תיק פרויקט אסמבלי

Pinball – חן שור

שם הפרויקט : Pinball

סביבת פיתוח : Dosbox 0.74

קומפיילר : Turbo Assembler

כתיבה : ++Notepad

סוג : משחק Pinball לשחקן אחד.

שפת תכנות : אסמבלי – tasm 8086

הכלים בהם השתמשתי בפרויקט :

קליטה מהמקלדת – בכדי לשאול את המשתמש איפה הוא רוצה להתחיל והאם הוא רוצה להתחיל מחדש את המשחק.

פתיחת תמונה – רקע אחיד, גם הרקע השחור הבסיסי מתאים אך רציתי קצת יותר צבע.

השמעת סאונד : אם הכדור פוגע במכשול יישמע סאונד מהיר.

פרוצדורות – מקצר את התוכנית, מייעל אותה וקריאה נוחה יותר.

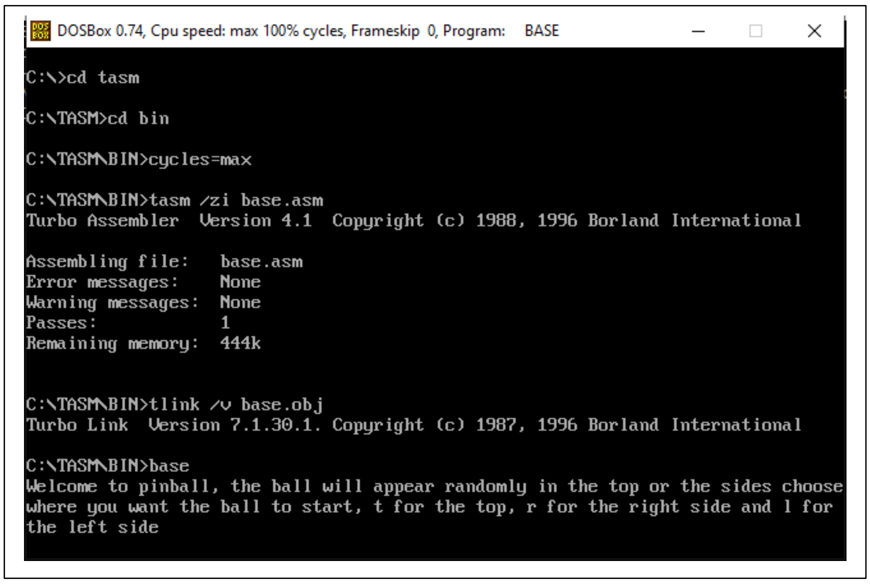
מחסנית : עזר לי לשמור ערכים, להשתמש ברג׳יסטרים שבהם היו הערכים ואז להשתמש בהם מחדש.

הדברים שלמדתי בעצמי : רנדום, לפתוח תמונה, לקלוט מהקלדת, להשמיע סאונד, להדפיס ולעדכן נקודות, דיליי, להזיז אובייקט, קריאת ערך צבע של פיקסל, קריאה מהבאפר.

מטרת המשחק : להשיג כמה שיותר נקודות לפני שהכדור יוצא מגבולות המשחק.

