University of the Philippines Cebu

Gorodo Avenue, Lahug, Cebu City

College of Science

Department of Computer Science

CMSC 131

Introduction to Computer Organization & Machine-level Programming

BATTLE GRIDS

Ena Margarita O. Bojos

Micah Nicole J. Chu Im

BS Computer Science 3

Table of contents

1. Project Summary/Description

1.1 Game Concept and Look 3

1.2 Gameplay 3

2. System Specifications

2.1 List of Procedures (w/ definition)

2.1.1 MAIN 4

2.1.2 PRINT_GRID 5

2.1.3 PRINT_SCORE 6

2.1.4 PRINT_MISSED 6

2.1.5 MOVE_ARROW 7

2.1.6 GET_KEY 12

2.1.7 SET_CURSOR 12

2.1.8 _GET_CHAR_AT_CURSOR 12

2.1.9 DELAY 13

2.1.10 FILE_READ 14

2.1.11 READ_HS 16

2.1.12 CHANGE_STR 17

2.1.13 FILE_WRITE 22

2.1.14 CLEAR_SCREEN 23

2.1.15 TO_STRING 24

2.1.16 CHECK_HS 26

3. Screen Cap of Game (w/ description) 29

Project Summary/ Description

1.1 Game Concept and Look

Inspired from the game Battleships, the player must destroy all ships. It is a one-player grid-based game where the ships will be hidden, and the player must use the arrow keys to choose a grid to attack.

In the game screen you'll see a 7x7 grid. The ships are hidden, you won't be able to see it. They are pre-positioned. Each ship occupies a number of consecutive squares on the grid, arranged either horizontally or vertically. The number of squares for each ship is determined by the type of the ship. The ships cannot overlap. You'll know you hit one when the square you've chosen printed a "O".

1.2 Gameplay

The game is simple. The player will use the arrow keys to navigate around the grid. The ships are pre-positioned. The player would then guess where the ships are positioned. When he has chosen a location, he must press ENTER to attack. If the chosen location shows "X", it means he missed. If it shows "O", it means that he has successfully hit the ship. If the player has 10 misses, the game is over. He wins if he successfully sunk all ships.

Link to GitHub:

https://github.com/enmargaret/CMSC-131

2. System Specifications

2.1 Procedures

2.1.1 Main

```
MAIN PROC FAR
        MOV AX,@DATA
        MOV DS,AX

        CALL PRINT_GRID
        CALL MOVE_ARROW

EXIT:
        MOV AH,4CH
        INT 21H

MAIN ENDP
```

This is the main procedure. We called our PRINT_GRID procedure and our MOVE_ARROW procedure here. This is the procedure that runs first.

2.1.2 PRINT_GRID

```
PRINT_GRID PROC NEAR ;Prints the grid on to the screen
        MOV DH, FIVE
        MOV DH_COL, DH
        MOV DL, ZERO
        MOV DL_ROW, DL
        CALL FILE_READ

        MOV DL, 6
        MOV DH, 1
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, HIGH_SCORE
        INT 21H

        CALL READ_HS
        MOV DL, 10
        MOV DH, 2
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, H_SCORE_STR
        INT 21H
        MOV DL, 67
        MOV DH, 1
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, CURRENT_SCORE
        INT 21H

        MOV DL, 32
        MOV DH, 1
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, NUM_MISS
        INT 21H
        RET
```

This procedure prints the grid to the screen

```
MOV DL, 26
        MOV DH, 6
        CALL SET_CURSOR
PRINT_GRID ENDP
```

2.1.3 PRINT_SCORE

```
PRINT_SCORE PROC   NEAR
        MOV DL, 69
        MOV DH, 2
        CALL SET_CURSOR
        LEA DX, INPUT
        PUSH DX
        CALL DISPLAY

        MOV DL, DL_ROW
        MOV DH, DH_COL
        CALL SET_CURSOR
        RET
PRINT_SCORE ENDP
```

This procedure is for printing the score.

2.1.4 PRINT_MISSED

```
PRINT_MISSED PROC  NEAR
        MOV DL, 37
        MOV DH, 2
        CALL SET_CURSOR
        LEA DX, MISS_STR
        PUSH DX
        CALL DISPLAY

        MOV DL, DL_ROW
        MOV DH, DH_COL
        CALL SET_CURSOR
        JMP ITERATE
        RET
PRINT_MISSED ENDP
```

This procedure is for printing whether you missed your attack or not.

