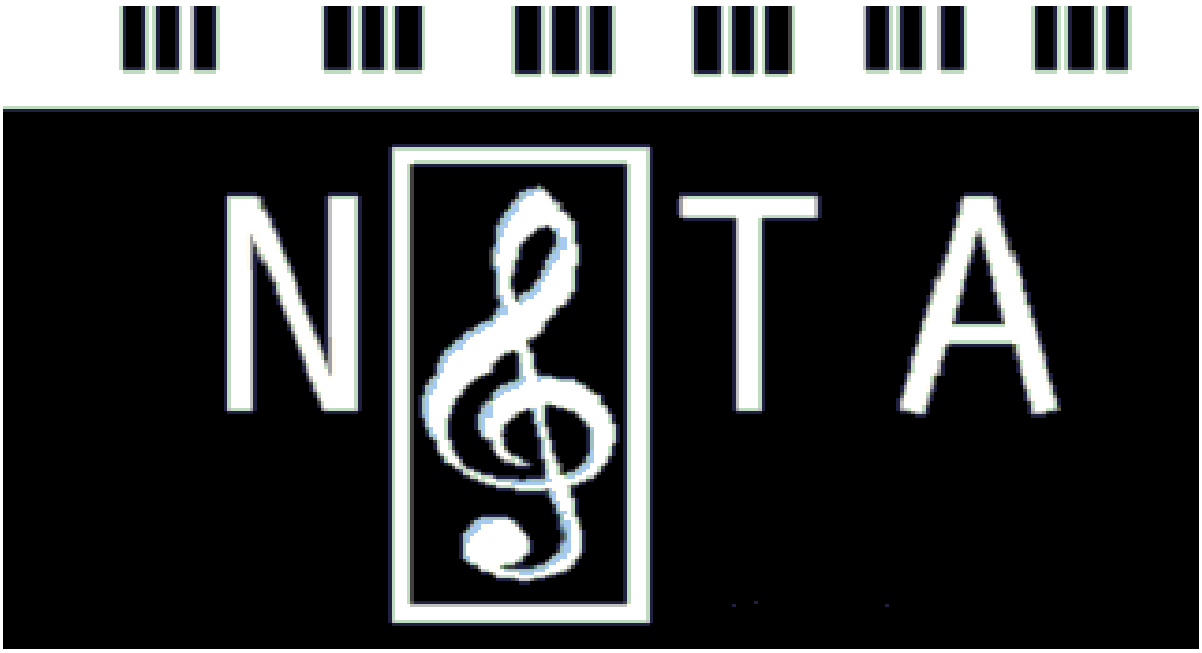


CMSC131 Introduction to Computer Organization and Machine-level Programming

CAPSTONE PROJECT

DANNY BOY NOYNAY

JEZZA VIÑALON

BS Computer Science III

Table of Contents

q. Procmis

r. Procfas

s. Procsols

t. Proclas

u. Procsis

v. Sound2

w. Highscore

x. Replace

y. Exitproc

3. Screen cap of the game

Project Summary

Nota is an assembly language programming game that uses TASM (Turbo Assembler) and DOSBox x86 emulator.

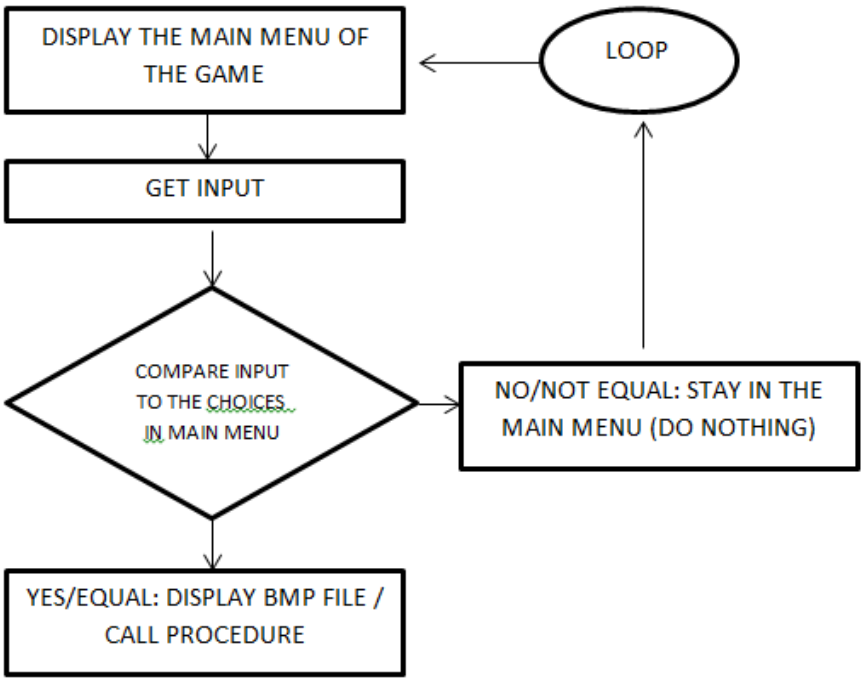
It is a single player game. The player can choose either to play freely the keyboard (K - Keyboard) or to play with a given mission (P - Play). The mission of the Player is to memorize the set of keys played before the actual game. Then the player is tasked to input the set of keys in the keyboard. Once the player input a wrong key, the game will be over.

Github Link: https://github.com/jizzaaaaa/Nota.git

Procedures

Open1 (for Main Menu)

- Definition
    - Display the main menu of the game.
    - Display the choices in the menu where the user wants to go.
        - (K) Keyboard
        - (P) Play
        - (I) How to play
        - (S) High score
        - (X) Exit
    - Get the user's inputted letter of choice in the menu and call its corresponding bmp and procedures.
- Flow Chart



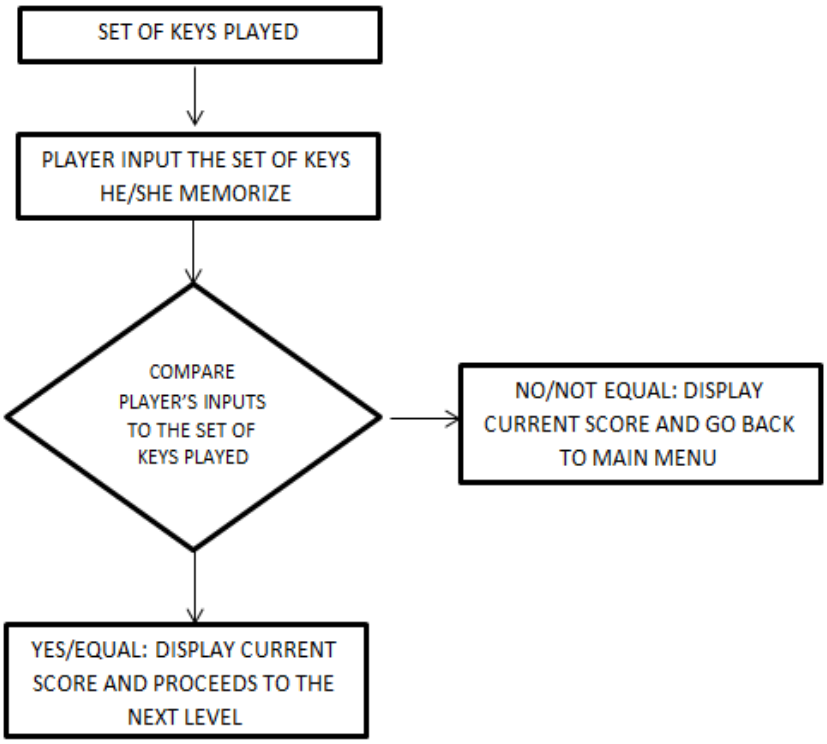
- Source Code

```
PROC OPEN1

        MOV     DX, offset open
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        MOV     DX, offset open10
        CALL    OpenShowBmp
        CALL    Timer
        MOV     DX, offset open11
        CALL    OpenShowBmp
        CALL    Timer
        MOV     DX, offset open12
        CALL    OpenShowBmp
        CALL    Timer
        MOV     DX, offset open13
        CALL    OpenShowBmp
        CALL    Timer
inp:
        PUSHA
        POPA
        MOV     DX, offset open14
        CALL    OpenShowBmp
;get input
        MOV     AH, 7
        INT     21H

choose1:
        CMP     AL, 'k'



        JE      keyboard
        JNE     choose2

        choose2:
        CMP     AL, 'p'
        JE      songcomplete
        JNE     choose3

        choose3:
        CMP     AL, 'i'
        JNE     choose4

        inp3:
                MOV     DX, offset howto
                CALL    OpenShowBmp
                CALL    Timer
                MOV     DX, offset howto1
                CALL    OpenShowBmp
                CALL    Timer
                MOV     DX, offset howto
                CALL    OpenShowBmp
                CALL    Timer
                MOV     DX, offset howto1
                CALL    OpenShowBmp
                CALL    Timer

                ;get input
```

```
                    MOV     AH, 7
                    INT     21H
                    CMP     AL, 'b'
                    JE      creditss
                    JNE     inp3
choose4:
        CMP     AL, 's'
        JNE     choose5
        inp5:
                CALL    highscore
                ;get input
                MOV     AH, 7
                INT     21H
                CMP     AL, 'b'
                JE      creditss
                JNE     inp5
choose5:
        CMP     AL, 'c'
        JNE     choose6
        inp2:
                MOV     DX, offset credits
                CALL    OpenShowBmp
                ;get input
                        MOV     AH, 7
                        INT     21H
                        CMP     AL, 'b'
                        JE      creditss
                        JNE     inp2
choose6:
        CMP     AL, 'x'
        JE      exit2
        JNE     inp

songcomplete:
        CALL    game1

keyboard:
        CALL    scp

creditss:
        CALL    OPEN1

exit2:
        CALL    exitproc

RET
ENDP OPEN
```