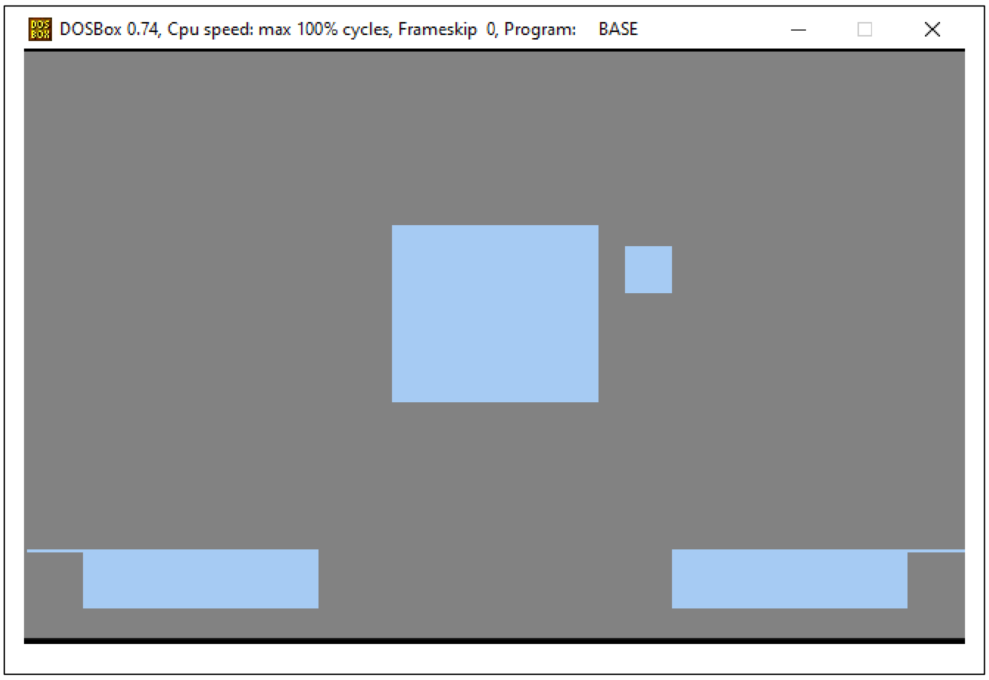
הוראות המשחק : בהתחלה מופיעה הודעה למשתמש שנותנת לו את הבחירה להחליט איפה הכדור יופיע, ימינה למעלה או למטה (ברירת מחדל למעלה), במהלך המשחק המשתמש יכול להזיז את השערים שלמטה בעזרת w וחץ למעלה ולנסות להשיג כמה שיותר נקודות.

תמונות מהמשחק:

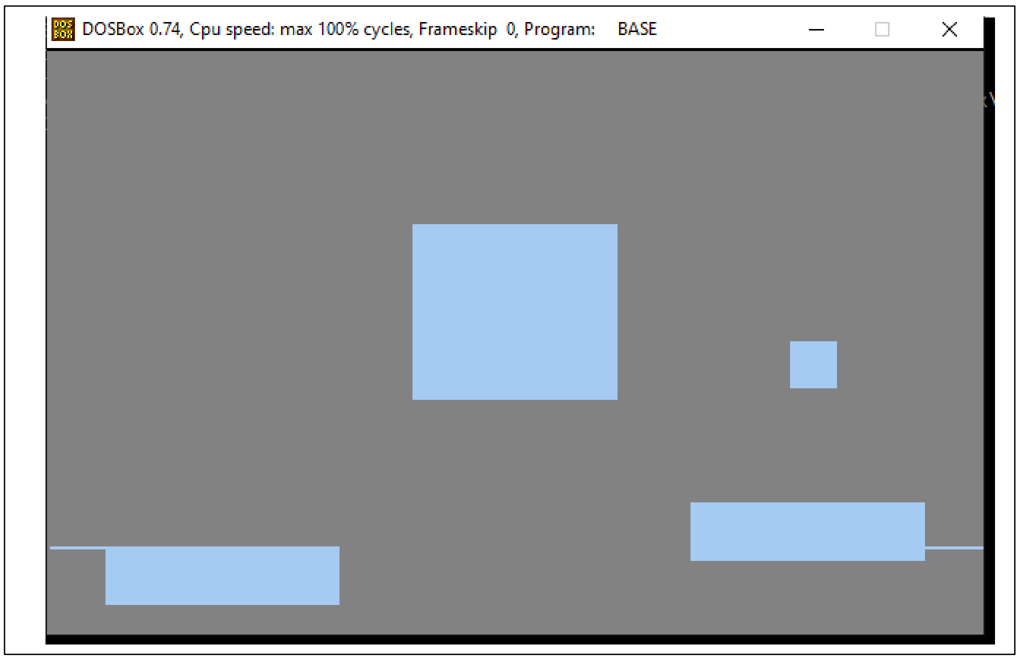


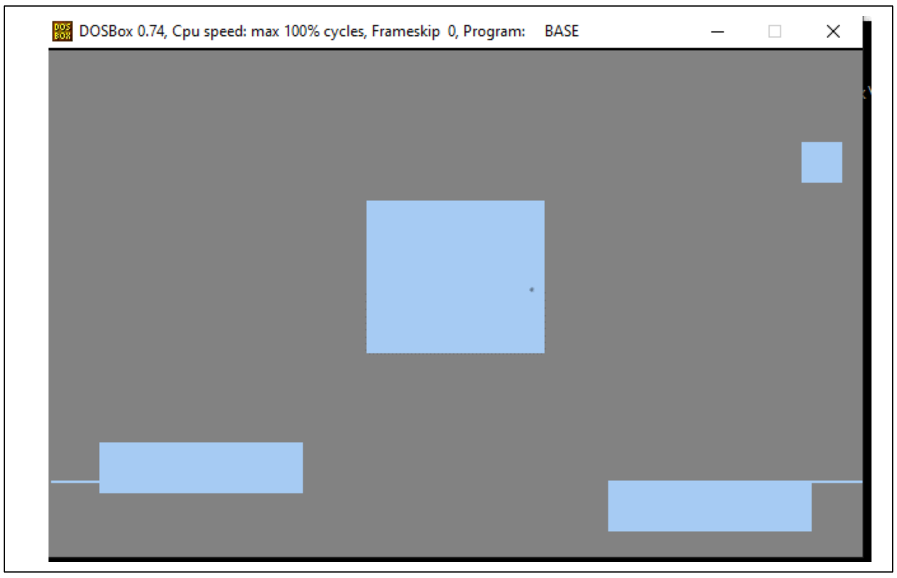
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BASE

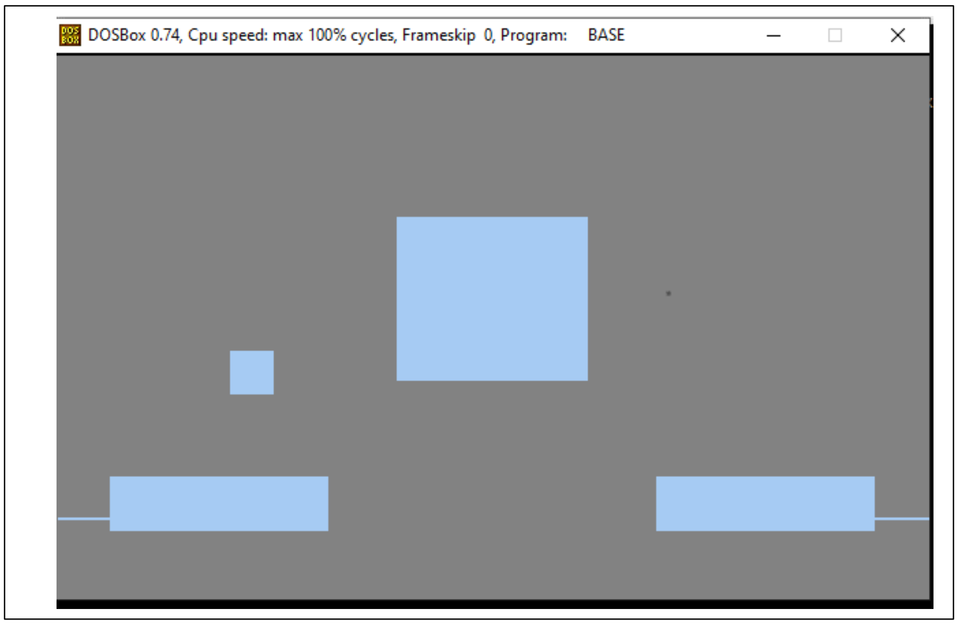
```
C:\>cd tasm

C:\TASM>cd bin

C:\TASM\BIN>cycles=max

C:\TASM\BIN>tasm /zi base.asm
Turbo Assembler  Version 4.1  Copyright (c) 1988, 1996 Borland International

Assembling file:   base.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  444k


C:\TASM\BIN>tlink /v base.obj
Turbo Link  Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International

C:\TASM\BIN>base
Welcome to pinball, the ball will appear randomly in the top or the sides choose
where you want the ball to start, t for the top, r for the right side and l for
the left side
```

