mov [x], 0
inc [y]
jmp draw_row___


finished_square___:
mov [x],0
mov [y],0



call makeScreen
;placement on the left road

;mov [BmpLeft],84 ; placement of the car from the left
;mov [BmpTop],150 ; placement of the car from the top

;placement on the middle road

;mov [BmpLeft],134 ; placement of the car from the left
;mov [BmpTop],150 ; placement of the car from the top

;placementon the right road

;mov [BmpLeft],184 ; placement of the car from the left
;mov [BmpTop],150 ; placement of the car from the top

;the size of the picture
mov [BmpColSize],22
mov [BmpRowSize],46
;placement
mov [BmpLeft],134 ; placement of the car from the left
mov [BmpTop],150 ; placement of the car from the top
;opening the picture
```

```
mov dx,offset SmallPicName
call OpenShowBmp
;deleting the car
;mov [delete_pic_x],84
;mov [delete_pic_y],150
;call delete_car

;starting the cmpring
keyboard_input:
call random
looping:

;check if its at the end of the screen
cmp [startposY],180 ; 200-20
je stone_is_at_the_end_ ; finish
jg stone_is_at_the_end_ ; finish
;checking if the stone hit the car in spaces of 10 pixels
cmp [startposY], 130
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 131 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car

cmp [startposY], 140
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 141 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car

cmp [startposY], 150
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 151 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car

cmp [startposY], 160
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 161 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car

cmp [startposY], 170
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 171 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car

cmp [startposY], 180
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 181 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car
```

```
cmp [startposY], 190
je check_if_boom ; checking if the stone hit the car
cmp [startposY], 191 ; for the second and third speed
je check_if_boom ; checking if the stone hit the car


jmp next
stone_is_at_the_end_:
mov [startposY],5
inc [score_counter]
jmp keyboard_input
next:
;mov [startposY],5

cmp [route],0
je left
cmp [route],1
je mid
cmp [route],2
je right
left:
mov [startposX],75
jmp draw
mid:
mov [startposX],125
jmp draw
right:
mov [startposX],175
jmp draw
draw:
first_cmp:
cmp [score_counter],15
jl first_speed ;
jmp second_cmp
second_cmp:
cmp [score_counter],40
jl second_speed
jmp third_cmp
third_cmp:

cmp [score_counter],80
jl third_speed
jmp forth_speed
;cmp [score_counter], 20
;je third_speed
;jg third_speed
first_speed:
```

```
call draw_stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],250 ;color = green
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
inc [startposY]

;jmp looping
jmp inp
second_speed:
call draw_stone
call draw_stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],251 ;color = yellow
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
add [startposY],2
jmp inp
third_speed:
call draw_stone
mov cx,[startposX]
```

```
mov dx,[startposY]
mov [stone_color],249 ;color = red
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
add [startposY],3
jmp inp

forth_speed:
add [increasingStone],2
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],249 ;color = red
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
add [startposY],3
jmp inp

player_won:
;if the player passed all the barriars (which is allmost imposible)
;he won the game
inp:
in al, 64h
cmp al, 10b
```

```
je looping

in al,60h
cmp al,[saveKey] ; check if the key is the same as already pressed
je looping
mov [saveKey],al
cmp al,[Akey]
je A_was_pressed
cmp al,[Dkey]
je D_was_pressed
jmp looping
A_was_pressed:
cmp [BmpLeft],84 ; check if you cant move anymore
je looping ; dont do anything
cmp [BmpLeft],134 ; check if the car is in the middle route
je move_car_left
jmp move_car_left
D_was_pressed:
cmp [BmpLeft],184 ; check if you cant move anymore
je looping ; dont do anything
cmp [BmpLeft],134 ; check if the car is in the middle route
je move_car_right
jmp move_car_right



move_car_left:
push ax
xor ax,ax
mov ax,[BmpLeft]
mov [delete_pic_x],ax
mov [delete_pic_y],150
call delete_car
;painting it on the left spot
;the size of the picture
mov [BmpColSize],22
mov [BmpRowSize],46
;placement
sub ax,50
mov [BmpLeft],ax ; placement of the car from the left ; ax means the place we deleted
mov [BmpTop],150 ; placement of the car from the top
;opening the picture
mov dx,offset SmallPicName
call OpenShowBmp
pop ax
jmp looping
```