Game1 to Game5 (Level 1 - 5)

- Definition
    - Call and play a set of keys designed to follow in each level (Player should memorize the played keys).
    - Input each key.
    - Compares each key inputted to the designed key in order
    - Once an inputted key was compared and the result is not equal to the designed key, it automatically jumps to exit, display the current score and return to the main menu.
- Flow Chart



- Source Code (Game1)

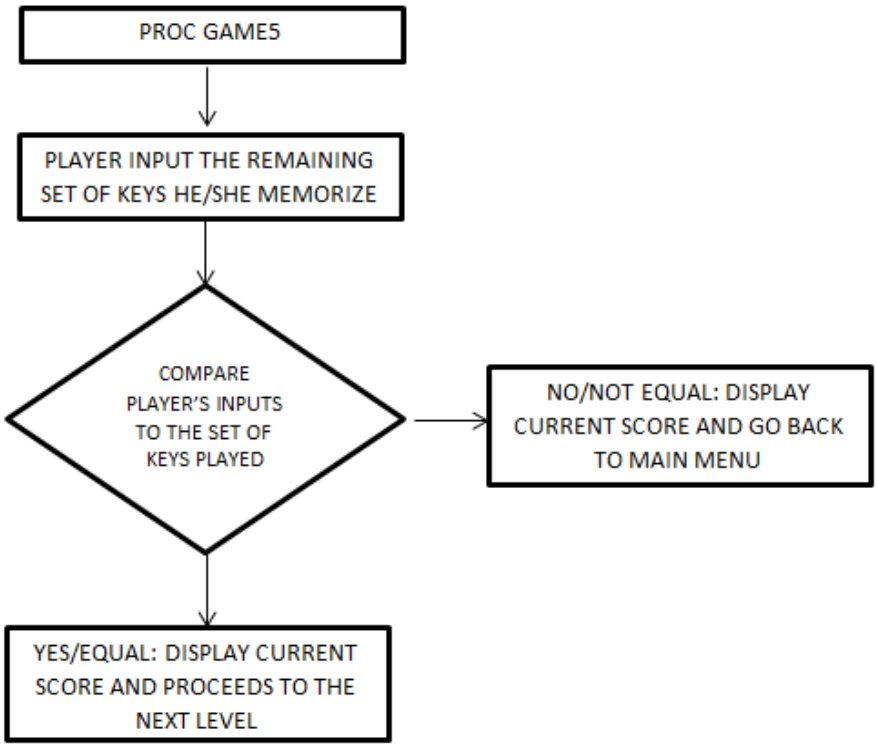
```
PROC game1

        MOV     DX, offset ready1
        CALL    OpenShowBmp
        CALL    Timer
        MOV     DX, offset ready2
        CALL    OpenShowBmp
        CALL    Timer
```

```
        MOV     DX, offset ready3
        CALL    OpenShowBmp
        CALL    Timer

        MOV     [VAR], 0

;LEVEL1PLAY
        MOV     DX, offset ready
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        CALL    Timer
        ;play level 1
                CALL    procdos   ;1
                CALL    procmis ;3
                CALL    procdos ;1
                CALL    procfas ;4
        MOV     DX, offset go
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        CALL    Timer
        MOV     DX, offset default2             ;default picture until game starts for piano
        CALL    OpenShowBmp
                ;game
                        MOV     AH, 7           ;----------input 1
                        INT     21H
                        CMP     AL, '1'
                        JNE     ExitLevel1
                        CALL    procdos
                        MOV     AH, 7           ;=----------input 2
                        INT     21H
                        CMP     AL, '3'
                        JNE     ExitLevel1
                        CALL    procmis
                        MOV     AH, 7           ;=----------input 3
                        INT     21H
                        CMP     AL, '1'
                        JNE     ExitLevel1
                        CALL    procdos
                        MOV     AH, 7           ;=----------input 4
                        INT     21H
                        CMP     AL, '4'
                        JNE     ExitLevel1
                        CALL    procfas

                MOV     DX, offset exact1
                CALL    OpenShowBmp
                CALL    Timer
                CALL    Timer
                CALL    Timer
                MOV     DX, offset level2
                CALL    OpenShowBmp
                CALL    Timer
                CALL    Timer
                CALL    Timer
                CALL    Timer

CALL    game2

ExitLevel1:
```

```
        MOV     DX, offset fail1
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        CALL    Timer
        CALL    Timer
        CALL    OPEN1


    RET
    ENDP game1
```

Game5a

- Definition
    - Continuation of Game5 (Game 5 is more complex than other game/level)
    - Since a procedure is limited with a number of jumps, game5a is used.
- Flow Chart



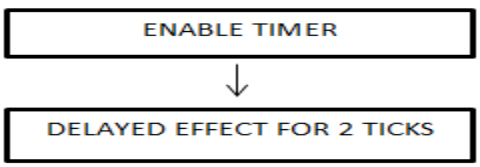
- Source Code

```
PROC game5a

    MOV     AH, 7               ;=----------input 6
    INT     21H
    CMP     AL, '6'
    JNE     ExitLevel6
```

```
        CALL    proclas
        MOV     AH, 7               ;=----------input 7
        INT     21H
        CMP     AL, '5'
        JNE     ExitLevel6
        CALL    procsols
        MOV     AH, 7               ;=----------input 8
        INT     21H
        CMP     AL, '6'
        JNE     ExitLevel6
        CALL    proclas
        MOV     AH, 7               ;=----------input 9
        INT     21H
        CMP     AL, '4'
        JNE     ExitLevel6
        CALL    procfas


        MOV     DX, offset exact5
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        MOV     DX, offset petmalu
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        CALL    Timer
        CALL    Timer

        CALL    OPEN1

        ExitLevel6:
        MOV     DX, offset fail5
        CALL    OpenShowBmp
        CALL    Timer
        CALL    Timer
        CALL    Timer
        CALL    Timer
        CALL    OPEN1

    RET
    ENDP game5a
```

Timer

- Definition
    - Set timer in giving delayed effect (2 ticks) on displaying bmp pictures or performing operations.
- Flowchart

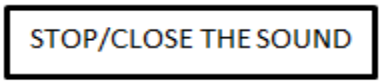


- Source Code

```
PROC Timer   ;timer with 2 ticks
      PUSHA
      MOV      AX, 40H              ;enable Timer
      MOV      ES, AX
      MOV      AX, [clock]
      FirstTick:
                CMP      AX, [clock]
                MOV      CX, 6      ;ticks
                JE       FirstTick
      DelayLoop:
                MOV      AX, [clock]
      Tick:
                CMP      AX, [clock]
                JE       Tick
                LOOP DelayLoop
      POPA
RET
ENDP Timer
```