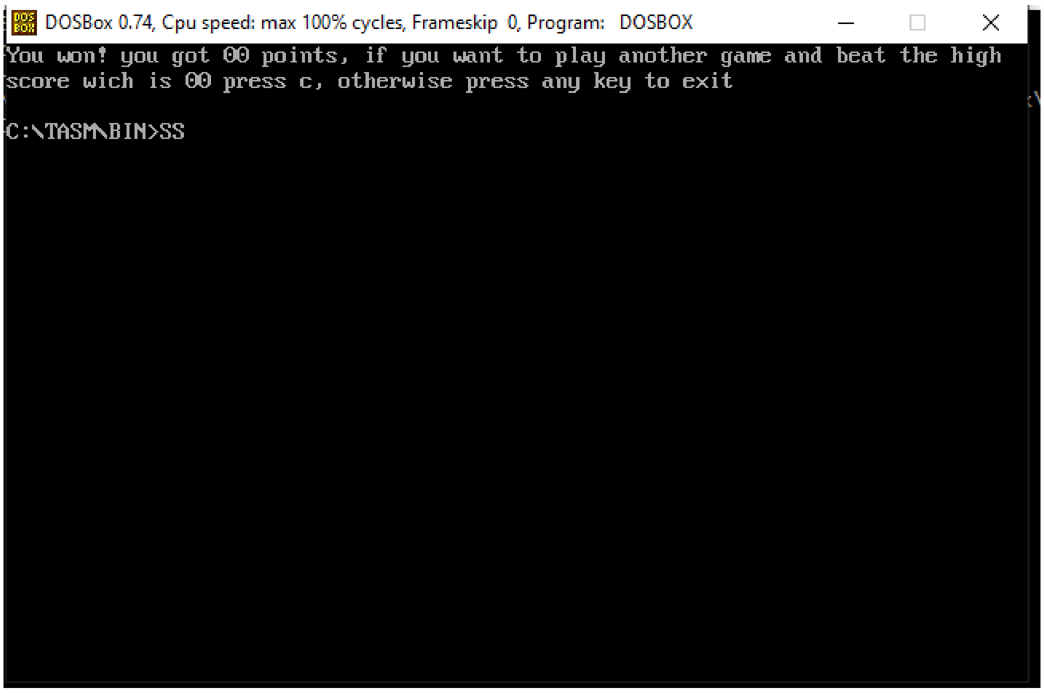


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BASE


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BASE


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BASE


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BASE


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: DOSBOX

```
You won! you got 00 points, if you want to play another game and beat the high
score wich is 00 press c, otherwise press any key to exit

C:\TASM\BIN>SS
```

רפלקציה

לפרויקט שלי היו 2 חלקים

חלק ראשון – היה לי מאוד קל לעשות את מה שחשבתי עליו ובא לי בקלות הקוד.

חלק שני – הגעתי למצבים שהייתי צריך לחשוב המון זמן על איך לעשות את מה שרציתי, סיבכתי את עצמי ובשבילי היו דברים מסובכים לוגית שלקח לי זמן לפתור.

בסופו של דבר אני יודע שאם הייתי מתחיל את הפרויקט מוקדם יותר (אפילו שלא יכולתי) אולי הוא היה יותר טוב מעט אבל אני בטוח שהעומס שהיה עליי בזמן האחרון לא היה כמו שהוא, בסופו של דבר אי אפשר לשנות את העבר ואני מרוצה מהפרויקט מהזמן שעשיתי אותו.

