Flappy Bird

Mike Greenbaum/211747639



# תוכן עניינים

[תוכן עניינים 2](#_Toc381916210)

[מבוא 3](#_Toc381916211)

[מדריך משתמש 4](#_Toc381916212)

[אמצעי הקלט 4](#_Toc381916213)

[סביבת עבודה 5](#_Toc381916214)

[סביבת פיתוח 5](#_Toc381916215)

[סביבת הרצה 5](#_Toc381916216)

[פונקציות שסופקו ע"י המורה 5](#_Toc381916217)

[מבנה תוכנה 5](#_Toc381916218)

[תרשים זרימה ראשי 6](#_Toc381916219)

[עץ קריאות - מקרא 7](#_Toc381916220)

[עץ קריאות - ראשי 7](#_Toc381916221)

[תיאור פונקציות עיקריות 8](#_Toc381916223)

[checkColBirdColision 8](#_Toc381916224)

[Random 8](#_Toc381916224)

[drawLineOfSqaures 8](#_Toc381916224)

[checkButtonColision 8](#_Toc381916224)

[אלגוריתמים ידועים / מעניינים 9](#_Toc381916225)

[לבדוק האם הציפור התנגשה בעמוד 9](#_Toc381916226)

[תיאור בעיות ואתגרים והתמודדות איתן 10](#_Toc381916227)

[גודל ציפור 10](#_Toc381916228)

[הבהוב ציפור 10](#_Toc381916228)

[הצעות להרחבה עתידית 11](#_Toc381916229)

[הכנת קפה 11](#_Toc381916230)

[עוד מכשולים 11](#_Toc381916230)

[מרחקים בין מכשולים 11](#_Toc381916230)

[מסך לאחר הפסד 11](#_Toc381916230)

[ניקוד מקסימאלי 11](#_Toc381916230)

[תודות 12](#_Toc381916231)

[נספחים 13](#_Toc381916232)

[נספח קוד מקור 13](#_Toc381916233)

# מבוא

תוכנית הflappy bird, נבחרה על ידי, כי אני חושב שזה היו משחק מצויין שכיף לשחק בו, אך מהיות והוא די קל לתכנות נאלצתי לעבוד הרבה על העיצוב של הדמות.

התוכנית נכתבה בשפת אסמבלי של 8086, עבור יחידת האסמבלי בבגרות במדעי המחשב

# 

# מדריך משתמש

המשחק הינו FLAPPY BIRD

## כללים בסיסיים

1. המשחק נועד לשחקן יחיד.
2. לוחצים SPACE BAR כדי לעלות את הציפור
3. הציפור נופלת כשלא לוחצים SPACE BAR
4. צריך לעבור במרווח שבין המכשולים כדי לא למות
5. אסור לגעת באדמה
6. עבור כל מכשול שעברת בהצלחה אתה מקבל נקודה

## אמצעי הקלט



כפתור בשביל לגרום לציפור לקפוץ

עכבר בשביל ללחוץ להתחלה וליציאה

# סביבת עבודה

## סביבת פיתוח

סביבת הפיתוח הינה התוכנה EMU8086.

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

האמולטור הינו סביבת עבודה משולבת (IDE), הכוללת

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

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

לאמולטור קיים חיסרון, הרצת התוכנית איטית מאוד.

## סביבת הרצה

סביבת DOSBOX הינה סביבת ההרצה של התוכנית.

DOSBOX הינו אמולטור מלא של מחשב 8086, הכולל מערכת הפעלה תואמת ל-DOS.

היתרון שלו, הוא שהוא מסוגל להריץ תוכניות COM, באופן מהיר.

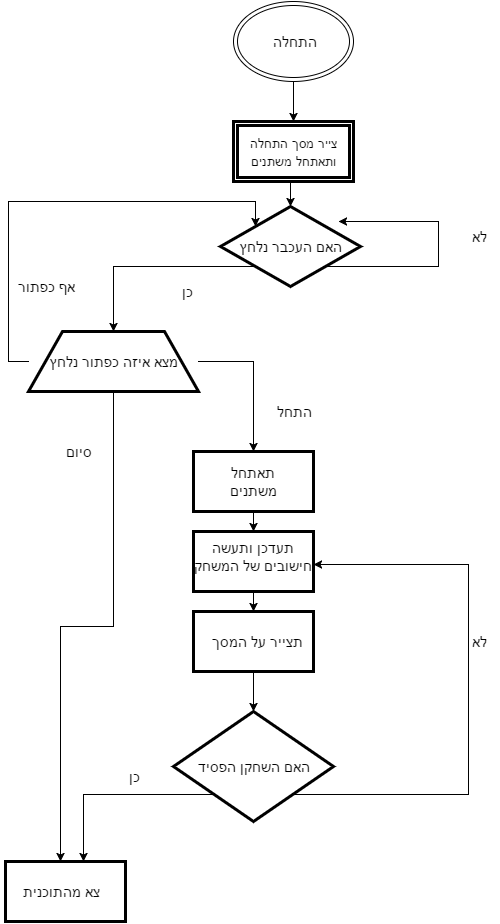
## פונקציות שסופקו ע"י המורה

הפונקציות שנמצאות בתוך קובץ ה-gvahim.asm וgvahim.mac סופקו ע"י בית הספר.

מתוכם, השתמשתי ב **print\_num** כמצויין בהמשך.