```
move_car_right:
push ax
xor ax,ax
mov ax,[BmpLeft]
mov [delete_pic_x],ax
mov [delete_pic_y],150
call delete_car
;painting it on the left spot
;the size of the picture
mov [BmpColSize],22
mov [BmpRowSize],46
;placement
add ax,50
mov [BmpLeft],ax ; placement of the car from the left ; ax means the place we deleted
mov [BmpTop],150 ; placement of the car from the top
;opening the picture
mov dx,offset SmallPicName
call OpenShowBmp
pop ax
jmp looping


check_if_boom:
;checking it the stone hit the car
cmp [startposX],75 ; first route
je firstRoute
cmp [startposX],125 ; second route
je secondRoute
cmp [startposX],175  ; third route
je thirdRoute
;jmp looping

firstRoute:
cmp [BmpLeft],84 ; check if the and the stone are on the same route
je gameOver ; if they are
inc [startposY] ; if they are not, the proggram goes on
jmp looping
secondRoute:
cmp [BmpLeft],134 ; check if the and the stone are on the same route
je gameOver
inc [startposY]
jmp looping
thirdRoute:
cmp [BmpLeft],184 ; check if the and the stone are on the same route
je gameOver
inc [startposY]
```

```
jmp looping
;painting again
;mov [BmpColSize],22
;mov [BmpRowSize],46
;mov [BmpLeft],134 ; placement of the car from the left
;mov [BmpTop],150 ; placement of the car from the top
;mov dx,offset SmallPicName
;call OpenShowBmp
;cmp [ErrorFile],1 ;checking if file opened
;jne cont1
;jmp exitError
gameOver:
mov [startposX],0
mov [startposY],0
mov [score_counter],0
call cls
lea dx, [loseMsg] ; print msg
mov ah,09h
int 21h

lea dx,[restart] ; print msg
mov ah,09h
int 21h

lea dx,[escape] ; print msg
mov ah,09h
int 21h

waitforkey:
in al, 64h
cmp al, 10b
je waitforkey

in al,60h
cmp al, 1h ; "esc" key
je exit

cmp al,13h ; "R" key
je start___
jne waitforkey

jmp waitforkey
cont1:
jmp exit
```

```
;jmp drawRock
exitError:
;printing error massage
mov dx, offset BmpFileErrorMsg
mov ah,9
int 21h



exit:
mov ax,4c00h
int 21h




;=========================
;==========================
;===== Procedures  Area ===
;==========================
;==========================

;procedures of the barriers

proc draw_stone
;pushing
push ax
push bx
push cx
push dx
;xoring them
xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx
draw_row:
; compare and check if x is equal to 40
cmp [x], 40
; if x == 40 : the row is finished.
je finished_row
; else: we need to keep drawing
inc [x]

mov bh, 0h
```

```
; add x position
mov cx, [startposX]
add cx, [x]

; add y position
mov dx, [startposY]
add dx, [y]

;put color in al
xor ax,ax
mov al, [stone_color]

mov ah, 0ch
int 10h
jmp draw_row

finished_row:
; compare and check if y == 20
cmp [y], 20
je finished_square

; else
mov [x], 0
inc [y]
jmp draw_row


finished_square:
mov [x],0
mov [y],0

pop dx
pop cx
pop bx
pop ax
ret



endp draw_stone

proc draw_bigstone
;pushing
push ax
push bx
push cx
push dx
```

```
;xoring them
xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx
draw_row_I:
; compare and check if x is equal to 40
cmp [x], 40
; if x == 40 : the row is finished.
je finished_row_I
; else: we need to keep drawing
inc [x]

mov bh, 0h

; add x position
mov cx, [startposX]
add cx, [x]

; add y position
mov dx, [startposY]
add dx, [y]

;put color in al
xor ax,ax
mov al, [stone_color]

mov ah, 0ch
int 10h
jmp draw_row_I

finished_row_I:
; compare and check if y == 20
push ax
mov ax,[increasingStone]
cmp [y],ax
pop ax
je finished_square_I

; else
mov [x], 0
inc [y]
jmp draw_row_I


finished_square_I:
mov [x],0
```

```
mov [y],0

pop dx
pop cx
pop bx
pop ax
ret



endp draw_bigstone

proc makeScreen
;pushing everything to save it
push ax
push bx
push cx
push dx

;clearing everything
xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx

;setting courdinations for first line
mov cx,70
mov dx,0
mov al,[road_color]