ניתן לפתח את הפרויקט בדרכים הבאות:

מכשולים בצורת עיגול או לא ישרים

להוסיף מכשולים

קוד התוכנית:

```
IDEAL IDEAL
MODEL small
p386
STACK 100h
DATASEG

checkby1 dw 165
checkbx1 dw 40
checkby dw 165
checkbx dw 250
time db 0
jump db 3
keeprectx dw ?
rectx dw ?
recty dw ?
keeprecty dw ?
```

```
saveKey db 0
wid dw ?
dev db 10
note dw 1000h
hscore dw 0
score dw ?
score1 db '0$'
hscore1 db '0$'
score2 db '0$'
hscore2 db '0$'
score3 db '0$'
hscore3 db '0$'
keepy dw ?
x_velocity dw 6h
y_velocity dw 3h
keepx dw ?
filename db 'CHEN3.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 ,'$'
message db 'Welcome to pinball, the ball will appear randomly in the top
or the sides choosewhere you want the ball to start, t for the top, r for the
right side and l for the left side',13,10,'$'
message1 db 'You won! you got $'
message2 db ' points, if you want to play another game and beat the high
score wich is $'
message3 db ' press c, otherwise press any key to exit',13,10,'$'
Clock equ es:6Ch
color db ?
```

```
Height dw ?
Widthh dw ?
x dw ?
y dw ?
key db ?

CODESEG

proc GAME
call BALL
return:
call rect
no_jump:
call timer
mov [color],248
mov cx,[keepy]
mov [y],cx
call BALL
; y and x
mov cx,[x_velocity]
add [keepx],cx
mov cx,[keepx]
mov [x],cx
mov cx,[y_velocity]
add [keepy],cx
mov cx,[keepy]
mov [y],cx
; check limits
cmp [y],180
jl a
```

```
jmp End_Game
a:
cmp [x],7
jge b
jmp neg_velocityx
b:
cmp [x],298
jle c
jmp neg_velocityx
c:
cmp [y],4
jge d
jmp neg_velocityy
d:
cmp [y],151
jle box
jmp neg_velocityltr
box:
cmp [y],44
jl left1
cmp [y],121
jg left1
cmp [x],202
jge left1
cmp [x],196
jl left1
neg [x_velocity]
add [score],10
call sound
jmp hit
```

```
left1:
cmp [y],44
jl top
cmp [y],121
jg top
cmp [x],103
jle top
cmp [x],124
jg top
neg [x_velocity]
add [score],10
call sound
jmp hit
top:
cmp [x],109
jl bottom
cmp [x],196
jg bottom
cmp [y],40
jle bottom
cmp [y],43
jg bottom
neg [y_velocity]
add [score],10
call sound
jmp hit
bottom:
cmp [x],109
jl hit
cmp [x],196
```

```
jg hit
cmp [y],124
jge hit
cmp [y],121
jl hit
neg [y_velocity]
add [score],10
call sound
hit:
mov [color],9
call BALL
jmp return
ret
endp GAME

proc send
push seg message1
pop ds
mov dx, offset message1
mov ah, 9h
int 21h
push seg score1
pop ds
mov dx, offset score1
mov ah, 9h
int 21h
push seg score2
pop ds
mov dx, offset score2
mov ah, 9h
```

```
int 21h
push seg score3
pop ds
mov dx, offset score3
mov ah, 9h
int 21h
push seg message2
pop ds
mov dx, offset message2
mov ah, 9h
int 21h
ret
endp send

proc rect
WaitForKey:
mov [jump],15
;check if there is a a new key in buffer
in al,64h
cmp al,10b
je no_jump
in al,60h
   cmp al,11h
je yes
cmp al,91h
je KeyReleased
cmp al,48h
je yes2
cmp al,0c8h
je KeyReleased1
```

```
jmp no_jump
yes:
; check if the key is same as already pressed
cmp al,[saveKey]
jne here5
cmp [x],101
jg here6
cmp [x],4
jl here6
cmp [y],151