# מבנה תוכנה

## תרשים זרימה ראשי



## עץ קריאות - מקרא

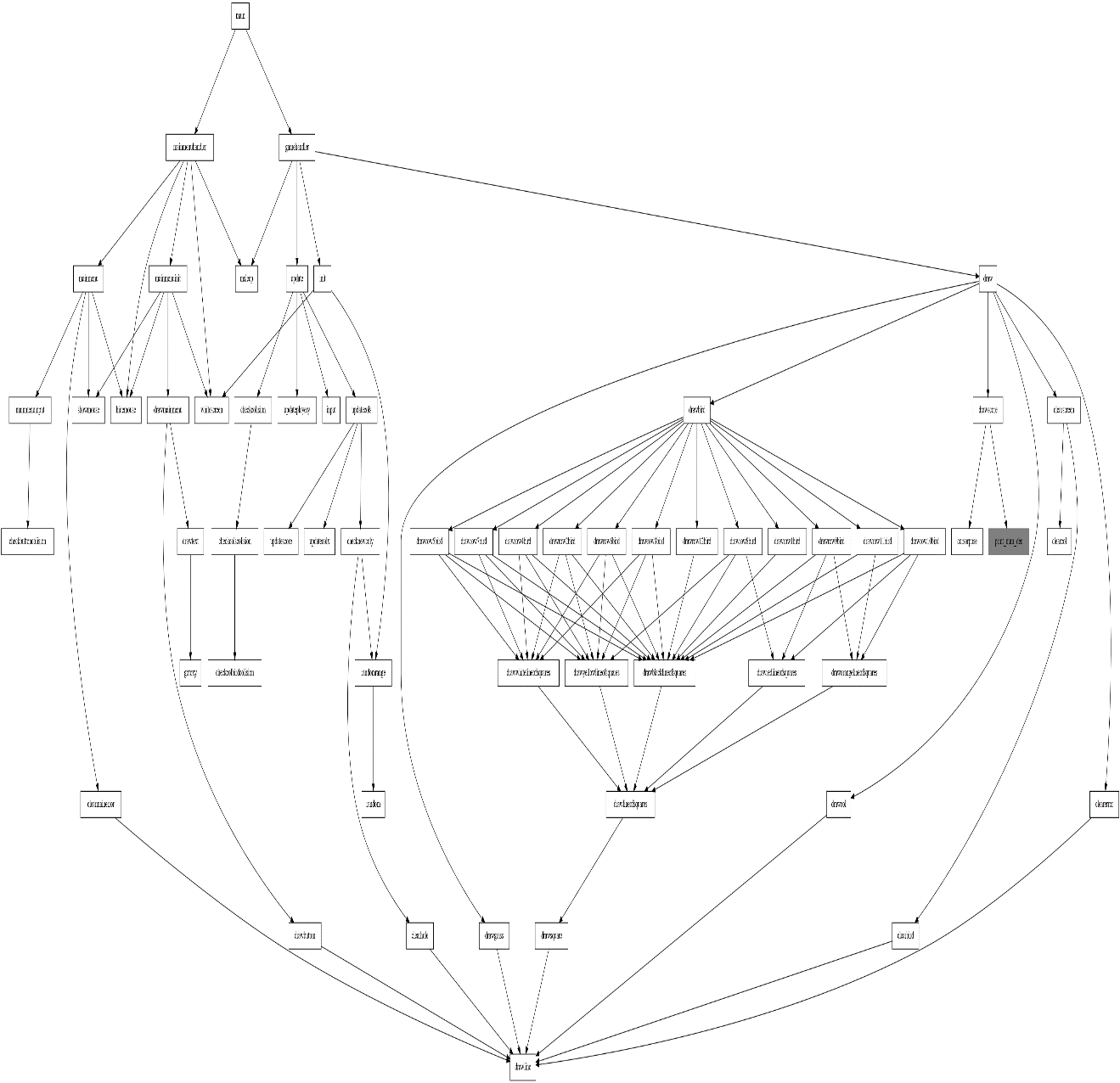
## 

תת עץ   
 ראה תרשים נוסף

פונקצית עזר נתונה מראש

פונקציה רגילה

## עץ קריאות - ראשי



## תיאור פונקציות עיקריות

### checkColBirdColision

הפונקציה בודקת האם הציפור נגעה בעמוד

### Random

הפונקציה נותנת מספר רנדומאלי בין 0 ל גובה המסך – גובה החור – גובה הדשא

### drawLineOfSquares

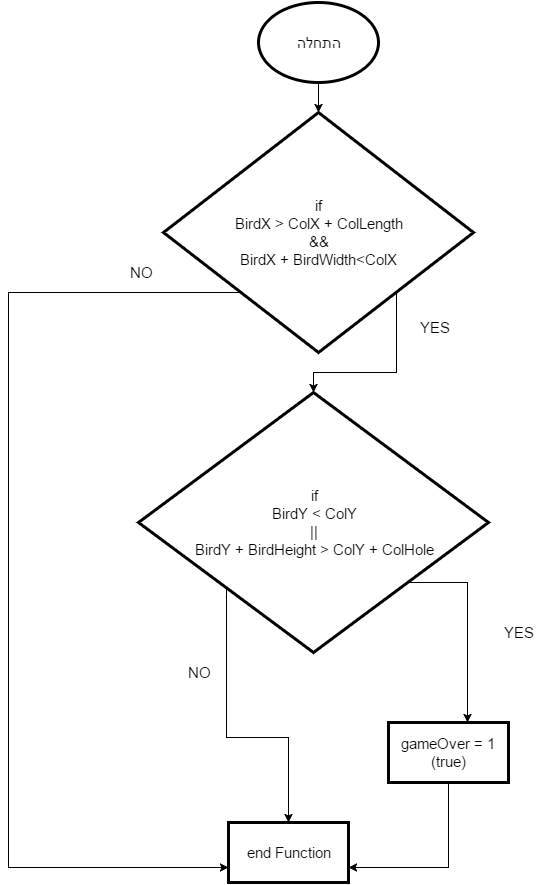
הפונקציה מציירת BX ריבועים בשורה באורך וברוחב של Squaresize