2.1.5 MOVE_ARROW

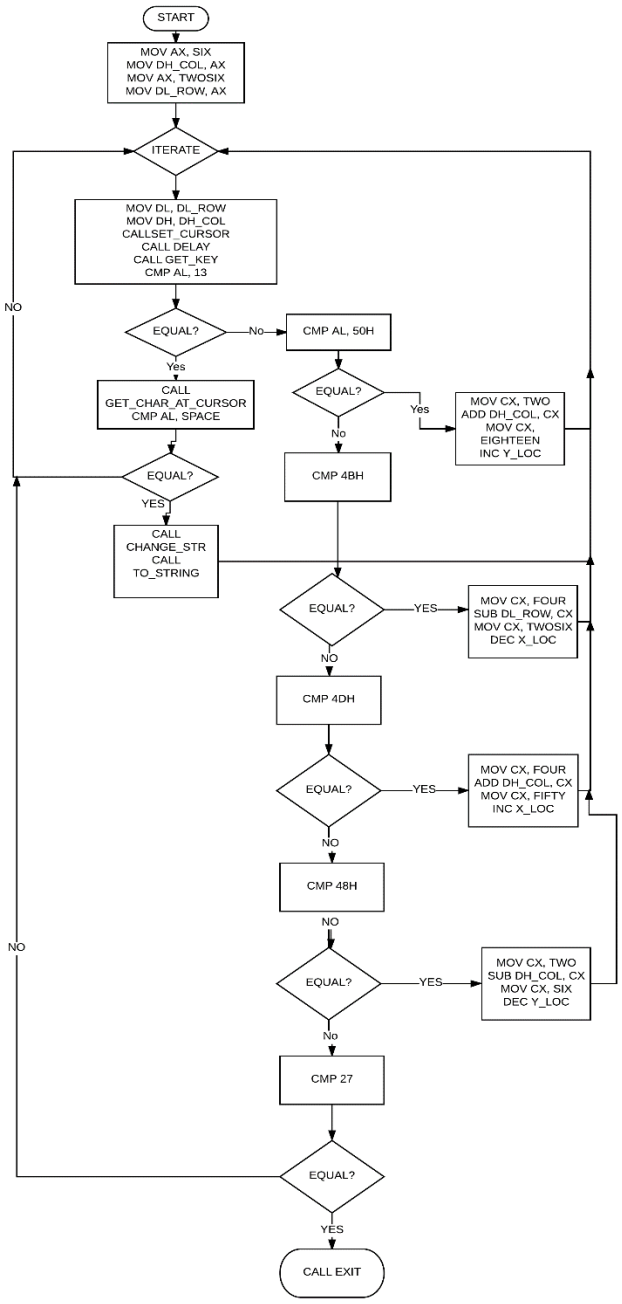
```
MOVE_ARROW PROC NEAR ;moves the cursor and checks for keys pressed

        MOV AX, SIX
        MOV DH_COL, AX
        MOV AX, TWOSIX
        MOV DL_ROW, AX

        ITERATE:
                MOV     DL, DL_ROW
                MOV     DH, DH_COL
                CALL SET_CURSOR

                CALL DELAY

                CALL GET_KEY

                CMP AL, 13;checks if the 'enter' key has been pressed
                JE ENTER_KEY

                CMP AL, 50H ;checks if the 'arrow down' key has been pressed
                JE ADD_DOWN

                CMP AL, 4BH ;checks if the 'arrow right' key has been pressed
                JE ADD_LEFT

                CMP AL, 4DH ;checks if the 'arrow left' key has been pressed
                JE ADD_RIGHT

                CMP AL, 27 ;checks if the 'esc' key has been pressed
                JE EXIT_PROG

                CMP AL, 48H ;checks if the 'arrow up' key has been pressed
                JE ADD_UP
                JA OTHERS
                JL OTHERS
```

This procedure basically moves the cursor and checks what keys are pressed.

```
EXIT_PROG:
                CALL EXIT


        ADD_UP:   ;moves the cursor up
                MOV CX, TWO
                SUB     DH_COL, CX
                MOV CX, SIX
                DEC Y_LOC
                CMP DH_COL, CX
                JL UPPER_BORDER
                JMP     ITERATE

        ADD_LEFT: ;moves the cursor left
                MOV CX, FOUR
                SUB     DL_ROW, CX
                MOV     CX, TWOSIX
                DEC X_LOC
                CMP     DL_ROW, CX
                JL LEFT_BORDER
                JMP     ITERATE

        ADD_DOWN: ;moves the cursor down
                MOV CX, TWO
                ADD     DH_COL, CX
                MOV CX, EIGHTEEN
                INC Y_LOC
                CMP     DH_COL, CX
                JG LOWER_BORDER
                JMP     ITERATE

        ADD_RIGHT: ;moves the cursor right
                MOV CX, FOUR
                ADD     DL_ROW, CX
                MOV CX, FIFTY
                INC X_LOC
                CMP     DL_ROW, CX
                JG RIGHT_BORDER
                JMP     ITERATE
```

```
RIGHT_BORDER: ;locks the cursor within the right boundary
                MOV CX, FIFTY
                MOV DL_ROW, CX
                MOV CX, SEVEN
                MOV X_LOC, CX
                JMP ITERATE

        LEFT_BORDER: ;locks the cursor within the left boundary
                MOV CX, TWOSIX
                MOV DL_ROW, CX
                MOV CL, ONE
                MOV X_LOC, CX
                JMP ITERATE

        UPPER_BORDER: ;locks the cursor within the upper boundary
                MOV CX, SIX
                MOV DH_COL, CX
                MOV CX, ONE
                MOV Y_LOC, CX
                JMP ITERATE

        LOWER_BORDER: ;locks the cursor within the lower boundary
                MOV CX, EIGHTEEN
                MOV DH_COL, CX
                MOV CX, SEVEN
                MOV Y_LOC, CX
                JMP ITERATE

        ENTER_KEY: ;when the enter key is pressed, it then checks whether player has hit a ship
or not
                CALL _GET_CHAR_AT_CURSOR
                CMP AL, SPACE
                JE CONTINUE
                CMP AL, HIT
                JE ITERATE
                CMP AL, MISS
                JE ITERATE
```

```
CONTINUE:
                        CALL CHANGE_STR
                        CALL TO_STRING
                        JMP ITERATE

    OTHERS:
            JMP ITERATE

    TERMINATE:
            MOV AH, 4CH
            INT 21H

    RET
MOVE_ARROW ENDP
```