jle here6
cmp [y],154
jg here6
neg [y_velocity]
call sound
jmp no_jump
here6:
cmp [y],154
jl here9
cmp [y],176
jg here9
cmp [x],107
jge here9
neg [x_velocity]
call sound
here9:
jmp no_jump
here5:
; new key- store it
mov [saveKey],al
```

```
KeyPressed:
mov [keeprecty],170
KeyPressed2:
; left rectangle
mov [color],248
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [rectx],20
mov [keeprectx],20
mov [Height],20
call rectangle
dec [keeprecty]
mov [color],9
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [Height],20
call rectangle
dec [keeprecty]
cmp [x],101
jg here2
cmp [x],4
jl here2
mov ax,[keeprecty]
cmp [y],ax
jg here2
sub ax,3
cmp [y],ax
```

```
jle here2
inc [keeprecty]
neg [y_velocity]
neg [x_velocity]
call sound
jmp KeyReleased
here2:
inc [keeprecty]
call timerrect
dec [jump]
cmp [jump],0
jne KeyPressed2
jmp no_jump

KeyReleased:
; left rectangle
mov [color],248
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [rectx],20
mov [keeprectx],20
mov [Height],20
call rectangle
mov [color],9
mov [wid],80
mov [recty],170
mov [Height],20
call rectangle
mov [saveKey],0
```

```
jmp no_jump

yes2:
;check if the key is same as already pressed
cmp al,[saveKey]
jne here8
cmp [x],204
jl here7
cmp [x],301
jg here7
cmp [y],151
jle here7
cmp [y],154
jg here7
neg [y_velocity]
call sound
jmp no_jump
here7:
cmp [y],154
jl here10
cmp [y],176
jge here10
cmp [x],198
jle here10
neg [x_velocity]
call sound
here10:
jmp no_jump
here8:
;new key- store it
```

```
mov [saveKey],al

KeyPressed1:
;right rectangle
mov [keeprecty],170
KeyPressed3:
mov [color],248
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [rectx],220
mov [keeprectx],220
mov [Height],20
call rectangle
dec [keeprecty]
mov [color],9
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [Height],20
call rectangle
dec [keeprecty]
cmp [x],204
jl here4
cmp [x],301
jg here4
mov ax,[keeprecty]
cmp [y],ax
jg here4
sub ax,3
```

```
cmp [y],ax
jle here4
inc [keeprecty]
neg [y_velocity]
neg [x_velocity]
call sound
jmp KeyReleased
here4:
inc [keeprecty]
call timerrect
dec [jump]
cmp [jump],0
jne KeyPressed3
jmp no_jump

KeyReleased1:
;right rectangle
mov [color],248
mov [wid],80
mov ax,[keeprecty]
mov [recty],ax
mov [rectx],220
mov [keeprectx],220
mov [Height],20
call rectangle
mov [color],9
mov [wid],80
mov [recty],170
mov [Height],20
call rectangle
```

```
mov [saveKey],0
ret
endp rect

proc sound
in al, 61h
or al, 00000011b
out 61h, al
mov al, 0B6h
out 43h, al
mov ax, [note]
out 42h, al ; Sending lower byte
mov al, ah
out 42h, al ; Sending upper byte
call timer
in al, 61h
and al, 11111100b
out 61h, al
ret
endp sound

proc BALL
mov [Height],16
Loopks:
mov[Widthh],16
Loopah:
call pixel
inc [x]
dec [Widthh]
cmp [Widthh],0
```

```
jne Loopah
inc [y]
mov cx,[keepx]
mov [x],cx
dec [Height]
cmp [Height],0
jne Loopks
ret
endp BALL

proc OpenFile
; Open file
mov ah, 3Dh