### CheckButtonColision

הפונקציה בןדקת האם יש כפתור שנלחץ ואם כן אז איזה כפתור ומבצעת את הפעולה

# אלגוריתמים ידועים / מעניינים

## לבדוק האם הציפור התנגשה בעמוד



# תיאור בעיות ואתגרים והתמודדות איתן

## הבהוב הציפור

אחרי כל ניקיון של המסך הציפור הבהבה בלבן והיה קשה להתרכז במשחק

במקום לנקות את כל הלוח אני מנקה רק את המקומות שהיו בהם דברים בפריים הקודם ולא רוצים אותם בפריים הזה

## גודל הציפור

בהתחלה כשהרצתי בemu8086 הציפור נראתה קטנה מדי יחסית למשחק

יצרתי פונקציה שמציירת במקום פיקסל ריבוע בגודל X על X ואז אם נציב X=1 נקבל את הציפור המקורית ונוכל לשנות את הערך אם נצטרך אפילו בסוף הסתבר שלא הייתי זקוק לזה

# הצעות להרחבה עתידית

## הכנת קפה

המשחק גם יכין קפה ...

## שמירת הניקוד הכי גבוה

הוספת אפשרות לשמור את הניקוד הכי גבוה שהגעתי ולהציג אותו במסך.

הצעת ישום קצרה :דבר זה כנראה יבוצע ,על ידי רישום בקובץ של התוצאה הכי גבוהה לאחר כל משחק ולהציגה מהקובץ על המסך.

## להוסיף מסך לאחר הפסד

הוספת אפשרות לחזור למסך הראשי ולמשחק.

הצעת ישום קצרה :דבר זה כנראה יבוצע ,על ידי שימוש באותן הפונקציות למסך הראשי פוט לשנות חלק מהמשתנים

## להוסיף עוד מכשולים

הוספת אפשרות לקבוע את כמות המכשולים על המסך.

הצעת ישום קצרה :דבר זה כנראה יבוצע ,על ידי רישום של כל המכשולים על המסך במערך ולעבור על כולו במקום על כל אחד בנפרד עם מערך דו מימדי לX ולY

## מרחקים רנדומאליים בין המכשולים

הוספת אפשרות לקבל את המרחק בין במכשולים וליצור אותו באופן רנדומאלי.

הצעת ישום קצרה : כשאתה מתחיל את המשחק לקרוא לרנדום בשביל לקבל מיקומים למכשולים ולהריץ עד שהמרחק בינהם הוא לפחות רבע מסך וקטן משלושה רבעי מסך

# תודות

תודה לשאול שמולה שלימד אותנו את שפת התכנות ונתן לנו את הבסיס שהיינו זקוקים לו

# נספחים

## נספח קוד מקור

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

; GVAHIM

;

; Template program for .COM files

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

ideal

model tiny

include "c:\gvahim\gvahim.mac"

dataseg

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

; Begin Data definitions

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

PlayerX dw ? ; Bird's X pos

PlayerY dw ? ; Bird's Y pos

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

squareH db ? ; a var to hold bx which is to draw a square

numOfSquares db ? ; a var to hold how many squares to draw

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

PlayerAdder dw ? ; how many frame to jump

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

Col1X dw ? ; Col 1 X Pos

Col2X dw ? ; Col 1 Y Pos

Col1Y dw ? ; Col 2 X Pos

Col2Y dw ? ; Col 2 Y Pos

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

HoleCleaner dw ?

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

gameOver dw ? ; is game over? 0 no 1 yes

Score dw ? ; player score

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

ColPlace dw ? ; used in draw col to check how long to draw

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

cursorX db ?

cursorY db ?

ColSpeed equ 2; how fast does the col moves

ColHole equ 60; how tall is the hole

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

ScreenWidth equ 320 ; Screen Width

ScreenHeight equ 200 ; Screen Height

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

Squaresize equ 1 ; HOw much is every square in the birds body

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

ColLength equ 14 ; how long is the col

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

VelocityJump equ 5 ; how fast does the bird jump

VelocityFall equ 3 ; how fast the bird falls

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

PlayerHeight equ 12 \* Squaresize; how tall is the bird

PlayerWidth equ 17 \* Squaresize; how fat is the bird

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

FramesToGoUp equ 8; how many frames does the bird need to jump after space was pressed