Soundclose

- Definition
    - For sound to close.
- Flowchart



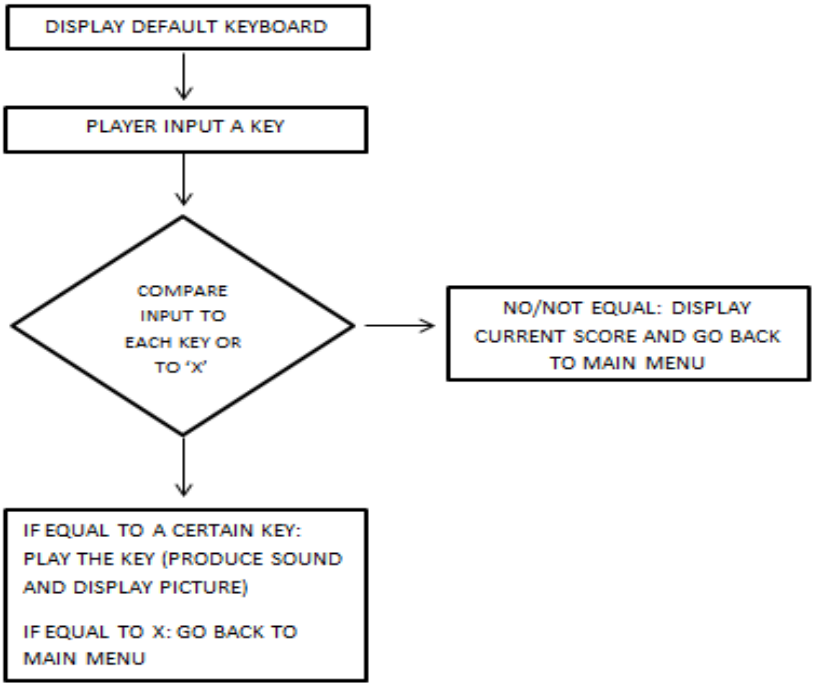
- Source Code

```
PROC soundclose        ;soundclose
      IN       AL, 61H
      AND      AL, 11111100B
      OUT      61H, AL
RET
ENDP soundclose
```

Scp

- Definition
    - Used when choosing (K) Keyboard.
    - Compares user's input to each key and if a key is equal to the user's input, it calls the procedure of the key (eg. procdos, procres,etc.), where it produces its sound and displays a picture where the key was pressed.

- Flowchart



- Source Code

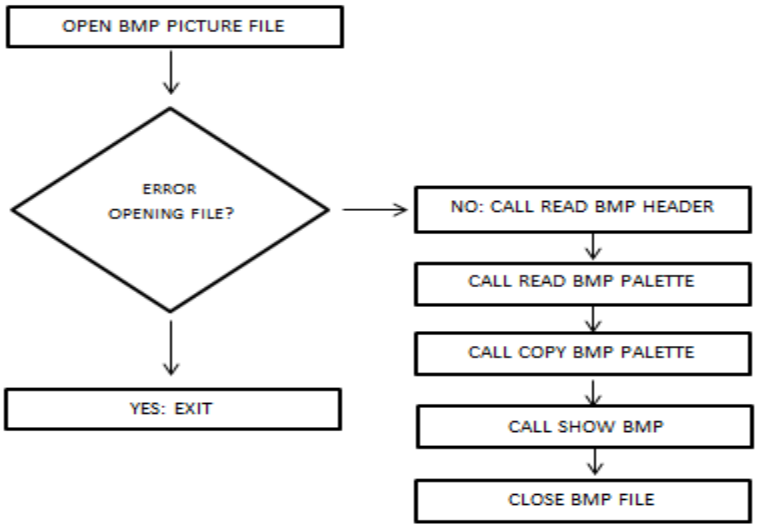
```
PROC scp

        MOV     DX, offset default2     ;default picture until game starts for piano
        CALL    OpenShowBmp

        scpstart:
                PUSHA
                POPA
                MOV     AH, 7           ;the input of the user which makes noise
                INT     21H

        CMP     AL, '1'                 ;do
        JNE     g
        CALL    procdos
        JMP     continue
        ;JL     scpstart
g:
        CMP     AL, '2'                 ;re
        JNE     g2
        CALL    procres
        JMP     continue
g2:
        CMP     AL, '3'                 ;mi
        JNE     g3
        CALL    procmis
        JMP     continue
g3:
        CMP     AL, '4'                 ;fa
        JNE     g4
        CALL    procfas
```

```
        JMP     continue
g4:
        CMP     AL, '5'                 ;sol
        JNE     g5
        CALL    procsols
        JMP     continue
g5:
        CMP     AL, '6'                 ;la3
        JNE     g6
        CALL    proclas
        JMP     continue
g6:
        CMP     AL, '7'                 ;ti
        JNE     exit3                   ;if the user got it wrong, he will be teleported back up to try again
without SI increasing
        CALL    procsis
                JMP     continue

        exit3:
                CMP     AL, 'x'
                JNE     scpstart
                CALL    OPEN1

        continue:
                LOOP    scpstart        ;DO IT ALL OVER AGAIN UNTIL HE FINISHES THE SONG
        RET
        ENDP scp
```

OpenShowBmp

- Definition
    - Displays the bmp picture.
    - Contains the bmp file handling procedures.
    - It calls the OpenBmpFile procedure then check if there's no error in opening the file. Then if there's no error, it calls the remaining procedures to display the bmp picture.

- Flowchart



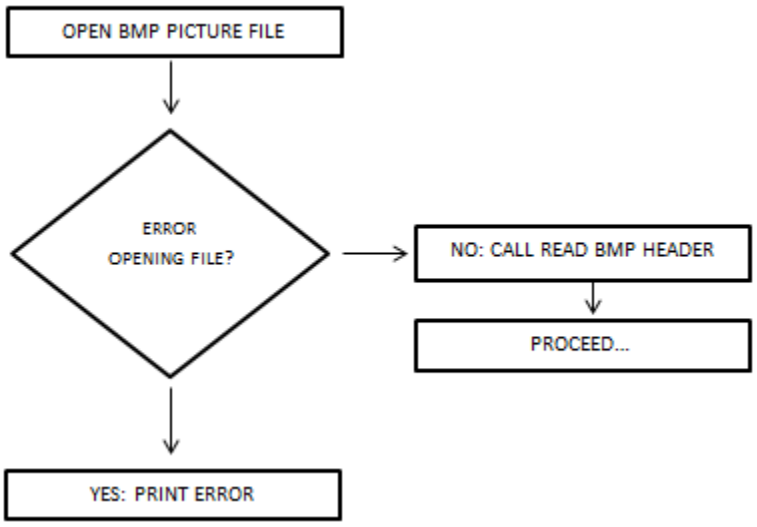
- Source Code

```
PROC OpenShowBmp NEAR
        PUSH    CX
        PUSH    BX
        CALL    OpenBmpFile
        CMP     [ErrorFile], 1
        JE              @@ExitProc
        CALL    ReadBmpHeader
        CALL    ReadBmpPalette
        CALL    CopyBmpPalette
        CALL    ShowBMP
        CALL    CloseBmpFile

        @@ExitProc:
                POP     BX
                POP     CX
RET
ENDP OpenShowBmp
```

OpenBmpFile

- Definition
    - Opens the bmp file then check if bmp file have no error/s on opening it.

- Flowchart

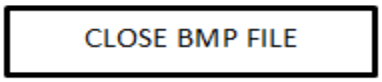


- Source Code

```
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
```

CloseBmpFile

- Definition
    - Close bmp file.
- Flowchart



- Source code

```
PROC CloseBmpFile NEAR
    MOV     AH, 3EH
    MOV     BX, [FileHandle]
    INT     21H
RET
ENDP CloseBmpFile
```

ReadBmpHeader

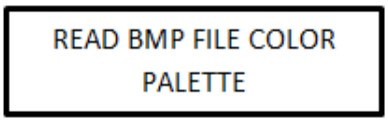
- Definition
    - Read BMP file header, 54 bytes
- Flowchart

- Source code

```
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
```

ReadBmpPalette

- Definition
    - Read BMP file color palette, 256 colors * 4 bytes (400h)
- Flowchart



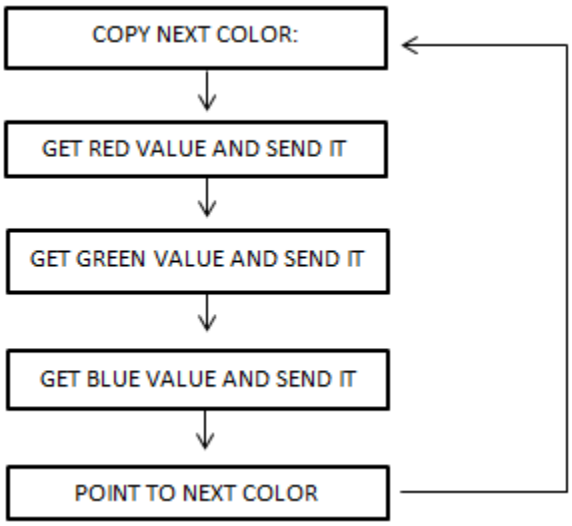
- Source code

```
PROC ReadBmpPalette near
    PUSH CX
    PUSH DX
    MOV AH, 3fh
    MOV CX,400h
    MOV DX, offset Palette
    INT 21H
```

```
        POP DX
        POP CX
    RET
    ENDP ReadBmpPalette
```