xor al, al
mov dx, offset filename
int 21h
jc openerror
mov [filehandle], ax
ret
openerror :
mov dx, offset ErrorMsg
mov ah, 9h
int 21h
ret
endp OpenFile

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 BMP
;Process BMP file
call OpenFile
call ReadHeader
call ReadPalette
call CopyPal
call CopyBitmap
ret
endp BMP
```

```
proc rectpixel
mov bh,0h
mov cx,[rectx]
mov dx,[recty]
mov al,[color]
mov ah,0ch
int 10h
ret
endp rectpixel

proc pixel
;Print dot
mov bh,0h
mov cx,[x]
mov dx,[y]
mov al,[color]
mov ah,0ch
int 10h
ret
endp pixel

proc RandR
; initialize
mov ax,40h
mov es,ax
mov bx,0
RandLoop3:
mov ax,[Clock] ;read timer counter
mov ah,[byte cs:bx] ;read one byte from memory
xor al,ah ;xor memory and counter
```

```
and al,11111111b ; random
inc bx
cmp al,148
ja RandLoop3
cmp al,0
je RandLoop3
mov ah,0
mov [y],ax
mov [keepy],ax
mov [x],297
mov [keepx],297
ret
endp RandR

proc RandL
; initialize
mov ax,40h
mov es,ax
mov bx,0
RandLoop2:
mov ax,[Clock] ; read timer counter
mov ah,[byte cs: bx] ; read one byte from memory
xor al,ah ; xor memory and counter
and al,11111111b ; random
inc bx
cmp al,148
ja RandLoop2
cmp al,0
je RandLoop2
mov ah,0
mov [y],ax
mov [keepy],ax
mov [x],7
mov [keepx],7
ret
endp RandL

proc RandT
; initialize
mov ax,40h
mov es,ax
mov bx,0
Loopi:
mov ax,[Clock] ; read timer counter
mov ah,[byte cs:bx] ; read one byte from memory
xor al,ah ; xor memory and counter
and al,11111111b ; random
inc bx
cmp al,151
ja Loopi
cmp al,0
je Loopi
mov bl,2
mul bl
cmp ax,160
jge here3
neg [x_velocity]
here3:
mov [x],ax
mov [keepx],ax
mov [y],7
mov [keepy],7
ret
endp RandT

proc timer
ziv:
mov ah,2ch
int 21h
cmp dl,[time]
je ziv
mov [time],dl
ret
endp timer

proc timerrect
mov cx,1h
delRep:
push cx
mov cx,0d090h
delDec:
dec cx
jnz delDec
pop cx
dec cx
jnz delRep
ret
endp timerrect

proc rectangle
```

```
LoopA:
mov ax,[wid]
mov [Widthh],ax
LoopB:
call rectpixel
inc [rectx]
dec [Widthh]
cmp [Widthh],0
jne LoopB
inc [recty]
mov ax,[keeprectx]
mov [rectx],ax
dec [Height]
cmp [Height],0
jne LoopA
ret
endp rectangle

start:

mov ax, @data
mov ds, ax

mov [score],0
mov [color],9

; send message
push seg message
pop ds
mov dx, offset message
mov ah,9h
int 21h
mov ah,0
int 16h
mov [key],al
mov ax,13h
int 10h
call BMP

; cube
mov [rectx],125
mov [keeprectx],125
mov [recty],60
mov [Height],60
mov [wid],70
call rectangle

; left rectangle
mov [wid],80
mov [rectx],20
mov [keeprectx],20
mov [recty],170
mov [Height],20
call rectangle

; line
mov [x],20
mov [y],170
mov[Widthh],20
line:
```

```
call pixel
dec [x]
dec [widthh]
cmp [widthh],0
jne line

;right rectangle
mov [wid],80
mov [recty],170
mov [rectx],220
mov [keeprectx],220
mov [Height],20
call rectangle

;line
mov [x],300
mov [y],170
mov[Widthh],20
line1:
call pixel
inc [x]
dec [widthh]
cmp [widthh],0
jne line1

cmp [key],'t'
jne key2
jmp Rand1

key2:
```

```