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

GrassHeight equ 5 ; how tall is the grass

ClearHeight equ 10

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

SleepTimes equ 50\*1000; how much to sleep

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

ScoreHeight equ 0 ; where is the score X

ScoreWidth equ 19 ; where is the score Y

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

ErrorCleaner equ 20 ; clean error in the main menu with the mouse

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

ButtonLength equ 70 ; how long is the button

ButtonHeight equ 35 ; how tall is the button

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

MainMenuButton1X equ ScreenWidth / 2 - ButtonLength / 2 + 3 ; X of button 1 pos

MainMenuButton1Y equ ScreenHeight / 6 - ButtonHeight / 2 + 2 ; Y of button 1 pos

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

MainMenuButton2X equ MainMenuButton1X - 4 ; X of button 2 pos

MainMenuButton2Y equ ScreenHeight - MainMenuButton1Y ; ; Y of button 2 pos

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

firstTextX equ 18 ; first text X pos

firstTextY equ 4 ; first text Y pos

firstStringColor equ 0Fh ; first text color

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

secondTextX equ 18 ; second text X pos

secondTextY equ 21 ; second text Y pos

secondStringColor equ 0Fh ; second text color

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

ButtonColor db ? ; which color is the button

ButtonAdder dw ? ; a var to draw the button

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

TextXInput db ? ; input to place cursor X

TextYInput db ? ; input to place cursor Y

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

gameStartLoop dw ? ; do i need to start the game

pressed dw ? ; if a button waas pressed 1 else 0

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

macro PRINT sdat

LOCAL next\_char, s\_dcl, printed, skip\_dcl

PUSH AX ; store registers...

PUSH SI ;

push bx

JMP skip\_dcl ; skip declaration.

s\_dcl DB sdat, 0

skip\_dcl:

LEA SI, [s\_dcl]

next\_char:

MOV AL, CS:[SI]

CMP AL, 0

JZ printed

INC SI

PUTC al

JMP next\_char

printed:

pop bx

POP SI ; re-store registers...

POP AX ;

ENDM

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

macro PUTC char

PUSH AX

MOV AL, char

MOV AH, 0Eh

INT 10h

POP AX

ENDM

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

; End Data definitions

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

codeseg

org 100h

ENTRY:

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

; Begin Instructions (Main)

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

call MainMenuHandler

call gameHandler

ret ; Return to O/S - Last instruction

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

; End Instructions (Main)

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

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

; Begin Functions

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

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

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

; Init Varaibles, ClearScreen and Clear Regs

; INPUT:

; NONE

; OUTPUT:

; ax = 0

; bx = 0

; cx = 0

; dx = 0

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

proc init

mov ah, 0

mov al, 13h

int 10h

call WhiteScreen

call RandomRange

mov dh, 0

mov [Col1Y], dx

mov [Col1X], ScreenWidth / 2 - 10

mov [Col2X], ScreenWidth - 10

mov [Score], 0

mov [PlayerX], ScreenWidth / 12

mov [PlayerY], ScreenHeight / 2 - 10;chosen randomly 10

mov [gameOver], 0

call RandomRange

mov dh, 0

mov [Col2Y], dx

xor ax, ax

xor bx, bx

xor cx, cx

xor dx, dx

ret

endp init

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

; Wait function until next run

; INPUT:

; ax = number of milisec\*1000 to wait until next run

; OUTPUT:

; NONE

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

proc msleep;

push ax

push cx

push dx

push bx

mov cx, 0

mov dx, ax

mov ah, 86h

int 15h

pop bx

pop dx

pop cx

pop ax

ret

endp msleep

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

; Draw a line in bx length

; INPUT:

; dx = current Y position

; cx = current X position

; bx = length of Line

; al = color of line

; OUTPUT:

; dx = current Y position

; cx = current X position + length of Line

; bx = length of Line

; al = color of line

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

proc drawLine

push dx

push bx

push ax

mov ah, 0Ch

@@drawLine\_loop1:

int 10h

dec bx

inc cx

cmp bx, 0

ja @@drawLine\_loop1

pop ax

pop bx

pop dx

ret

endp drawLine

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

; Draw a line in bx length

; INPUT:

; bx = Height and Width of the Squaresize

; cx = Current X Position

; dx = Current Y Position

; al = Color of Square

; OUTPUT:

; bx = Height and Width of the Squaresize

; cx = Current X Position + Width of the Square

; dx = Current Y Position

; al = Color of Square

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

proc drawSquare

push dx

push bx

push ax

mov [squareH], bl

@@drawSquareLoop1:

call drawLine

sub cx, bx

inc dx

dec [squareH]

mov ah, 0

cmp [squareH], ah

ja @@drawSquareLoop1

pop ax

pop bx

pop dx

ret

endp drawSquare

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

; Draw a line Of Squares

; INPUT:

; al = color

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawLineOfSquares

push ax

mov [numOfSquares], bl

@@drawLineOfSquaresloop1:

mov bx, Squaresize

call drawSquare

add cx, bx

dec [numOfSquares]

mov ah, 0

cmp [numOfSquares], ah

ja @@drawLineOfSquaresloop1

pop ax

ret

endp drawLineOfSquares

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

; Draw a line Of Black Squares

; INPUT:

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawBlackLineOfSquares

push ax

mov al, 00h

call drawLineOfSquares

pop ax

ret

endp drawBlackLineOfSquares

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

; Draw a line Of Orange Squares

; INPUT:

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawOrangeLineOfSquares

push ax

mov al, 2Bh

call drawLineOfSquares

pop ax

ret

endp drawOrangeLineOfSquares

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

; Draw a line Of Red Squares

; INPUT:

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawRedLineOfSquares

push ax

mov al, 4h

call drawLineOfSquares

pop ax

ret

endp drawRedLineOfSquares

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

; Draw a line Of Yellow Squares

; INPUT:

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawYellowLineOfSquares

push ax

mov al, 0Eh

call drawLineOfSquares

pop ax

ret

endp drawYellowLineOfSquares

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

; Draw a line Of White Squares

; INPUT:

; bx = Number Of Squares

; cx = Current X Position

; dx = Current Y Position

; OUTPUT:

; bx = Number Of Squares

; cx = Current X Position + Width of the Square

; dx = Current Y Position

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

proc drawWhiteLineOfSquares

push ax

mov al, 0Fh

call drawLineOfSquares

pop ax

ret

endp drawWhiteLineOfSquares

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

; Draw Bird Row 1

; INPUT:

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow1Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 6\*Squaresize

mov dx, [PlayerY]

mov bx, 6

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow1Bird

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

; Draw Bird Row 2

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow2Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 4\*Squaresize

mov bx, 2

call drawBlackLineOfSquares

mov bx, 3

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 2

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow2Bird

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

; Draw Bird Row 3

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow3Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 3\*Squaresize

mov bx, 1

call drawBlackLineOfSquares

mov bx, 4

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 4

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow3Bird

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

; Draw Bird Row 4

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow4Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, Squaresize

mov bx, 4

call drawBlackLineOfSquares

mov bx, 3

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 3

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 1

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow4Bird

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

; Draw Bird Row 5

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow5Bird

push ax

push bx

push cx

mov cx, [PlayerX]

mov bx, 1

call drawBlackLineOfSquares

mov bx, 4

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 2

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 3

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 1

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow5Bird

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

; Draw Bird Row 6

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow6Bird

push ax

push bx

push cx

mov cx, [PlayerX]

mov bx, 1

call drawBlackLineOfSquares

mov bx, 5

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 2

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 4

call drawWhiteLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow6Bird

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

; Draw Bird Row 7

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow7Bird

push ax

push bx

push cx

mov cx, [PlayerX]

mov bx, 1

call drawBlackLineOfSquares

mov bx, 1

call drawYellowLineOfSquares

mov bx, 3

call drawWhiteLineOfSquares

mov bx, 1

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 3

call drawYellowLineOfSquares

mov bx, 6

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow7Bird

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

; Draw Bird Row 8

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow8Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, Squaresize

mov bx, 1

call drawBlackLineOfSquares

mov bx, 3

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 3

call drawYellowLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 6

call drawRedLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow8Bird

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

; Draw Bird Row 9

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow9Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 2\*Squaresize

mov bx, 3

call drawBlackLineOfSquares

mov bx, 3

call drawOrangeLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 1

call drawRedLineOfSquares

mov bx, 6

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow9Bird

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

; Draw Bird Row 10

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow10Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 2\*Squaresize

mov bx, 1

call drawBlackLineOfSquares

mov bx, 6

call drawOrangeLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

mov bx, 5

call drawRedLineOfSquares

mov bx, 1

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow10Bird

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

; Draw Bird Row 11

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow11Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 3\*Squaresize

mov bx, 2

call drawBlackLineOfSquares

mov bx, 5

call drawOrangeLineOfSquares

mov bx, 5

call drawBlackLineOfSquares

add dx, Squaresize

pop cx

pop bx

pop ax

ret

endp drawRow11Bird

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

; Draw Bird Row 12

; INPUT:

; dx = Current Y Position

; OUTPUT:

; dx = Current Y Position + Squaresize

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

proc drawRow12Bird

push ax

push bx

push cx

mov cx, [PlayerX]

add cx, 4\*Squaresize

mov bx, 6

call drawBlackLineOfSquares

pop cx

pop bx

pop ax

ret

endp drawRow12Bird

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

; Draw Bird calls draw Bird Row Number

; INPUT:

; NONE

; OUTPUT:

; dx = Current Y + PlayerHeight

; cx = Current X + PlayerWidth

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

proc drawBird

call drawRow1Bird

call drawRow2Bird

call drawRow3Bird

call drawRow4Bird

call drawRow5Bird

call drawRow6Bird

call drawRow7Bird

call drawRow8Bird

call drawRow9Bird

call drawRow10Bird

call drawRow11Bird

call drawRow12Bird

ret

endp drawBird

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

; Draw Col

; INPUT:

; cx = Col's X position

; bx = Col's Y Hole Position

; OUTPUT:

; NONE

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

proc drawCol

push ax

push bx

push cx

push dx

mov [ColPlace], bx

mov dx, 0

drawColLoop:

cmp dx, [ColPlace]

jb drawColCondition

mov ax, ColHole

add ax, [ColPlace]

cmp dx, ax

ja drawColCondition

inc dx

cmp dx, ScreenHeight

jb drawColLoop

drawColCondition:

mov al, 11

mov bx, ColLength

call drawLine

sub cx, ColLength

inc dx

cmp dx, ScreenHeight - GrassHeight

jb drawColLoop

pop dx

pop cx

pop bx

pop ax

ret

endp drawCol

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

; Clears Error From Col going from one side to the other

; INPUT:

; ax = Col's 1 X Position

; bx = Col's 2 X Position

; OUTPUT:

; NONE

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

proc clearError

push ax

push bx

push cx

push dx

cmp ax, ScreenWidth - ColLength

ja @@Clear

cmp ax, ColLength

jb @@Clear

cmp bx, ScreenWidth - ColLength

ja @@Clear

cmp bx, ColLength

jb @@Clear

jmp @@notClear

@@Clear:

mov cx, 0

mov dx, 0

mov bx, [PlayerX]

dec bx

mov al, 0Fh

@@loop1:

call drawLine

sub cx, bx

inc dx

cmp dx, ScreenHeight - GrassHeight

jb @@loop1

@@notClear:

pop dx

pop cx

pop bx

pop ax

ret

endp clearError

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

; Draws Grass

; INPUT:

; al = Color Of Grass

; dx = From which until ScreenHeight to Draw Grass (ScreenHeight - GrassHeight)

; OUTPUT:

; NONE

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

proc drawGrass

push dx

push cx

push bx

push ax

drawGrassLoop:

mov cx, 0

mov bx, ScreenWidth

call drawLine

inc dx

cmp dx, ScreenHeight

jb drawGrassLoop

pop ax

pop bx

pop cx

pop dx

ret

endp drawGrass

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

; draw Score On Screen

; INPUT:

; ScoreHeight = Where to Print On Y

; ScoreWidth = Where to Print On X

; OUTPUT:

; NONE

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

proc drawScore

push ax

mov [cursorX], ScoreWidth

mov [cursorY], ScoreHeight

call cursorPose

mov ax, [Score]

call print\_num\_dec

pop ax

ret

endp drawScore

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

; draw Everything On Screen

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc draw

push ax

push bx

push cx

push dx

call clearScreen

mov dx, ScreenHeight - GrassHeight

mov al, 2;color green

call drawGrass

mov ax, [Col1X]

mov bx, [Col2X]

call ClearError

mov cx, [Col1X]

mov bx, [Col1Y]

call drawCol

mov cx, [Col2X]

mov bx, [Col2Y]

call drawCol

call drawBird

call drawScore

pop dx

pop cx

pop bx

pop ax

ret

endp draw

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

; clear Screen to White

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc WhiteScreen

push dx

push cx

push bx

push ax

mov cx, 0

mov dx, 0

mov al, 0Fh

mov ah, 0Ch

@@loop1:

int 10h

inc cx

cmp cx, ScreenWidth

jb @@loop1

mov cx, 0

inc dx

cmp dx, ScreenHeight

jb @@loop1

pop ax

pop bx

pop cx

pop dx

ret

endp WhiteScreen

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

; clear LeftOvers from last frame

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc clearScreen

push dx

push cx

push bx

push ax

mov cx, [Col1X]

call ClearCol

mov cx, [Col2X]

call ClearCol

call ClearBird

pop ax

pop bx

pop cx

pop dx

ret

endp clearScreen

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

; clear left Overs from last col's frame

; INPUT:

; cx = Col's X Position

; OUTPUT:

; NONE

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

proc clearCol

push dx

push cx

push bx

push ax

add cx, ColLength

mov al, 0Fh

mov ah, 0Ch

mov dx, 0

mov bx, ColSpeed

@@loop1:

int 10h

inc dx

cmp dx, ScreenHeight - GrassHeight

jb @@loop1

mov dx, 0

inc cx

dec bx

cmp bx, 0

ja @@loop1

pop ax

pop bx

pop cx

pop dx

ret

endp clearCol

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

; clear left Overs from last bird's frame

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc ClearBird

push dx

push cx

push bx

push ax

mov dx, [PlayerY]

cmp dx, VelocityJump

jb @@SetZero1

sub dx, VelocityJump

@@SetZero1:

cmp dx, ClearHeight

jb SetZero2

sub dx, ClearHeight

jmp SetZero3

SetZero2:

mov dx, 0

SetZero3:

mov al, 0Fh

mov ah, 0Ch

mov cx, [PlayerX]

@@loop1:

mov bx, PlayerWidth

call drawLine

sub cx, bx

inc dx

mov bx, [PlayerY]

add bx,PlayerHeight + VelocityFall + ClearHeight

cmp dx, bx

jb @@loop1

pop ax

pop bx

pop cx

pop dx

ret

endp ClearBird

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

; update Cols' Bird' check Colision and check for input

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc update

push dx

push cx

push bx

push ax

call input

call updatePlayerY

call updateCols

call checkColision

pop ax

pop bx

pop cx

pop dx

ret

endp update

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

; uCheck input if there is input set PlayerAdder to how many frames do u want to jump

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc input

push ax

mov ah,1h ;Check if any key was pressed in the keyboard!

int 16h

jnz @@input\_getKey;key was pressed!

jmp @@input\_end;key wasent pressed!

@@input\_getKey:

mov ah,0h;Get the key that was pressed!

int 16h

cmp al,' '

jne @@input\_end

mov [PlayerAdder], FramesToGoUp

@@input\_end:

pop ax

ret

endp input

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

; update Birds Y Position if touches sky leave it there if touches ground leave it there

; INPUT:

; PlayerY = current player Position

; OUTPUT:

; PlayerY = new player Y Position

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

proc updatePlayerY

push ax

mov ax, [PlayerAdder]

cmp ax, 1

ja @@jump

mov ax, [PlayerY]

cmp ax, ScreenHeight - GrassHeight - PlayerHeight

jnb @@end

cmp ax, ScreenHeight - GrassHeight - VelocityFall - PlayerHeight

ja @@bottom

add ax, VelocityFall

jmp @@end

@@bottom:

mov ax, ScreenHeight - GrassHeight - PlayerHeight

jmp @@end

@@jump:

dec [PlayerAdder]

mov ax, [PlayerY]

cmp ax, VelocityJump

jb @@top

sub ax, VelocityJump

jmp @@end

@@top:

mov ax, 1

@@end:

mov [PlayerY], ax

pop ax

ret

endp updatePlayerY

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

; update each col X Position' Y hole position and update Score if needed

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc updateCols

push ax

mov ax, [Col1X]

call updateColX

mov [Col1X], ax

mov ax, [Col1X]

call updateScore

mov ax, [Col1X]

mov bx, [Col1Y]

call CheckNewColY

mov [Col1Y], bx

mov ax, [Col2X]

call updateColX

mov [Col2X], ax

mov ax, [Col2X]

call updateScore

mov ax, [Col2X]

mov bx, [Col2Y]

call CheckNewColY

mov [Col2Y], bx

pop ax

ret

endp updateCols

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

; update Col's X position

; INPUT:

; ax = Col's X position

; OUTPUT:

; ax = new X Col's Position

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

proc updateColX

cmp ax, ColSpeed

ja @@above

add ax, ScreenWidth

jmp @@end

@@above:

sub ax, ColSpeed

jmp @@end

@@end:

ret

endp updateColX

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

; update Col's Y position

; INPUT:

; ax = Col's X position

; bx = Col's y position

; OUTPUT:

; bx = new Col's y position

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

proc CheckNewColY

push ax

push dx

cmp ax, ScreenWidth - ColLength

jne @@end

call ClearHole

call RandomRange

mov bx, dx

@@end:

pop dx

pop ax

ret

endp CheckNewColY

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

; clear Hole If neede to create a new one

; INPUT:

; ax = Col's X Position

; bx = Col's Y position

; OUTPUT:

; NONE

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

proc ClearHole; bx Col1Y ax Col1X

push dx

push cx

push bx

push ax

mov dx, ColHole

mov [HoleCleaner],dx

mov dx, bx

mov cx, ax

mov bx, ColLength + 1

mov al, 11

@@loop1:

call drawLine

dec [HoleCleaner]

sub cx, ColLength + 1

inc dx

cmp [HoleCleaner], 0

ja @@loop1

pop ax

pop bx

pop cx

pop dx

ret

endp ClearHole

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

; Generate a random number in range

; INPUT:

; NONE

; OUTPUT:

; dx = Random number between 0 to ScreenHeight - ColHole - GrassHeight - 1

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

proc Random

push ax

push cx

mov ah, 00h

int 1AH

mov ax, dx

xor dx, dx

mov cx, ScreenHeight - ColHole - GrassHeight

div cx

pop cx

pop ax

ret

endp Random

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

; check Colision with ground and Cols

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc checkColision

push ax

push bx

push cx

push dx

mov ax, ScreenHeight - GrassHeight - PlayerHeight

cmp [PlayerY], ax

jb @@notcolide

mov [gameOver], 1

@@notcolide:

call checkColsColision

pop dx

pop cx

pop bx

pop ax

ret

endp checkColision

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

; check Colision with Cols

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc checkColsColision

push ax

push bx

push cx

push dx

mov cx, [Col1X]

mov dx, [Col1Y]

call checkColBirdColision

mov cx, [Col2X]

mov dx, [Col2Y]

call checkColBirdColision

pop dx

pop cx

pop bx

pop ax

ret

endp checkColsColision

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

; update Col Bird Colision

; INPUT:

; cx = Col's X position

; dx = Col's Y position

; OUTPUT:

; if colide set game over 1

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

proc checkColBirdColision;

push ax

push cx

mov ax, [PlayerX]

add ax, PlayerWidth

cmp cx, ax

ja @@notColide

add cx, ColLength

sub ax, PlayerWidth

cmp cx, ax

jb @@notColide

mov ax, [PlayerY]

cmp ax, dx

jb @@colide

add dx, ColHole

add ax, PlayerHeight

cmp ax, dx

ja @@colide

@@notColide:

jmp @@end

@@colide:

mov [gameOver], 1

@@end:

pop cx

pop ax

ret

endp checkColBirdColision

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

; check Random Range

; INPUT:

; NONE

; OUTPUT:

; dl = random number in range

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

proc RandomRange; 0<=r<=200-70-5-10

push bx

push ax

@@loop1:

call Random

mov bx, dx

cmp bx, ScreenHeight-ColHole-GrassHeight-10

ja @@loop1

pop ax

pop bx

ret

endp RandomRange

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

; update [Score] if needed

; INPUT:

; ax = COl's X position

; OUTPUT:

; [Score]++ if PlayerX = ColX

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

proc updateScore; ax has X of Col.

push ax

push bx

push cx

push dx

add ax, ColLength

cmp ax, [PlayerX]

jne @@end

inc [Score]

@@end:

pop dx

pop cx

pop bx

pop ax

ret

endp updateScore

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

; Set cursor place to type

; INPUT:

; [cursorY] = Y Height to type

; [cursorX] = X Height to type

; OUTPUT:

; NONE

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

proc cursorPose

push ax

push bx

push dx

mov dh, [cursorY]

mov dl, [cursorX]

mov bh, 0

mov ah, 2

int 10h

pop dx

pop bx

pop ax

ret

endp cursorPose

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

; main menu handeling

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc MainMenuHandler

call MainMenuinit

@@loop1:

call MainMenu

mov bx, 3

@@loop2:

mov ax, 50\*1000

call msleep

dec bx

cmp bx, 0

ja @@loop2

mov ax, 0

cmp ax, [gameStartLoop]

jz @@loop1

call hidemouse

call WhiteScreen

ret

endp MainMenuHandler

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

; turn off the cursor

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc cursorOff

PUSH AX

PUSH CX

MOV AH, 1

MOV CH, 28h

MOV CL, 09h

INT 10h

POP CX

POP AX

ret

endp cursorOff

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

; turn on the cursor

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc cursorOn

PUSH AX

PUSH CX

MOV AH, 1

MOV CH, 08h

MOV CL, 09h

INT 10h

POP CX

POP AX

ret

endp cursorOn

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

; main menu functions

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc MainMenu

call hidemouse

call clearMainError

call showmouse

call MainMenuinput

ret

endp MainMenu

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

; clear vars of main menu to work better and clear screen

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc MainMenuinit

mov ah, 00

mov al, 13h

int 10h

xor ax, ax

int 33h

call showmouse

call WhiteScreen

mov [ButtonColor], 00h

mov [gameStartLoop], 0

call hidemouse

call DrawMainMenu

call showmouse

ret

endp MainMenuinit

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

; draw main menu

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc DrawMainMenu

mov cx, MainMenuButton1X

mov dx, MainMenuButton1Y

call DrawButton

mov cx, MainMenuButton2X

mov dx, MainMenuButton2Y

call DrawButton

call DrawText

ret

endp DrawMainMenu

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

; clear the error the mouse creates

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc clearMainError

mov cx, ScreenWidth / 2 - ErrorCleaner

mov bx, 2\*ErrorCleaner

mov dx, ScreenHeight / 2 - ErrorCleaner

mov al, 0Fh

@@loop1:

call drawLine

inc dx

sub cx, bx

cmp dx, ScreenHeight / 2 + ErrorCleaner

jb @@loop1

ret

endp clearMainError

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

; draw button in X Y

; INPUT:

; cx = X buttton

; dx = Y button

; OUTPUT:

; NONE

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

proc DrawButton

push ax

push bx

push cx

push dx

mov [ButtonAdder], ButtonHeight

@@loop1:

mov al, [ButtonColor]

mov bx, ButtonLength

call DrawLine

sub cx, bx

inc dx

dec [ButtonAdder]

cmp [ButtonAdder], 0

ja @@loop1

pop dx

pop cx

pop bx

pop ax

ret

endp DrawButton

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

; draw text in PreSet Positions

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc DrawText

mov [TextXInput], firstTextX

mov [TextYInput], firstTextY

call gotoxy

mov bl, firstStringColor

print 'start'

mov [TextXInput], secondTextX

mov [TextYInput], secondTextY

call gotoxy

mov bl, secondStringColor

print 'quit'

ret

endp DrawText

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

; Set cursor to type place to type

; INPUT:

; [TextYInput] = Y Height to type

; [TextXInput] = X Height to type

; OUTPUT:

; NONE

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

proc gotoxy

push ax

push bx

push cx

push dx

mov dl, [TextXInput]

mov dh, [TextYInput]

mov ah, 2 ;SERVICE TO SET CURSOR POSITION.

mov bh, 0 ;PAGE.

int 10h ;BIOS SCREEN SERVICES.

pop dx

pop cx

pop bx

pop ax

ret

endp gotoxy

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

; Handle Click Input and check react to which button was pressed

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc MainMenuinput

mov [pressed], 0

push ax

push bx

push cx

push dx

mov ax, 5

mov bx, 0

int 33h

and ax, 1

cmp ax, 1; if left is pressed

jnz @@end

mov ax, 3

int 33h

shr cx, 1

mov ax, MainMenuButton1X

mov bx, MainMenuButton1Y

call CheckButtonColision

cmp [pressed], 1

jne @@Check2Button

mov [gameStartLoop], 1

jmp @@end

@@Check2Button:

mov ax, MainMenuButton2X

mov bx, MainMenuButton2Y

call CheckButtonColision

cmp [pressed], 1

jne @@end

mov ax, 0

int 21h

@@end:

pop dx

pop cx

pop bx

pop ax

ret

endp MainMenuinput

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

;check if pressed the button

; input ax = Button X

; bx = Button Y

; cx = Mouse Preesed X

; dx = Mouse Preesed Y

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

proc CheckButtonColision

push ax

push bx

push cx

push dx

cmp cx, ax

jb @@end

add ax, ButtonLength

cmp cx, ax

ja @@end

cmp dx, bx

jb @@end

add bx, ButtonHeight

cmp dx, bx

ja @@end

mov [pressed], 1

@@end:

pop dx

pop cx

pop bx

pop ax

ret

endp CheckButtonColision

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

; hide mouse

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc hidemouse

push ax

mov ax, 2

int 33h

pop ax

ret

endp hidemouse

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

; show mouse

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc showmouse

push ax

mov ax, 1

int 33h

pop ax

ret

endp showmouse

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

; handle all game loop

; INPUT:

; NONE

; OUTPUT:

; NONE

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

proc gameHandler

call init

@@gameloop:

call update

call draw

mov ax, SleepTimes

call msleep

mov ax, [gameOver]

cmp ax, 0

je @@gameloop

ret

endp gameHandler

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

; End Functions

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

include "c:\gvahim\gvahim.asm"

end ENTRY