CopyBmpPalette

- Definition
    - Copy the colors palette to the video memory
- Flowchart

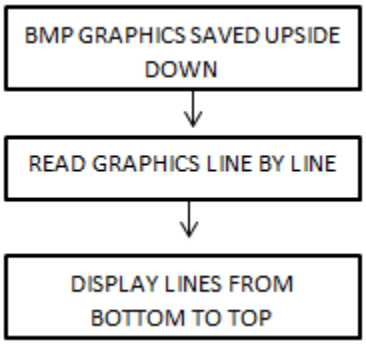


- Source code

```
PROC CopyBmpPalette  NEAR
        PUSH    CX
        PUSH    DX
        MOV     SI, offset Palette
        MOV     CX, 256
        MOV     DX, 3C8H
        MOV     AL, 0
        OUT     DX, AL
        INC     DX
        CopyNextColor:                  ;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 maxima.
                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
                LOOP    CopyNextColor
                POP     DX
                POP CX
    RET
    ENDP CopyBmpPalette
```

ShowBMP

- Definition
  - Transform and show BMP File
- Flowchart



- Source code

```
PROC ShowBMP
    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

        MOV     AH, 3fH         ; small read one line
        MOV     CX, [BmpColSize]
        ADD     CX, BP          ;extra  bytes to each row must be divided by 4
        MOV     DX, offset ScreenLineMax
        INT     21H
        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
```

SetGraphic

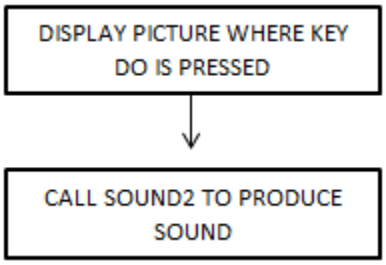
- Definition
  - For graphic mode.
- Flowchart

- Source code

```
PROC SetGraphic
        MOV     AX, 13H         ; 320 X 200
        INT     10H
RET
ENDP SetGraphic
```

Procdos

- Definition
  - Displays a picture where the key do was pressed and produces note1 (023A1h) sound.
- Flowchart

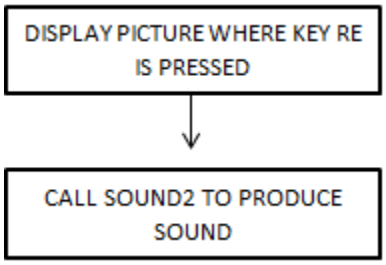


- Source code

```
PROC procdos
        PUSHA
        MOV     DX, offset dos
        CALL    OpenShowBmp
        MOV     AX, [note1]
        MOV     [note], AX
        CALL    sound2
        POPA
RET
ENDP procdos
```

Procres

- Definition
    - Displays a picture where the key re was pressed and produces note2 (01FBEh) sound.
- Flowchart

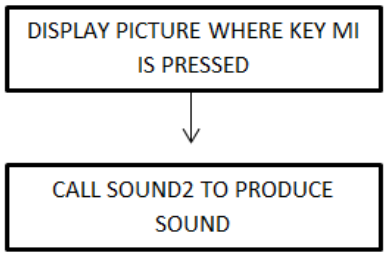


- Source code

```
PROC procres
        PUSHA
        MOV     DX, offset res
        CALL    OpenShowBmp
        MOV     AX, [note2]
        MOV     [note], ax
        CALL    sound2
        POPA
RET
ENDP procres
```

Procmis

- Definition
    - Displays a picture where the key mi was pressed and produces note3 (01C47h) sound.
- Flowchart

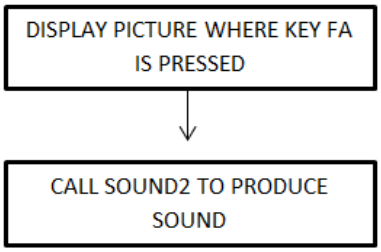


- Source code

```
PROC procmis
        PUSHA
        MOV     DX, offset mis
        CALL    OpenShowBmp
        MOV     AX, [note3]
        MOV     [note], AX
        CALL    sound2
        POPA
RET
ENDP procmis
```

Procfas

- Definition
    - Displays a picture where the key fa was pressed and produces note4 (01AB1h) sound.
- Flowchart



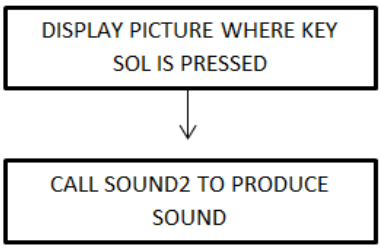
- Source code

```
PROC procfas
        PUSHA
        MOV     DX, offset fas
        CALL    OpenShowBmp
        MOV     AX, [note4]
        MOV     [note],     AX
        CALL    sound2
        POPA
RET
ENDP procfas
```

Procsols

- Definition
    - Displays a picture where the key so was pressed and produces note5 (017C7h) sound.

- Flowchart

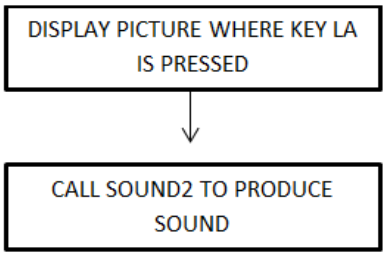


- Source code

```
PROC procsols
        PUSHA
        MOV     DX, offset sols
        CALL    OpenShowBmp
        MOV     AX, [note5]
        MOV     [note],AX
        CALL    sound2
        POPA
RET
ENDP procsols
```

Proclas

- Definition
    - Displays a picture where the key la was pressed and produces note6 (0152Fh) sound.
- Flowchart

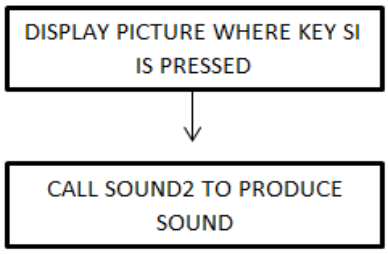


- Source code

```
PROC proclas
        PUSHA
        MOV     DX, offset las
        CALL    OpenShowBmp
        MOV     AX, [note6]
        MOV     [note], AX
        CALL    sound2
        POPA
RET
ENDP proclas
```

Procsis

- Definition
    - Displays a picture where the key si was pressed and produces note7 (012DFh) sound.
- Flowchart

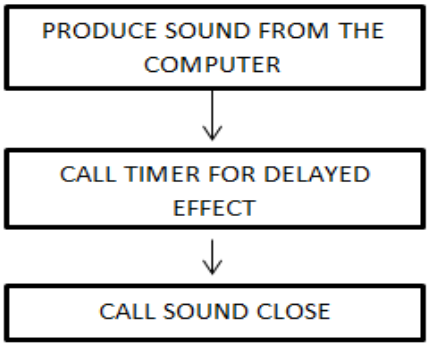


- Source code

```
PROC procsis
        PUSHA
        MOV     DX, offset sis
        CALL    OpenShowBmp
        MOV     AX, [note7]
        MOV     [note], AX
        CALL    sound2
        POPA
RET
ENDP procsis
```

Sound2

- Definition
    - Generating the sound
- Flowchart

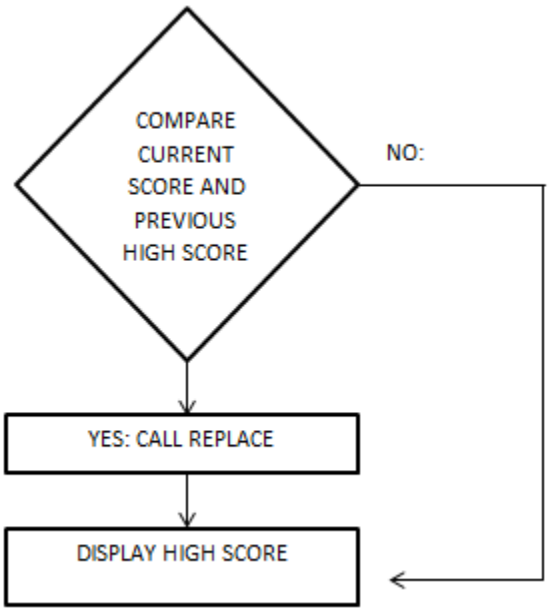


- Source Code

```
PROC sound2
        PUSHA
        MOV     BP, SP
        IN      AL, 61H
        OR      AL, 00000011B
        OUT     61H, AL
        MOV     AL, 0B6H
        OUT     43H, AL
        MOV     AX, [note]
        OUT     42H, AL
        MOV     AL, AH
        OUT     42H, AL
        CALL    Timer
        CALL    soundclose
        MOV     DX, offset default2
        CALL    OpenShowBmp
        POPA
RET
ENDP sound2
```