*Flowchart of MOVE_ARROW procedure

2.1.6 GET_KEY

```
GET_KEY         PROC    NEAR
        MOV AH, 07
        INT 21H


        RET
GET_KEY         ENDP
```

This procedure gets the key inputted by the user

2.1.7 SET_CURSOR

```
SET_CURSOR PROC     NEAR
        MOV         AH, 02H
        MOV         BH, 00
        INT         10H
        RET
SET_CURSOR ENDP
```

This procedure sets the cursor when called.

2.1.8 _GET_CHAR_AT_CURSOR

```
_GET_CHAR_AT_CURSOR PROC NEAR
        MOV         AH, 08H
        MOV         BH, 00
        INT         10H
_GET_CHAR_AT_CURSOR ENDP
```

This procedure gets the character at where the cursor is at.

2.1.9 DELAY

```
DELAY PROC              NEAR
mov bp, 2 ;lower value faster
mov si, 2 ;lower value faster
        delay2:
                dec bp
                nop
                jnz delay2
                dec si
                cmp si,0
                jnz delay2
                RET
DELAY ENDP
```

This procedure delays the speed of the loading bar at the start of the game.

2.1.10 FILE_READ

```
FILE_READ PROC NEAR
;open file
        MOV AH, 3DH         ;request open file
        MOV AL, 00          ;read only; 01 (write only); 10 (read/write)
        LEA DX, PATHFILENAME
        INT 21H
        JC DISPLAY_ERROR1
        MOV FILEHANDLE, AX

 ;read file
        MOV AH, 3FH         ;request read record
        MOV BX, FILEHANDLE    ;file handle
        MOV CX, 1000        ;record length
        LEA DX, RECORD_STR    ;address of input area
        INT 21H
        JC DISPLAY_ERROR2
        CMP AX, 00          ;zero bytes read?
        JE DISPLAY_ERROR3

        MOV STR_LEN, AL

        CALL CLEAR_SCREEN
        MOV DL, DL_ROW
        MOV DH, DH_COL
        CALL SET_CURSOR

        ;display record
        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        INC DL_ROW
```

This procedure is for the File reading. We used text files for our grid.

```
;close file handle
        MOV AH, 3EH         ;request close file
        MOV BX, FILEHANDLE    ;file handle
        INT 21H

        RET

DISPLAY_ERROR1:
 LEA DX, ERROR1_STR
 MOV AH, 09
 INT 21H

 JMP EXIT

DISPLAY_ERROR2:
 LEA DX, ERROR2_STR
 MOV AH, 09
 INT 21H

 JMP EXIT

DISPLAY_ERROR3:
 LEA DX, ERROR3_STR
 MOV AH, 09
 INT 21H
FILE_READ ENDP
```

2.1.11 READ_HS

```
READ_HS PROC NEAR

;open file
        MOV AH, 3DH         ;request open file
        MOV AL, 00          ;read only; 01 (write only); 10 (read/write)
        LEA DX, FR_PATHFILENAME
        INT 21H
        JC DISPLAY_ERROR1
        MOV FILEHANDLE, AX

        ;read file
        MOV AH, 3FH         ;request read record
        MOV BX, FILEHANDLE    ;file handle
        MOV CX, 49          ;record length
        LEA DX, H_SCORE_STR    ;address of input area
        INT 21H

        JC DISPLAY_ERROR2
        CMP AX, 00          ;zero bytes read?
        JE DISPLAY_ERROR3

        MOV STR_LEN2, AX

        ;close file handle
        MOV AH, 3EH         ;request close file
        MOV BX, FILEHANDLE    ;file handle
        INT 21H

        RET
READ_HS ENDP
```

This procedure is for reading the high score from the text file.