FirstLine:
inc dx
mov ah,0ch
int 10h
cmp dx,200
jl FirstLine ; if it didnt finish the line

;setting courdinations for second line
mov cx,120
mov dx,0
mov al,[road_color]

je SecondLine ; if its at the end of the screen

SecondLine:
add dx,4
mov ah,0ch
```

```
int 10h

cmp dx,200 ; if its at the end of the screen
jl SecondLine

;setting the courdinathions for third line
mov cx,170
mov dx,0
mov al,[road_color]


je ThirdLine

ThirdLine:
add dx,4
mov ah,0ch
int 10h

cmp dx,200
jl ThirdLine ;if it hasnt finished

; setting courdinations for forth line
mov cx,220
mov dx,0
mov al,[road_color]

je ForthLine ; if its at the end of the screen

ForthLine:
inc dx
mov ah,0ch
int 10h

cmp dx,200
jl ForthLine ; if it hasnt finished
je FinishScreen ; if it has finished
FinishScreen:
pop dx
pop cx
pop bx
pop ax
ret
endp makeScreen

proc move_stone_down

push ax
```

```
push bx
push cx
push dx

top_of_the_screen:
;where the stone is + putting the right color
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],247 ;color = grey
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
inc [startposY]
;check if its at the end of the screen
cmp [startposY],180 ; 200-20
je stone_is_at_the_end ; finish
;else:
jmp top_of_the_screen ; go down

stone_is_at_the_end:
;when a stone gets to the end we add to the counter one to calculate the score at the
end
inc [score_counter]

pop dx
pop cx
pop bx
pop ax
ret
endp move_stone_down

proc fast

push ax
push bx
push cx
```

```
push dx

top:
;where the stone is + putting the right color
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],8 ;color = grey
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
add [startposY],2
;check if its at the end of the screen
cmp [startposY],180 ; 200-20
je endScreen ; finish
jg endScreen
;else:
jmp top ; go down

endScreen:
;when a stone gets to the end we add to the counter one to calculate the score at the
end
inc [score_counter]

pop dx
pop cx
pop bx
pop ax
ret
endp fast
proc veryFast

push ax
push bx
push cx
push dx
```

```
top_very_fast:
;where the stone is + putting the right color
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],8 ;color = grey
mov al,[stone_color]

;print the stone
call draw_stone
;the same courdinates but the color is black - deleting the stone
mov cx,[startposX]
mov dx,[startposY]
mov [stone_color],0;color = black
mov al,[stone_color]

;print the square
call draw_stone

;the stone goes down
add [startposY],3

;check if its at the end of the screen
cmp [startposY],180 ; 200-20
je end_very_fast ; finish
jg end_very_fast
;else:
jmp top_very_fast ; go down

end_very_fast:
;when a stone gets to the end we add to the counter one to calculate the score at the
end
inc [score_counter]

pop dx
pop cx
pop bx
pop ax
ret
endp veryFast

proc start_button
push ax
push bx
push cx
push dx
```

```
pop dx
pop cx
pop bx
pop ax
ret
endp start_button



proc random
push ax
push bx
push cx
push dx
mov ah, 2Ch
int 21h

; dl = millisec
; dh = sec
xor dl, dh

xor ax, ax ; clear
mov al, dl
mov ah, 0

xor dx, dx
mov  cx, 3
div  cx       ; here dx contains the remainder of the division - from 0 to 2
mov [route], dx
pop dx
pop cx
pop bx
pop ax
ret
endp random

;procedures of the picture
; input :
; 1.BmpLeft offset from left (where to start draw the picture)
; 2. BmpTop offset from top
; 3. BmpColSize picture width ,
; 4. BmpRowSize bmp height
; 5. dx offset to file name with zero at the end
proc OpenShowBmp near
push cx
push bx
```

```
call OpenBmpFile
cmp [ErrorFile],1
je @@ExitProc
call ReadBmpHeader
; from  here assume bx is global param with file handle.
call ReadBmpPalette
call CopyBmpPalette
call ShowBMP
call CloseBmpFile
@@ExitProc:
pop bx
pop cx
ret
endp OpenShowBmp
; input dx filename to open
proc OpenBmpFile near
mov ah, 3Dh
xor al, al
int 21h
jc @@ErrorAtOpen
mov [FileHandle], ax
jmp @@ExitProc
@@ErrorAtOpen:
mov [ErrorFile],1
@@ExitProc:
ret
endp OpenBmpFile

proc CloseBmpFile near
mov ah,3Eh
mov bx, [FileHandle]
int 21h
ret
endp CloseBmpFile