Highscore

- Definition
    - Compare current score and the previous high score. If current score is greater, call replace procedure.
- FlowChart

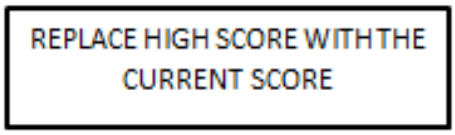


- Source Code

```
PROC highscore NEAR
MOV AL, [VAR]
CMP AL, [HIGHVAR]
JG toreplace

highss:
        CMP     [HIGHVAR], 0
        JE zeroscore

        CMP     [HIGHVAR], 1
        JE onescore

        CMP     [HIGHVAR], 2
        JE twoscore

        CMP     [HIGHVAR], 3
        JE threescore

        CMP     [HIGHVAR], 4
        JE fourscore

        CMP     [HIGHVAR], 5
        JE fivescore

zeroscore:
        MOV     DX, offset score0
        CALL    OpenShowBmp
        RET
onescore:
        MOV     DX, offset score1
        CALL    OpenShowBmp
        RET
twoscore:
        MOV     DX, offset score2
        CALL    OpenShowBmp
        RET
threescore:
        MOV     DX, offset score3
        CALL    OpenShowBmp
        RET
fourscore:
        MOV     DX, offset score4
        CALL    OpenShowBmp
        RET
fivescore:
        MOV     DX, offset score5
        CALL    OpenShowBmp
        RET

toreplace:
        CALL replace

RET
ENDP highscore
```

Replace

- Definition
  - Replace highscore with the current score.
- Flowchart



- Source Code

```
PROC replace NEAR
        MOV AL, [HIGHVAR]
        MOV AL, [VAR]
        MOV [HIGHVAR], AL
        RET
ENDP replace
```

Exitproc

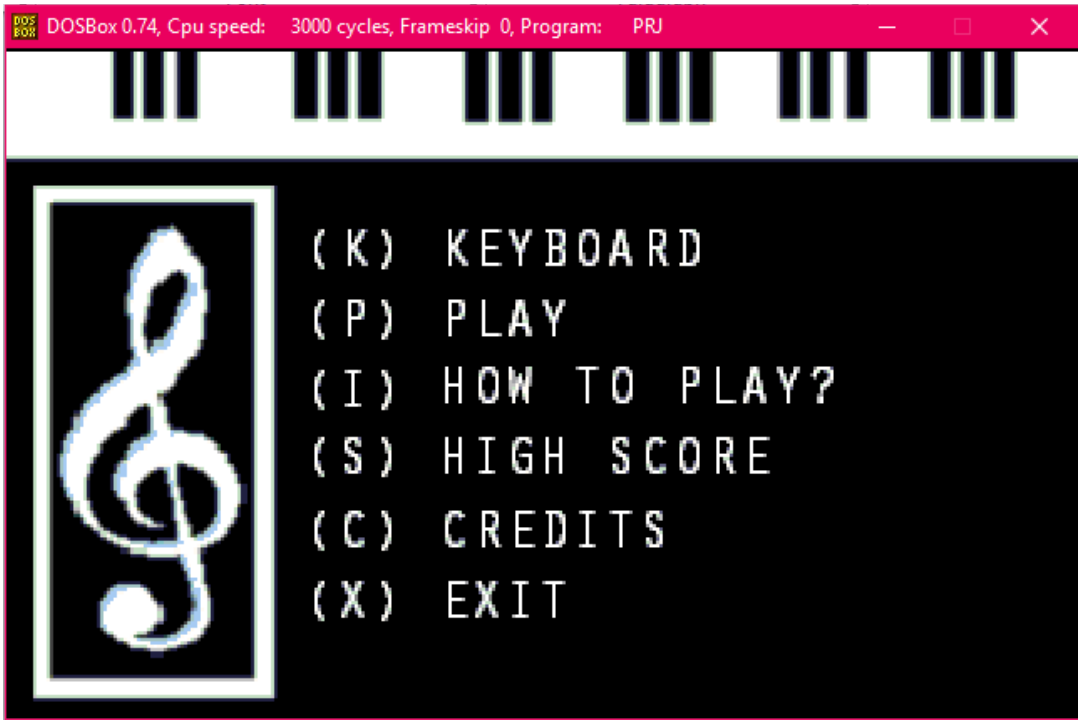
- Definition
  - Exit game
- Flowchart

- Source Code

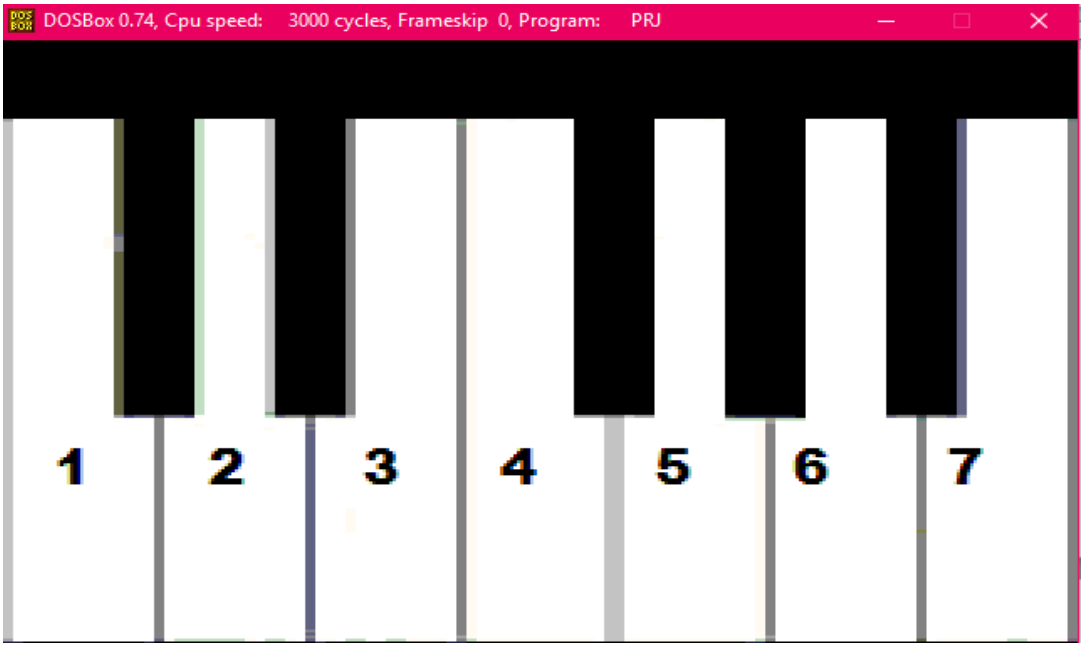
```
PROC exitproc
        MOV     AX, 2
        INT     10H
        MOV     AX, 4C00H
        INT     21H
ENDP exitproc
```