2.1.12 CHANGE_STR

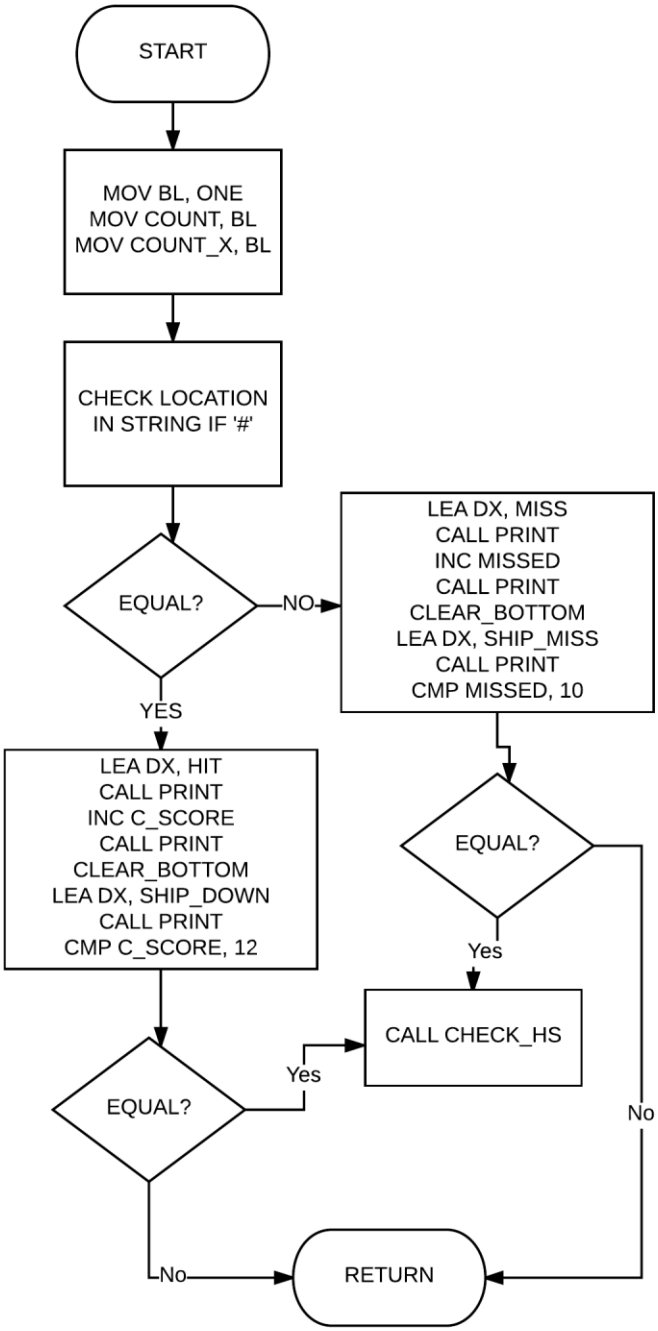
```
CHANGE_STR PROC NEAR
        MOV BL, ONE
        MOV COUNT, BL
        MOV COUNT_X, BL

        CMP Y_LOC, BL
        JE STRING_1

        MOV BL, TWO
        CMP Y_LOC, BL
        JE STRING_2

        MOV BL, THREE
        CMP Y_LOC, BL
        JE STRING_3

        MOV BL, FOUR
        CMP Y_LOC, BL
        JE STRING_4

        MOV BL, FIVE
        CMP Y_LOC, BL
        JE STRING_5

        MOV BL, SIX
        CMP Y_LOC, BL
        JE STRING_6

        MOV BL, SEVEN
        CMP Y_LOC, BL
        JE STRING_7
```

This procedure checks whether you hit a ship or not.

```
STRING_1:
        LEA SI, C1
        JMP CHECK_X

STRING_2:
        LEA SI, C2
        JMP CHECK_X

STRING_3:
        LEA SI, C3
        JMP CHECK_X

STRING_4:
        LEA SI, C4
        JMP CHECK_X

STRING_5:
        LEA SI, C5
        JMP CHECK_X

STRING_6:
        LEA SI, C6
        JMP CHECK_X

STRING_7:
        LEA SI, C7
        JMP CHECK_X

CHECK_X:
        MOV BL, COUNT_X
        CMP X_LOC, BL
        JL ADD_X
        JE CHECK_STAR
```

```
ADD_X:
        INC SI
        INC COUNT_X
        JMP CHECK_X

CHECK_STAR:
        MOV AL, [SI]
        CMP AL, '*'
        JE PRINT_MISS
        JNE CHECK_HASH

CHECK_HASH:
        CMP AL, '#'
        JE PRINT_HIT
        JMP RETURN
PRINT_MISS:
        MOV DL, DL_ROW
        MOV DH, DH_COL
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, MISS
        INT 21H
        INC MISSED
        CALL CLEAR_BOTTOM
        MOV DL, 30
        MOV DH, 22
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, SHIP_MISS
        INT 21H

        CMP MISSED, 10
        JE GO_TO_EXIT

        JMP RETURN
```

```
PRINT_HIT:
        MOV DL, DL_ROW
        MOV DH, DH_COL
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, HIT
        INT 21H
        INC C_SCORE
        CALL CLEAR_BOTTOM
        MOV DL, 28
        MOV DH, 22
        CALL SET_CURSOR
        MOV AH, 09
        LEA DX, SHIP_DOWN
        INT 21H

        CMP C_SCORE, 12
        JE GO_TO_EXIT

        JMP RETURN

GO_TO_EXIT:
        CALL CHECK_HS
RETURN:
        RET
CHANGE_STR ENDP
```



*Flow chart of CHANGE_STR procedure

*note: only procedures with logic has a flowchart.