cmp [key],'l'
jne key3
jmp Rand2

key3:
cmp [key],'r'
jne Rand1
jmp Rand3

Rand1:
; draw ball
call RandT
call GAME

Rand2:
; draw ball
call RandL
call GAME

Rand3:
; draw ball
call RandR
neg [x_velocity]
call GAME

neg_velocityx:
neg [x_velocity]
jmp hit

neg_velocityy:
```

```
neg [y_velocity]
jmp hit

neg_velocityltr:
cmp [y],155
jge s
cmp [x],101
jg w
neg [y_velocity]
call sound
jmp hit
w:
cmp [x],204
jl s
neg [y_velocity]
call sound
jmp hit
s:
cmp [x],198
jle no_2
neg [x_velocity]
call sound
jmp hit
no_2:
cmp [x],107
jge no_1
neg [x_velocity]
call sound
no_1:
jmp hit
```

```
End_Game:
mov ah,0
mov al,2
int 10h
mov ax,[score]
div [dev]
cmp al,9
jg here
jmp two_num
here:
div [dev]
add al,30h
mov [score1],al
add ah,30h
mov [score2],ah
mov ax,[hscore]
cmp [score],ax
jg HS1
jmp send1
HS1:
call send
push seg score1
pop ds
mov dx, offset score1
mov ah, 9h
int 21h
push seg score2
pop ds
mov dx, offset score2
mov ah, 9h
int 21h
push seg score3
pop ds
mov dx, offset score3
mov ah, 9h
int 21h
push seg message3
pop ds
mov dx, offset message3
mov ah, 9h
int 21h
mov ax,[score]
mov [hscore],ax
mov al,[score1]
mov [hscore1],al
mov al,[score2]
mov [hscore2],al
jmp keybord
send1：
call send
push seg hscore1
pop ds
mov dx, offset hscore1
mov ah, 9h
int 21h
push seg hscore2
pop ds
mov dx, offset hscore2
mov ah, 9h
```

```
int 21h
push seg hscore3
pop ds
mov dx, offset hscore3
mov ah, 9h
int 21h
push seg message3
pop ds
mov dx, offset message3
mov ah, 9h
int 21h
jmp keybord

two_num:
add al,30h
mov [score1],al
add ah,30h
mov [score2],ah
mov ax,[hscore]
cmp [score],ax
jg HS
jmp send2
HS:
push seg message1
pop ds
mov dx, offset message1
mov ah, 9h
int 21h
push seg score1
pop ds
```

```
mov dx, offset score1
mov ah, 9h
int 21h
push seg score2
pop ds
mov dx, offset score2
mov ah, 9h
int 21h
push seg message2
pop ds
mov dx, offset message2
mov ah, 9h
int 21h
push seg score1
pop ds
mov dx, offset score1
mov ah, 9h
int 21h
push seg score2
pop ds
mov dx, offset score2
mov ah, 9h
int 21h
push seg message3
pop ds
mov dx, offset message3
mov ah, 9h
int 21h
mov ax,[score]
mov [hscore],ax
mov al,[score1]
mov [hscore1],al
mov al,[score2]
mov [hscore2],al
jmp keybord
send2：
push seg message1
pop ds
mov dx, offset message1
mov ah, 9h
int 21h
push seg score1
pop ds
mov dx, offset score1
mov ah, 9h
int 21h
push seg score2
pop ds
mov dx, offset score2
mov ah, 9h
int 21h
push seg message2
pop ds
mov dx, offset message2
mov ah, 9h
int 21h
push seg hscore1
pop ds
mov dx, offset hscore1
mov ah, 9h
```

```
int 21h
push seg hscore2
pop ds
mov dx, offset hscore2
mov ah, 9h
int 21h
cmp [hscore],99
jl k
push seg hscore3
pop ds
mov dx, offset hscore3
mov ah, 9h
int 21h
k:
push seg message3
pop ds
mov dx, offset message3
mov ah, 9h
int 21h
keybord:
mov ah,0ch
mov al,07h
int 21h

cmp al,'c'
jne exit
jmp start

exit:
mov ax, 4c00h
```

```
int 21h
END start
```