Screen Cap of the Game

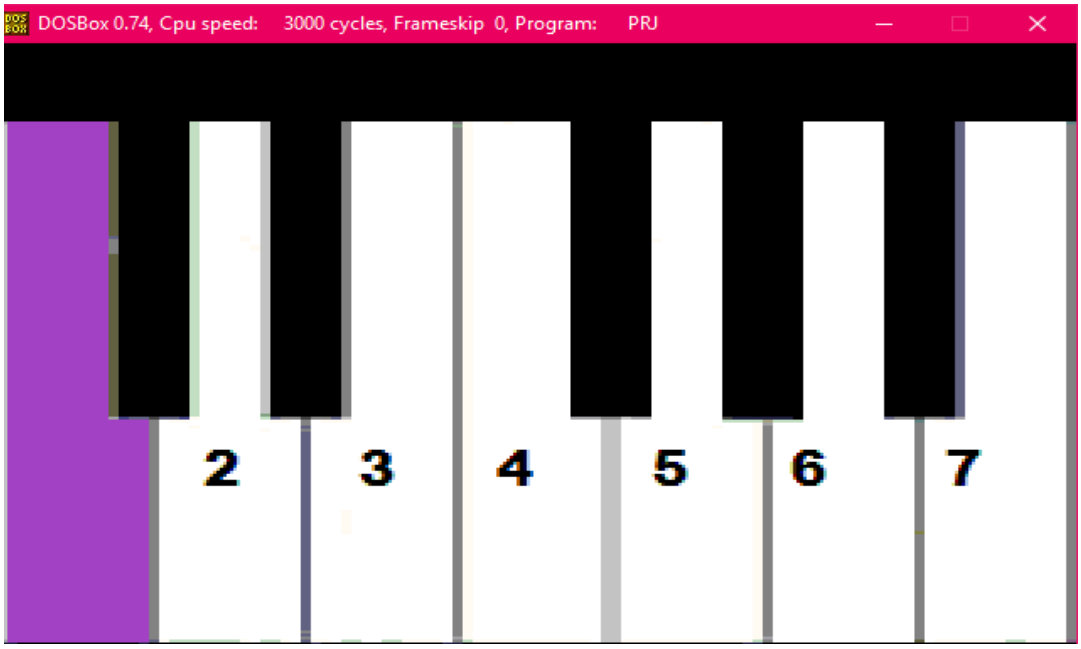
Game Title



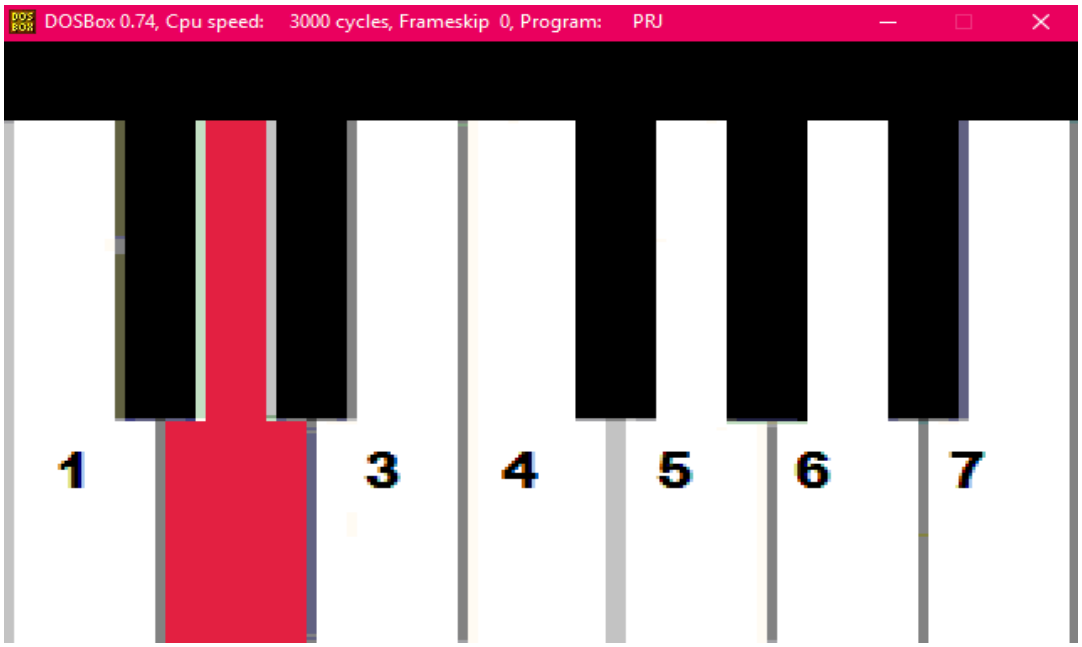
Main menu of the game



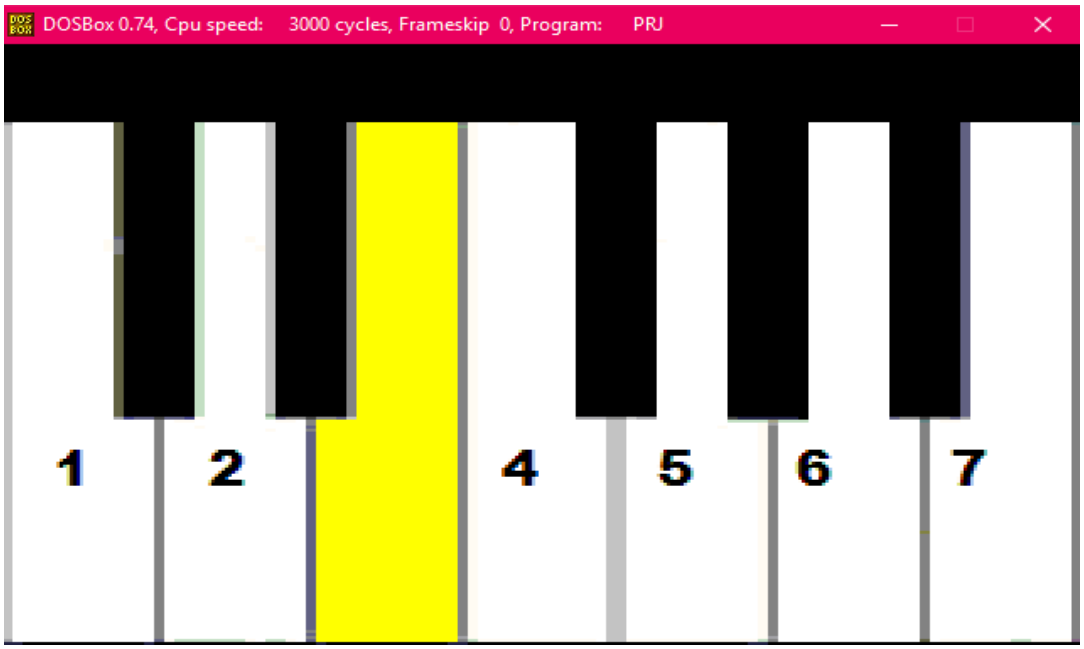
Keyboard/Play -- Default Keyboard



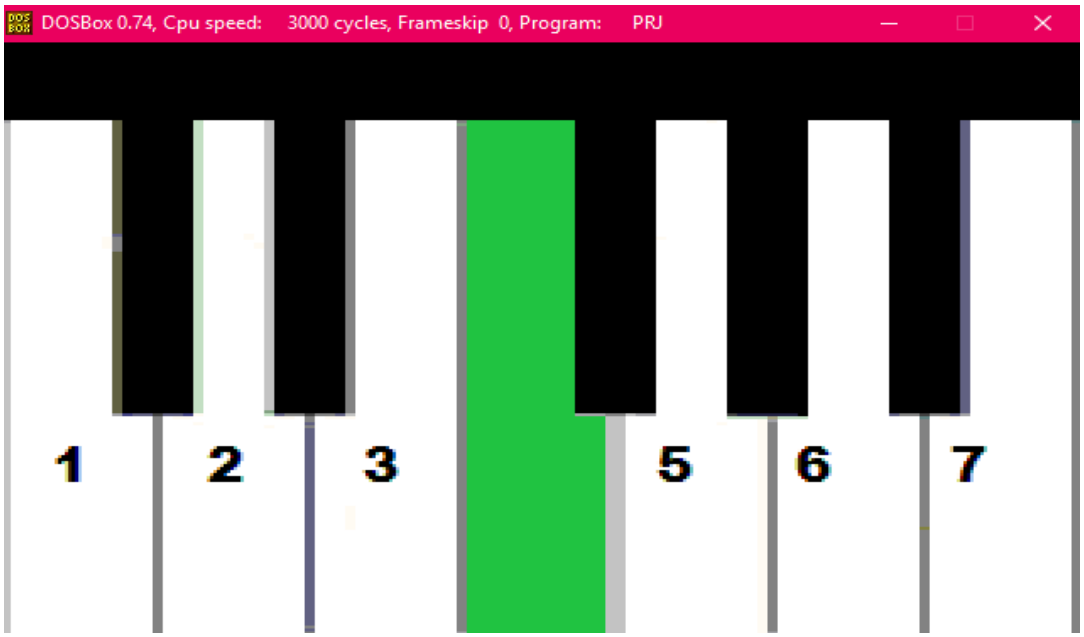
Keyboard/Play -- When 'do' or input 1 is press