2.1.13 FILE_WRITE

```
FILE_WRITE PROC NEAR
  MOV AH, 3CH          ;request create file
  MOV CX, 00           ;normal attribute
  LEA DX, FW_PATHFILENAME  ;load path and file name
  INT 21H
  JC FW_DISPLAY_ERROR1     ;if there's error in creating file, carry flag = 1, otherwise 0
  MOV FW_FILEHANDLE, AX

  ;write file
  MOV AH, 40H          ;request write record
  MOV BX, FILEHANDLE    ;file handle
  MOV CX, STR_LEN2           ;record length
  LEA DX, C_SCORE_STR    ;address of output area
  INT 21H
  JC FW_DISPLAY_ERROR2     ;if carry flag = 1, there's error in writing (nothing is written)
  CMP AX, STR_LEN2           ;after writing, set AX to size of chars nga na write
  JNE FW_DISPLAY_ERROR3

  ;close file handle
  MOV AH, 3EH          ;request close file
  MOV BX, FW_FILEHANDLE    ;file handle
  INT 21H

  RET

FW_DISPLAY_ERROR1:
  LEA DX, FW_ERROR1_STR
  MOV AH, 09
  INT 21H

  JMP EXIT
```

This procedure is for writing the new high score to a file.

```
FW_DISPLAY_ERROR2:
  LEA DX, FW_ERROR2_STR
  MOV AH, 09
  INT 21H

  JMP EXIT

FW_DISPLAY_ERROR3:
  LEA DX, FW_ERROR3_STR
  MOV AH, 09
  INT 21H

FILE_WRITE ENDP
```

2.1.14 CLEAR_SCREEN

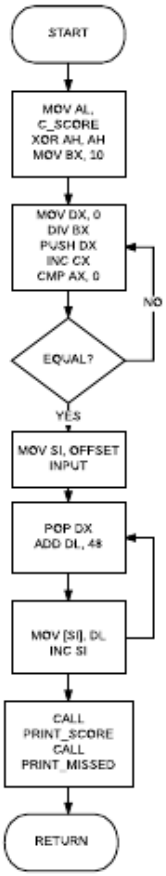
```
CLEAR_SCREEN PROC NEAR
  ; MOV AX, 0600H   ;full screen

  ;MOV BH, 07H     ;black background
  ;MOV CX, 0000H   ;upper left row: column (00:00)
  ;MOV DX, 184FH   ;lower right row: column (24:79)
  ;INT 10H

  MOV AX, 0600H   ;full screen
  MOV BH, 07H     ;white background (7), blue foreground (1)
  MOV CX, 0000H   ;upper left row: column (01:01)
  MOV DX, 184FH   ;lower right row: column (23:78)
  INT 10H

  RET
CLEAR_SCREEN ENDP
```

This procedure is for clearing the screen.

2.1.15 TO_STRING

```
TO_STRING PROC NEAR
        MOV AL, C_SCORE
        XOR AH, AH

        ;number to convert is in AX
        ;variable to store to is INPUT

         mov  bx, 10
         mov  cx, 0
        cycle1:
         mov  dx, 0
         div  bx
         push dx
         inc  cx
         cmp  ax, 0
         jne  cycle1

         mov  si, offset INPUT
        cycle2:
         pop  dx
         add  dl, 48
         mov  [si], dl
         inc  si
         loop cycle2

         CALL PRINT_SCORE
         CALL TO_STRING_MISS
         CALL PRINT_MISSED

        RET
TO_STRING ENDP
```

This procedure is for converting the score to string to be printed on screen.



*Flow chart of TO_STRING procedure

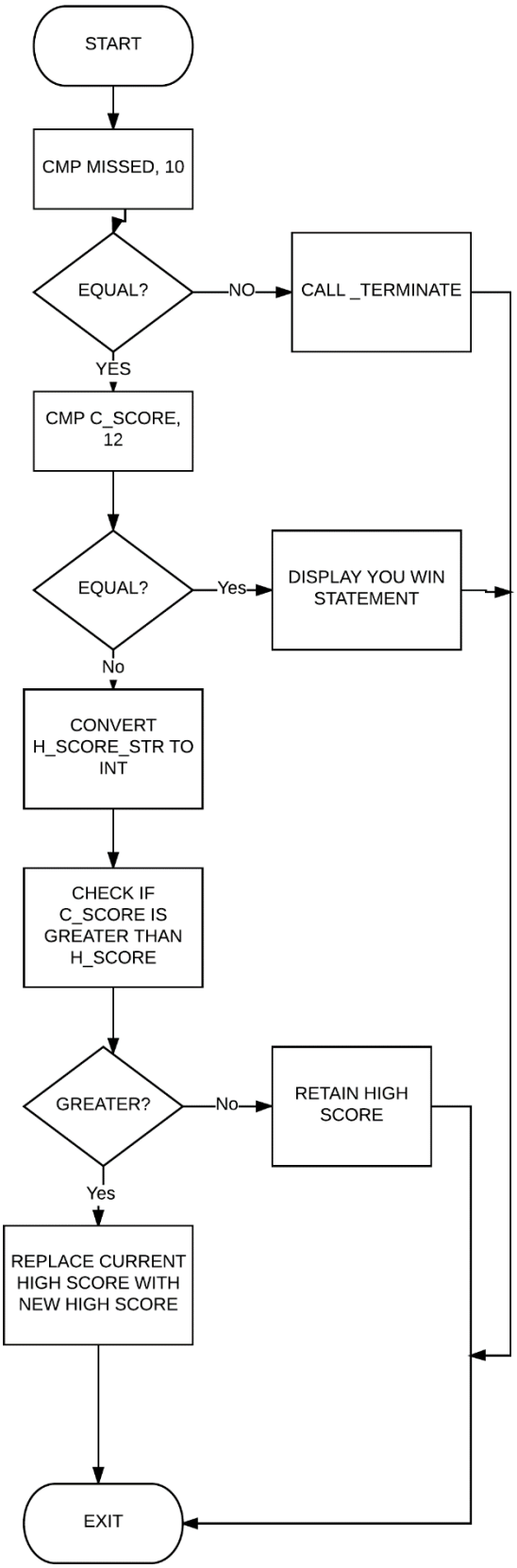
*note: there is also a TO_STRING_MISS procedure. It basically has the same code and flowchart with TO_STRING procedure.