; Read 54 bytes the Header
proc ReadBmpHeader near
push cx
push dx

mov ah,3fh
mov bx, [FileHandle]
mov cx,54
mov dx,offset Header
```

```
int 21h

pop dx
pop cx
ret
endp ReadBmpHeader


proc ReadBmpPalette near ; Read BMP file color palette, 256 colors * 4 bytes (400h)
; 4 bytes for each color BGR + null)
push cx
push dx

mov ah,3fh
mov cx,400h
mov dx,offset Palette
int 21h

pop dx
pop cx

ret
endp ReadBmpPalette


; Will move out to screen memory the colors
; video ports are 3C8h for number of first color
; and 3C9h for all rest
proc CopyBmpPalette near

push cx
push dx

mov si,offset Palette
mov cx,256
mov dx,3C8h
mov al,0  ; black first
out dx,al ;3C8h
inc dx  ;3C9h
CopyNextColor:
mov al,[si+2] ; Red
shr al,2 ; divide by 4 Max (cos max is 63 and we have here max 255 ) (loosing color
resolution).
out dx,al
mov al,[si+1] ; Green.
shr al,2
```

```
out dx,al
mov al,[si] ; Blue.
shr al,2
out dx,al
add si,4 ; Point to next color.  (4 bytes for each color BGR + null)

loop CopyNextColor

pop dx
pop cx

ret
endp CopyBmpPalette
proc ShowBMP
; BMP graphics are saved upside-down.
; Read the graphic line by line (BmpRowSize lines in VGA format),
; displaying the lines from bottom to top.
push cx

mov ax, 0A000h
mov es, ax

mov cx,[BmpRowSize]

mov ax,[BmpColSize] ; row size must dived by 4 so if it less we must calculate the extra
padding bytes
xor dx,dx
mov si,4
div si
mov bp,dx

mov dx,[BmpLeft]

@@NextLine:
push cx
push dx

mov di,cx  ; Current Row at the small bmp (each time -1)
add di,[BmpTop] ; add the Y on entire screen


; next 5 lines  di will be  = cx*320 + dx , point to the correct screen line
mov cx,di
shl cx,6
shl di,8
add di,cx
add di,dx
```

```
; small Read one line
mov ah,3fh
mov cx,[BmpColSize]
add cx,bp  ; extra  bytes to each row must be divided by 4
mov dx,offset ScreenLineMax
int 21h
; Copy one line into video memory
cld ; Clear direction flag, for movsb
mov cx,[BmpColSize]
mov si,offset ScreenLineMax
rep movsb ; Copy line to the screen

pop dx
pop cx

loop @@NextLine

pop cx
ret
endp ShowBMP



proc  SetGraphic
mov ax,13h   ; 320 X 200
;Mode 13h is an IBM VGA BIOS mode. It is the specific standard 256-color mode
int 10h
ret
endp SetGraphic



proc delete_car
;pushing
push ax
push bx
push cx
push dx
;xoring them
xor ax,ax
xor bx,bx
xor cx,cx
xor dx,dx
draw_row_:
; compare and check if x is equal to 40
cmp [picX], 22
```

```
; if x == 22 : the row is finished.
je finished_row_
; else: we need to keep drawing
inc [picX]

mov bh, 0h

; add x position
mov cx, [delete_pic_x]
add cx, [picX]

; add y position
mov dx, [delete_pic_y]
add dx, [picY]

;put color in al
xor ax,ax
mov al, 0 ; black color

mov ah, 0ch
int 10h
jmp draw_row_

finished_row_:
; compare and check if y == 46
cmp [picY], 46
je finished_square_

; else
mov [picX], 0
inc [picY]
jmp draw_row_


finished_square_:
mov [picX],0
mov [picY],0

pop dx
pop cx
pop bx
pop ax
ret



endp delete_car
```

```
proc cls
;clear screen
push ax
mov ax,3h
int 10h
;graphics mode
mov ax,13h
int 10h
pop ax
ret
endp cls
END start
```