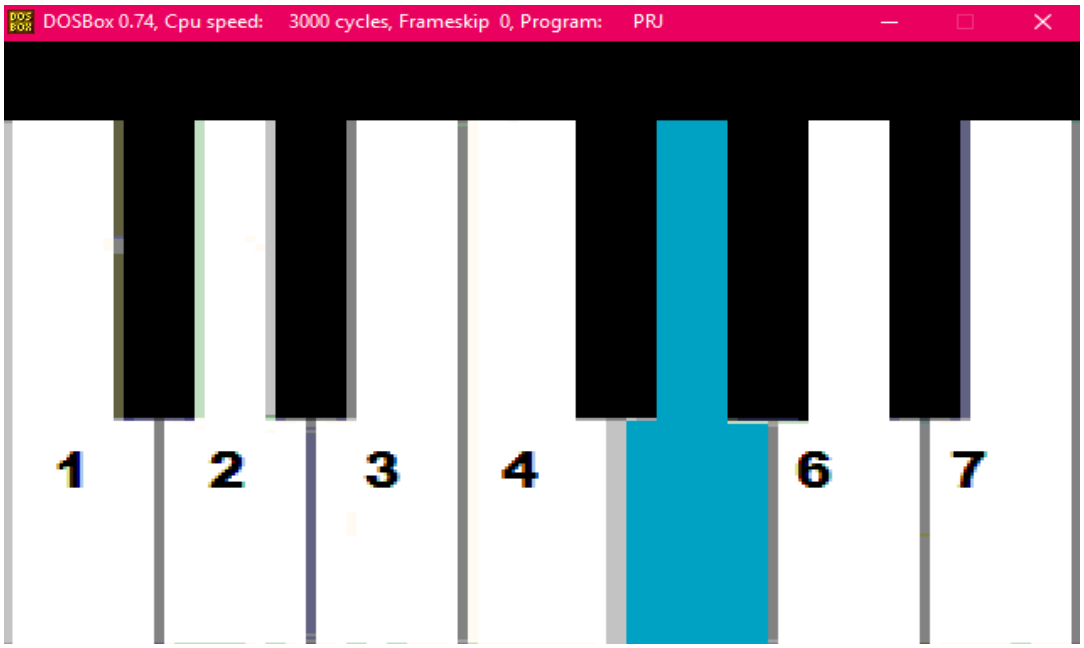
Keyboard/Play -- When 're' or input 2 is press



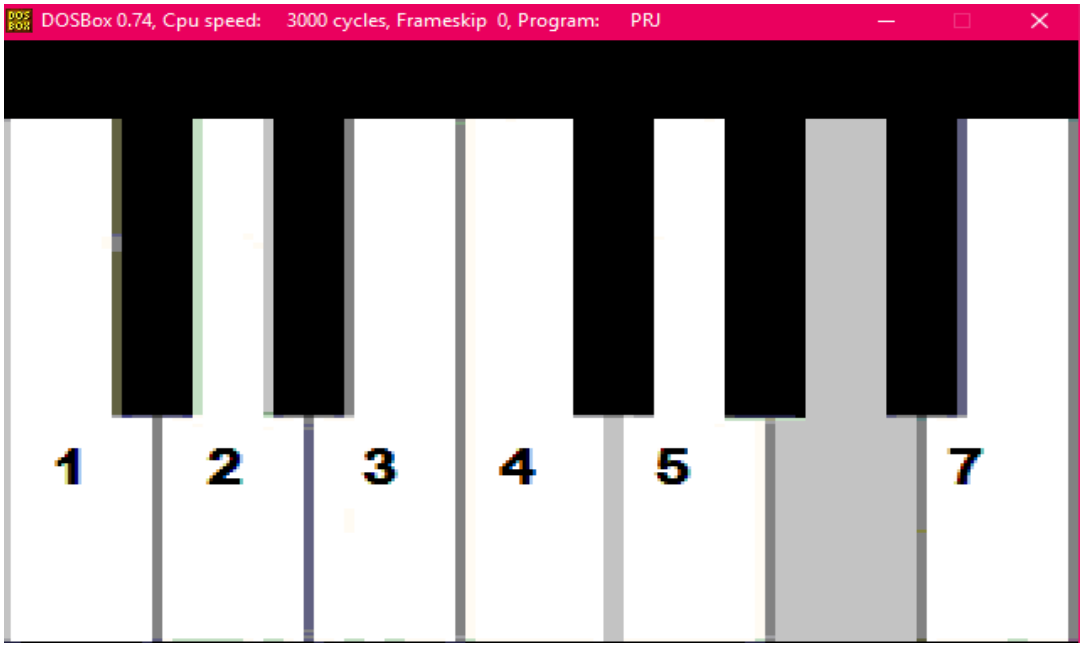
Keyboard/Play -- When 'mi' or input 3 is press



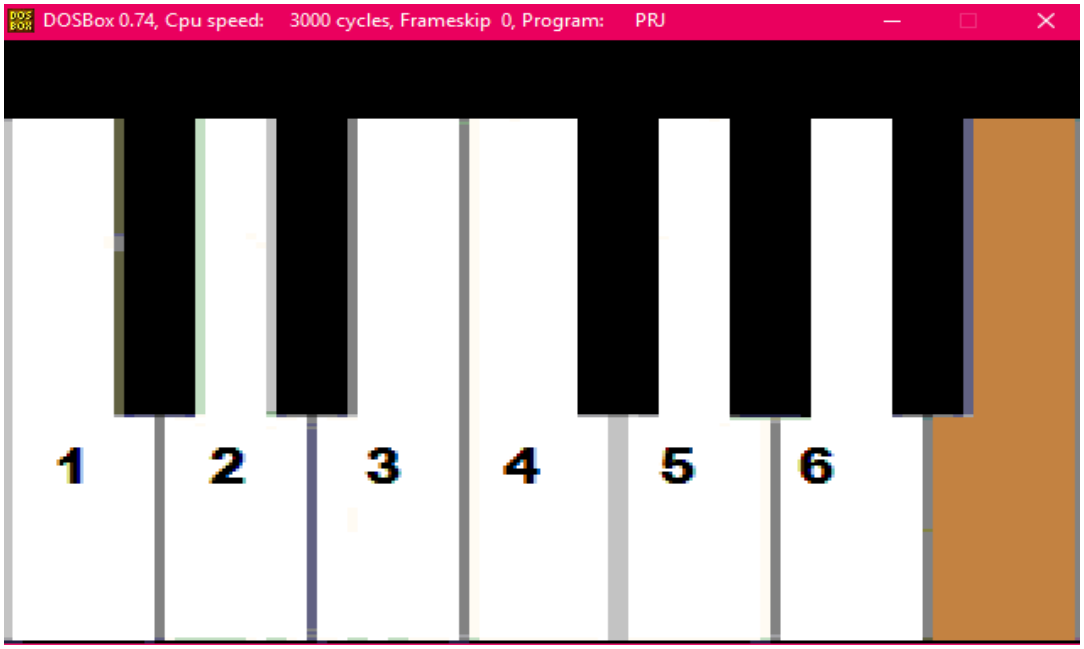
Keyboard/Play -- When 'fa' or input 4 is press



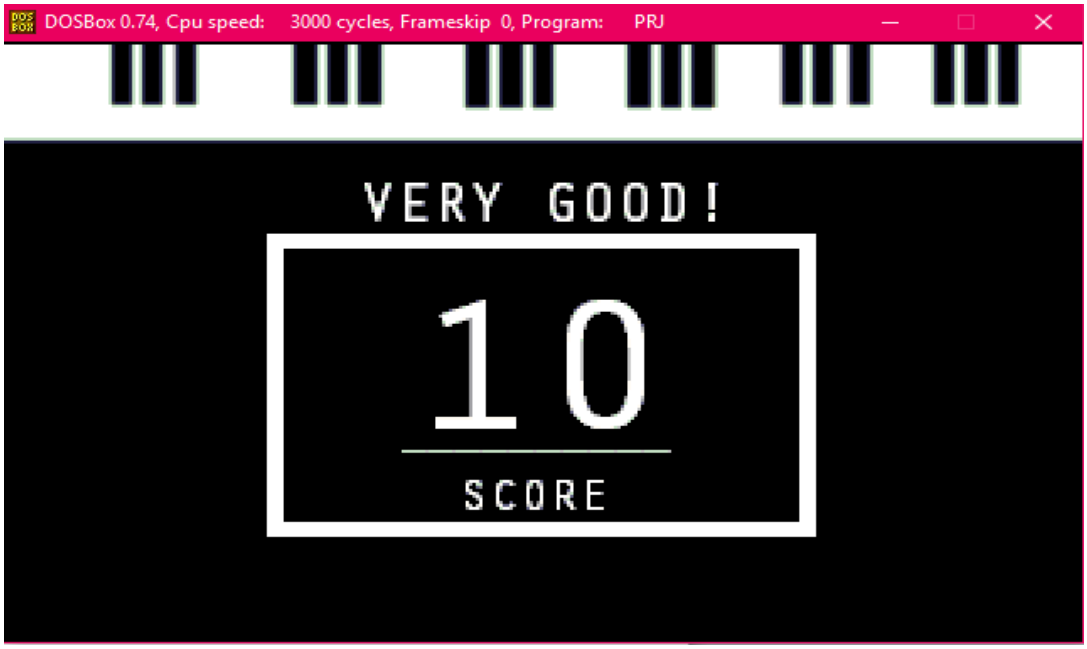
Keyboard/Play -- When' sol' or input 5 is press