2.1.16 CHECK_HS

```
CHECK_HS PROC NEAR
        CMP MISSED, 10
        JE GAME_OVER
        CMP C_SCORE, 12
        JE YOU_WIN

        XOR AH, AH

        LEA SI, H_SCORE_STR
        MOV CX, STR_LEN2
        MOV NUM, 0
        CMP CX, 1
        JE ONE_DIGIT

        MOV BX, 10
        REPEAT:
        LEA SI, H_SCORE_STR
        MOV AL, [SI]
        SUB AL, 48
        MOV AH, 00
        MUL BX

        MOV NUM, AX
        MOV BX, 0
        INC SI
```

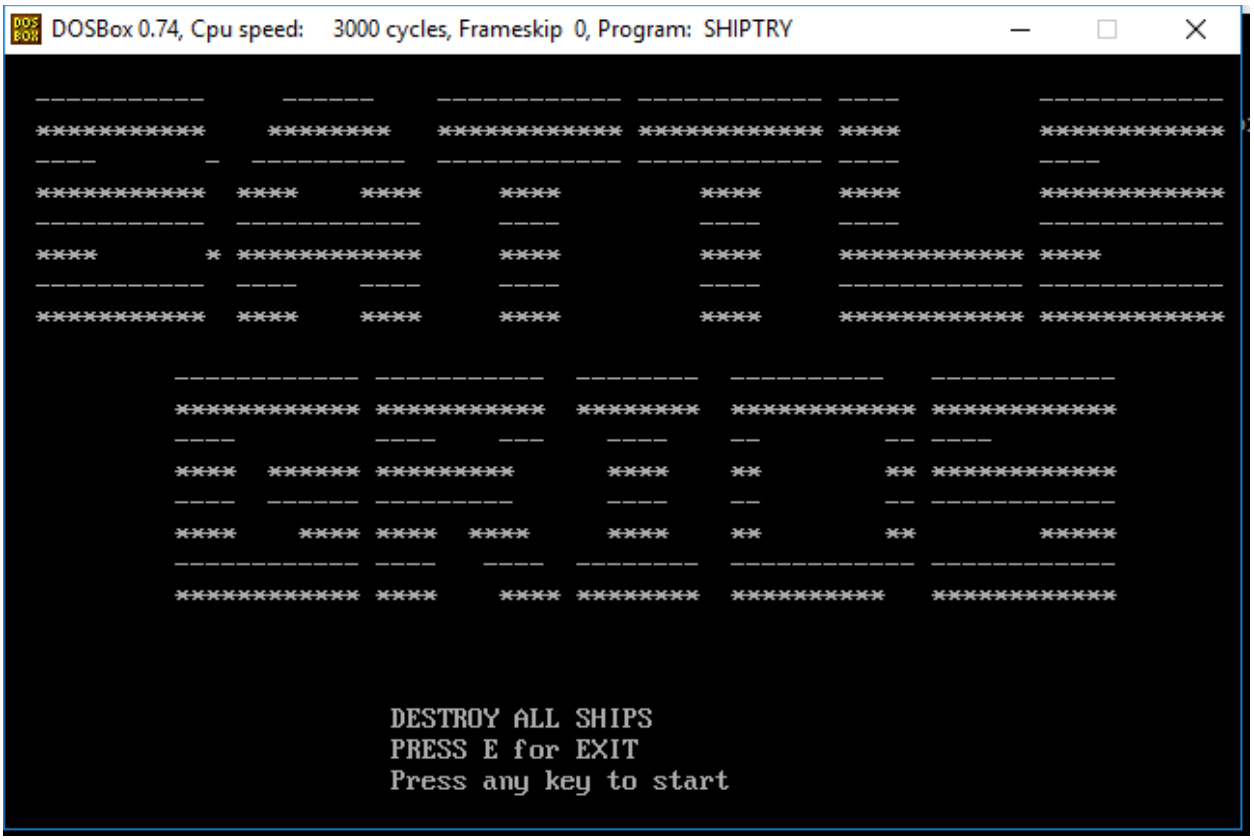
This procedure will check if the player has won or lose game. It will also check whether the player has surpassed the current high score.

```
ONE_DIGIT:
        MOV AL, [SI]
        SUB AL, 48
        MOV AH, 00
        ADD NUM, AX

        MOV CX, C_SCORE
        CMP CX, NUM
        JG REPLACE_HS
        JLE NO_CHANGE

GAME_OVER:
        CALL _TERMINATE

YOU_WIN:
        CALL CLEAR_BOTTOM
        MOV DL, 34
        MOV DH, 22
        CALL SET_CURSOR
        LEA DX, WIN
        CALL PRINT
        MOV AH,4CH
        INT 21H
REPLACE_HS:
        CALL FILE_WRITE

NO_CHANGE:
        CALL EXIT
        RET
CHECK_HS ENDP
```



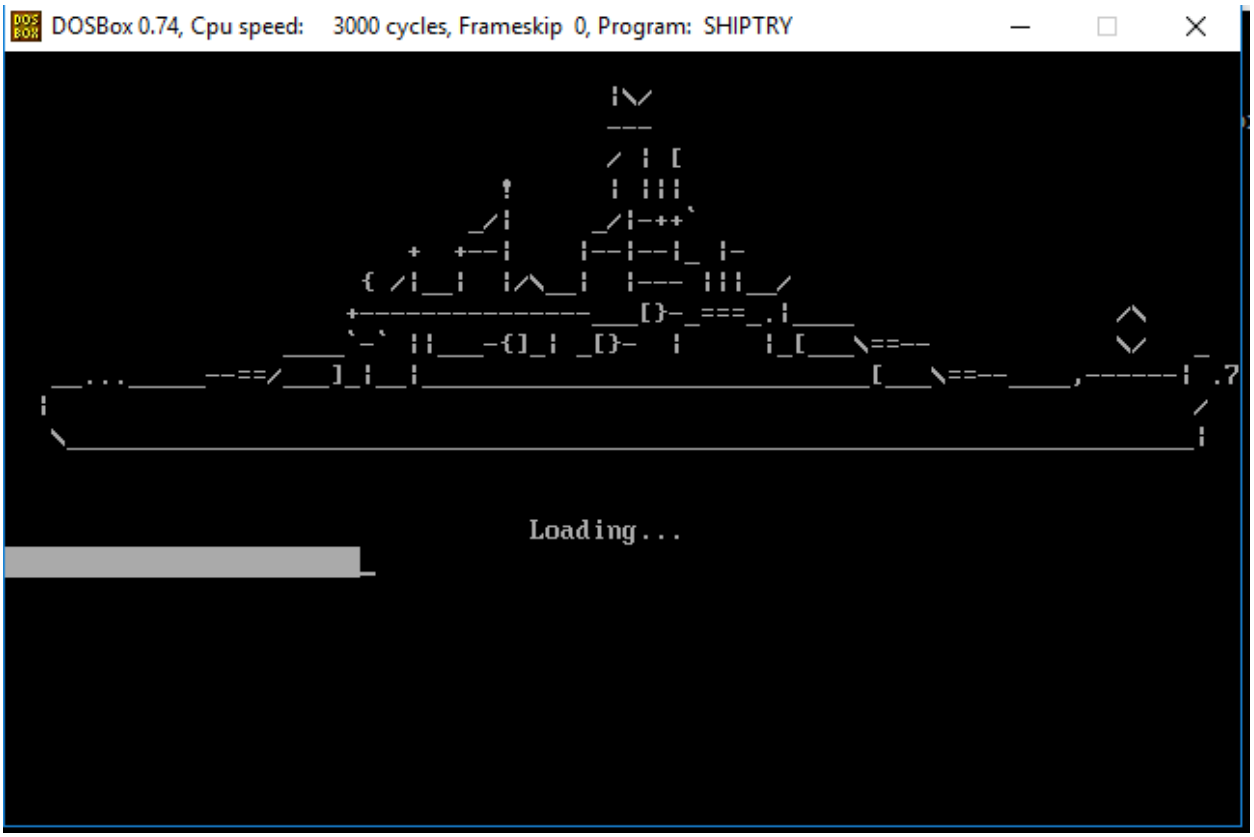
* Flow chart of CHECK_HS procedure

3. Screen cap of the game

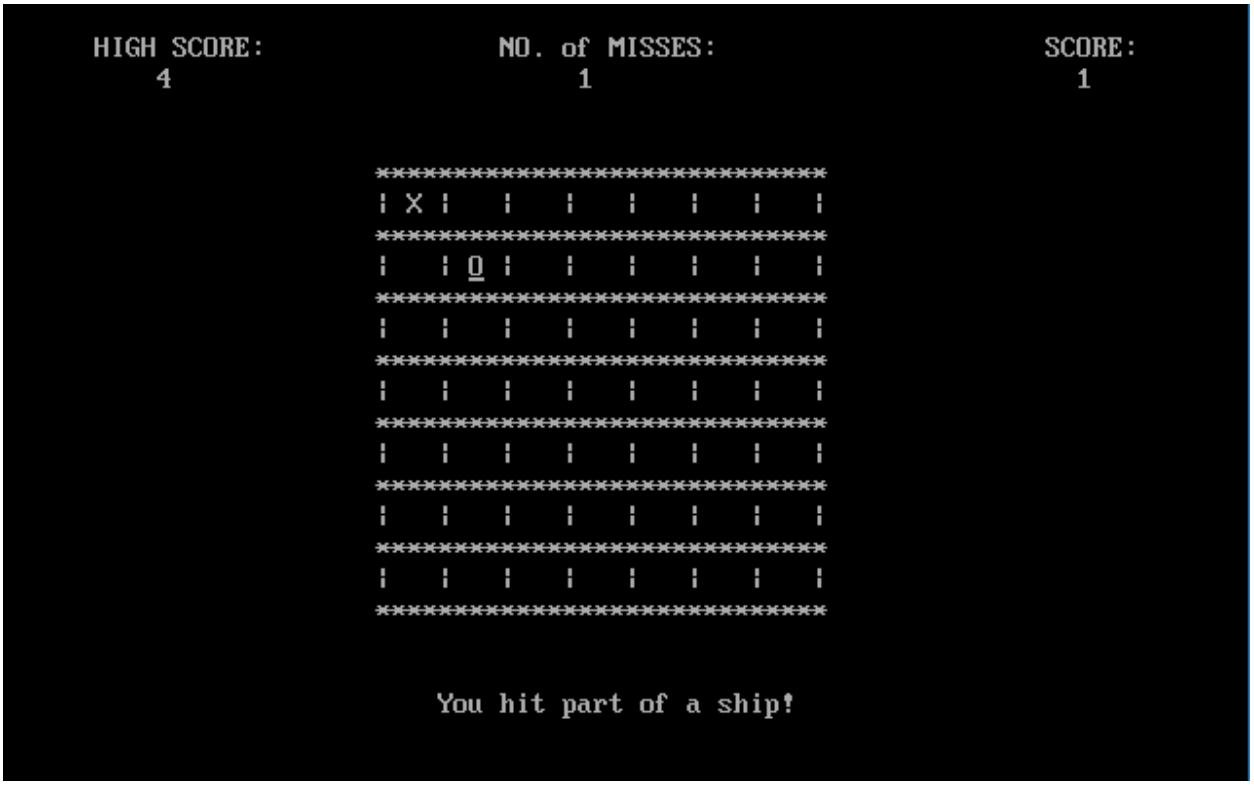


This is the main menu of the game.

This is the instructions or help screen.

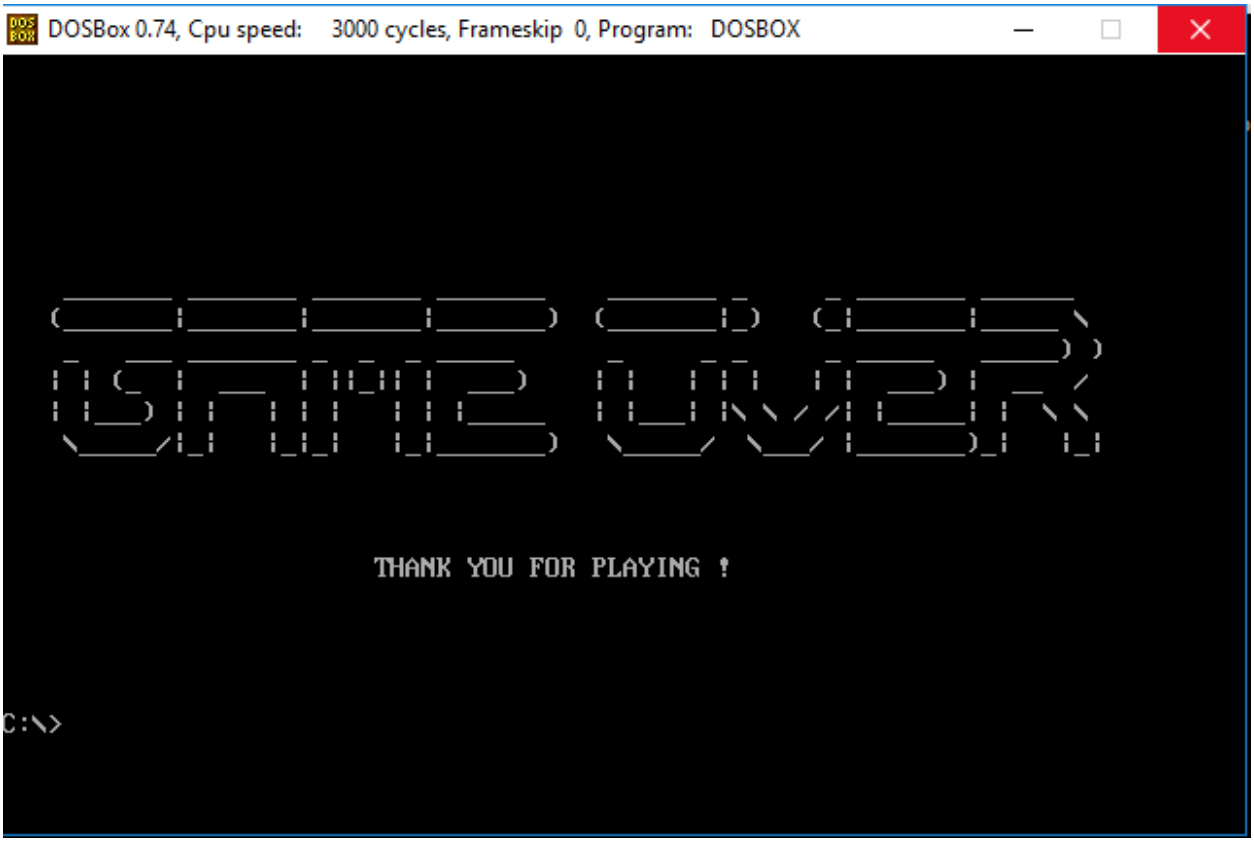


This is the loading screen of the game. Before the game screen appears, the loading screen will show.



This is the game screen. You'll see the current high score, your no. of misses and your score.

You will also know if you've hit a part of a ship.



This is the game over screen.