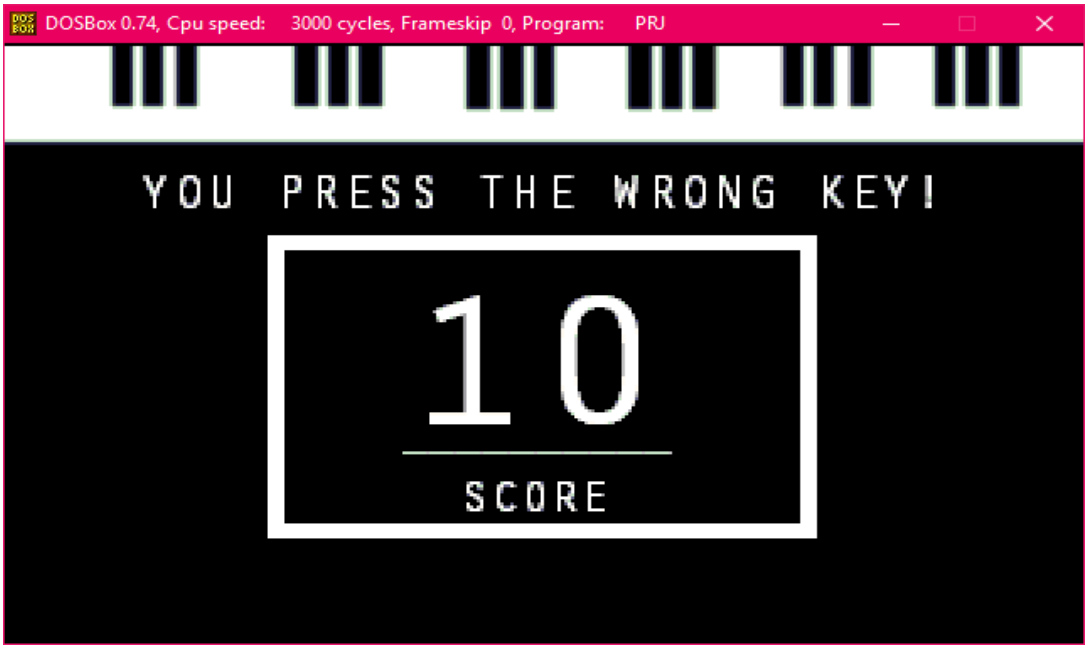
Keyboard/Play -- When 'la' or input 6 is press



Keyboard/Play -- When 'si' or input 7 is press

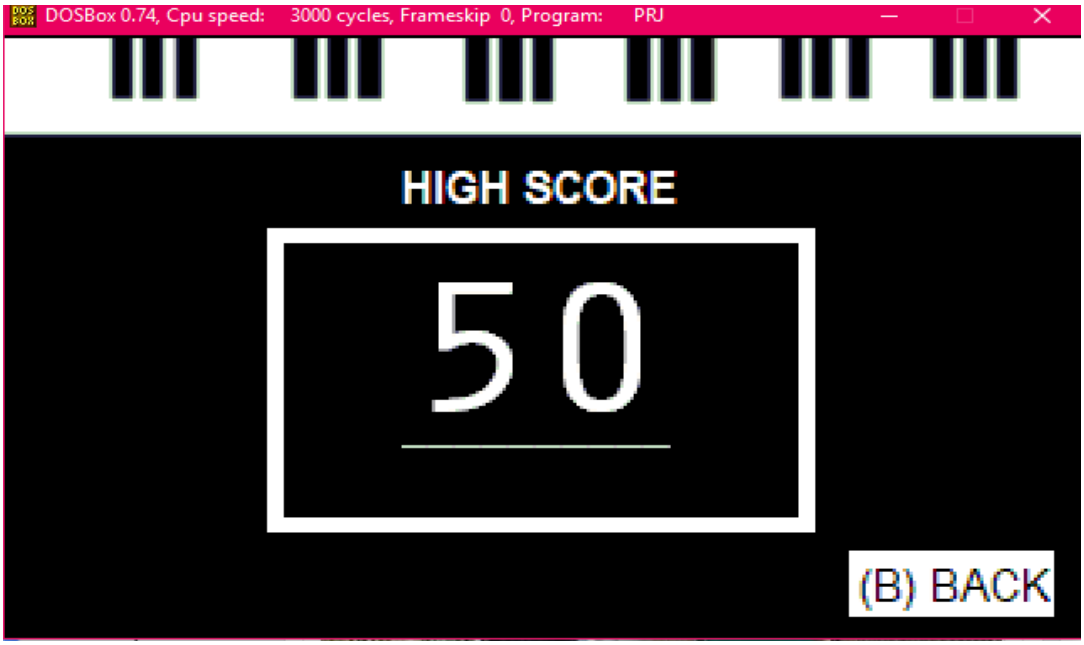


Completed Level 1

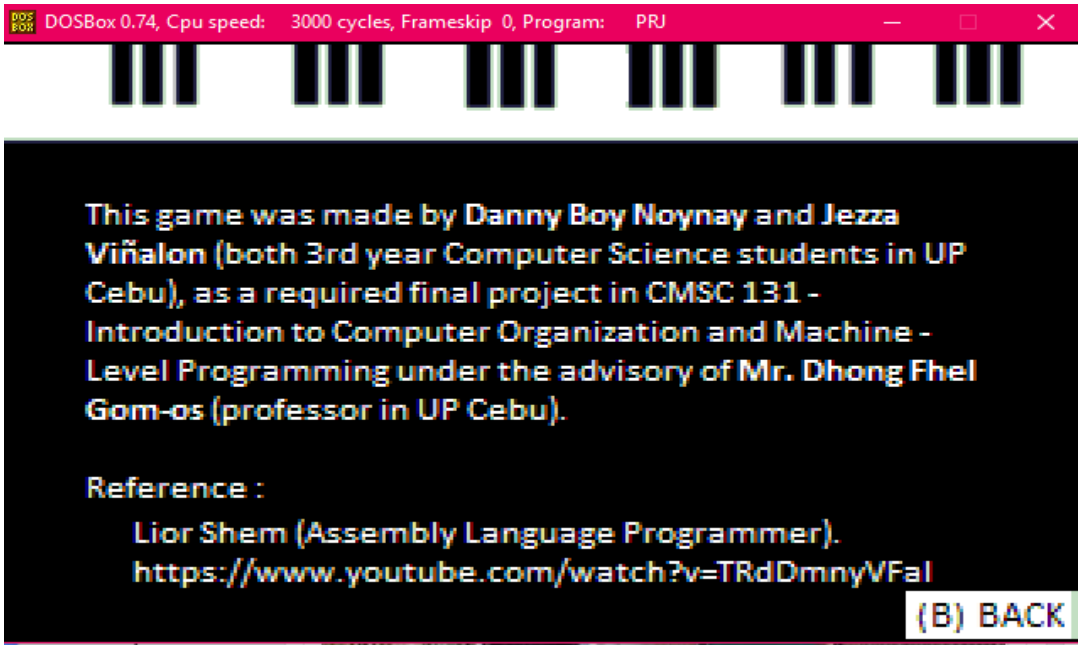


When wrong key was pressed in Level 2

How To Play?



High Score


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: PRJ
This game was made by Danny Boy Noynay and Jezza Viñalon (both 3rd year Computer Science students in UP Cebu), as a required final project in CMSC 131 - Introduction to Computer Organization and Machine - Level Programming under the advisory of Mr. Dhong Fhel Gom-os (professor in UP Cebu).
Reference :
Lior Shem (Assembly Language Programmer).
https://www.youtube.com/watch?v=TRdDmnyVFaI
(B) BACK

Credits