University of the Philippines Cebu
College of Science
Department of Computer Science

CMSC 131: Introduction to Computer Organization and Machine-level Programming

CAPSTONE PROJECT

“SWITCH ME NOT”

Pacaña, Michael Ervin
Salera, Marie Curie

TABLE OF CONTENTS

1. PROJECT SUMMARY

OVERVIEW

SWITCH ME NOT is an assembly game based on basic platformer games. The goal of this game is to guide the ‘jumpers’ through obstacles and avoid getting hit. The player must jump over obstacles to traverse the environment. The twist within the game is that the player will have to control two (2) ‘jumpers’ simultaneously. The controls of respective jumpers will randomly switch as the player’s score increases.

HOW TO PLAY

There will be assigned control key for each jumper. Initially, the upper and lower jumper’s controls are “arrow up” and “arrow down” respectively. Take note, if the player’s score reaches to a certain value, the controls switches.

Github: https://github.com/mcsalera/Switch-Me-Not-CMSC-131-/blob/master/proj.asm

2. LIST OF PROCEDURES

A. DEFINITION

a. MAIN

- In this procedure, the calls for different screens are displayed: menu, loading and how to play.

b. GET_KEY

- This procedure checks if the player enters a key or not. The keys will either display how to play, loading page or game Title. This is also used to check on what jumper should jump during the game state.

c. _MOVTHIS & _MOVTHIS2

- These procedures move Jumper 1 and Jumper 2 based on key pressed. The key pressed indicates the state of Jumper 1 and Jumper 2.
- Specific keys to be pressed:
    1. (Up)
    2. (Down)
    3. *Note: Keys for Jumpers will be switched throughout the game
- Movement of player:
    1. Jumping
    2. Descending
- Player 1 has three states:
    1. Stationary
    2. Jumping
    3. Descending

d. SHOW_LOADING

- This function displays the loading screen. This iterates until it has reached the end of the screen.

e. CLEAR_SCREEN

- Sets-up the background for title.

f. _CLEAR_SCREEN

- Sets-up the screen for the game proper.

g. CLEAR_SCREEN_BLACK

- Sets-up screens to completely black background.

h. CLEAR_SCREEN1

- Sets-up screens for upper blinking design in the game title screen.

i. CLEAR_SCREEN2

- Sets-up screens for lower blinking design in the game title screen.

j. **CLEAR_SCREEN_UP**
   - Sets-up screens for arrow up blinking design in the how to play screen.

k. **CLEAR_SCREEN_DOWN**
   - Sets-up screens for arrow down blinking design in the how to play screen.

l. **CLEAR_SCREEN_UPPERHALF**
   - Sets-up screens for upper half of blinking ‘GAME OVER’ sign in the game over screen (yellow in color)

m. **CLEAR_SCREEN_LOWERHALF**
   - Sets-up screens for lower half of blinking ‘GAME OVER’ sign in the game over screen (green in color)

n. **SHOW_MENU**
   - This function displays the game title screen which is the portal for other screens: How to Play, Play Game and Exit

o. **SHOW_HOWTOPLAY**
   - This function displays the how to play screen where the description of the game and instructions are displayed.

p. **SHOW_GAMEPROPER**
   - This function displays the actual playing screen and initialization of the variables are placed.

q. **_DISPLAY**
   - Displays players and iterates until game is over

r. **_SCORE_DISPLAY**
   - Displays and checks the score

s. **_SURROUNDINGS**
   - This procedure sets the surroundings of the game proper. It iterates printing the proper placement of the game environment.

t. **_GET_CHAR_AT_CURSOR**
   - This procedure gets a specific character at which the cursor is currently pointing at.

u. **_ERASECUR**
   - This procedure erases the character at the point which the cursor is currently pointing at. This is used in many methods countless of times to simulate a moving object. The constant printing and erasing of characters is what gives the program the illusion of movement.

v. _OBJECTS

- This procedure does multiple things mainly focused on the “objects” or obstructions being used.
    1. Calls _ERASECUR to remove the previous object to simulate movement of first and second obstruction.
    2. Resets the first (below) and second (above) object/ obstruction at a specific point once the object has gone off-screen.
    3. Increases the score of the player once the object/ obstruction has gone off-screen signifying that the player has successfully avoided the object/ obstruction in the path.
    4. Calls _CHECKSCORE to checks for certain key scores indicating when to switch the players.
    5. Randomizes the obstruction to be evaded by the player based on the time.

w. _CHECKSCORE

- This procedure checks whether it is the right time to switch the controls of the Jumpers.

x. _ERASEOBJ & _ERASEOBJ2

- These procedures focuses on erasing the specific obstructions generated for jumper 1 and jumper 2. It erases at the specific coordinates of the obstruction for efficiency. There are two _ERASEOBJ procedures needed because each obstruction of the Jumpers have unique placements.

y. _OBJECTS_MOVE & _OBJECTS_MOVE2

- These procedures contain the specific points of the obstructions to be generated.
- The procedures also move the obstructions after each iteration in which it is called to simulate movement.

z. _PRINT_OBJECT

- Prints character contained in DL at the point which cursor is pointing
- Skips if coordinates are beyond the screen

aa. SHOW_GAMEOVER

- Shows game-over screen which displays the current score and high score. The player can also choose to Play Again, go back to Main Menu or Quit the game.

bb. INPUT_LOOP

- Gets input when a key has been pressed when screens are being shown

cc. DELAY

- Delays movement of jumpers and obstructions to make it easier for the players to evade.

dd._SET_CURSOR

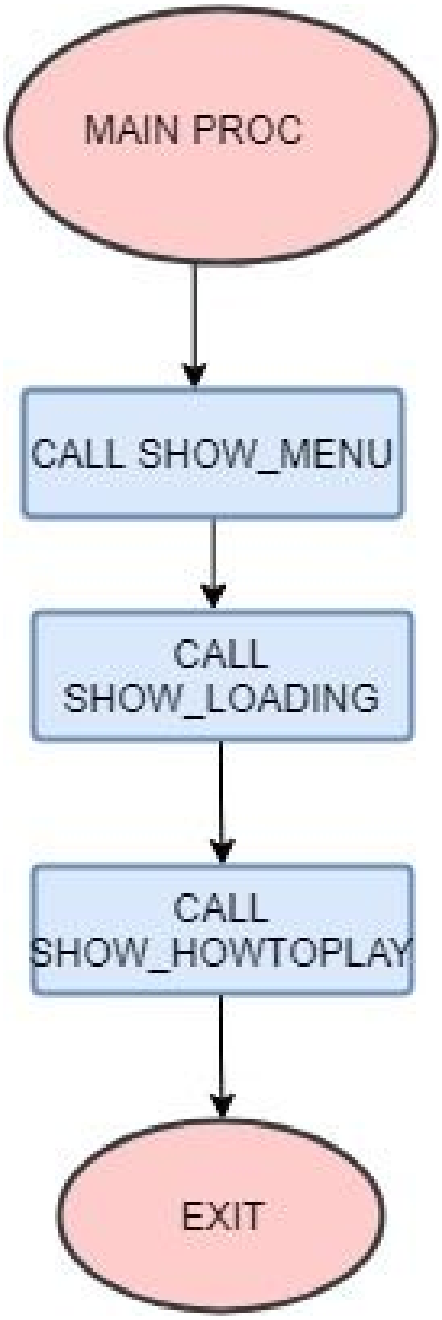
- This procedure sets the cursor at a specific point. This is called whenever something is to be printed or erased.

ee. TERMINATOR

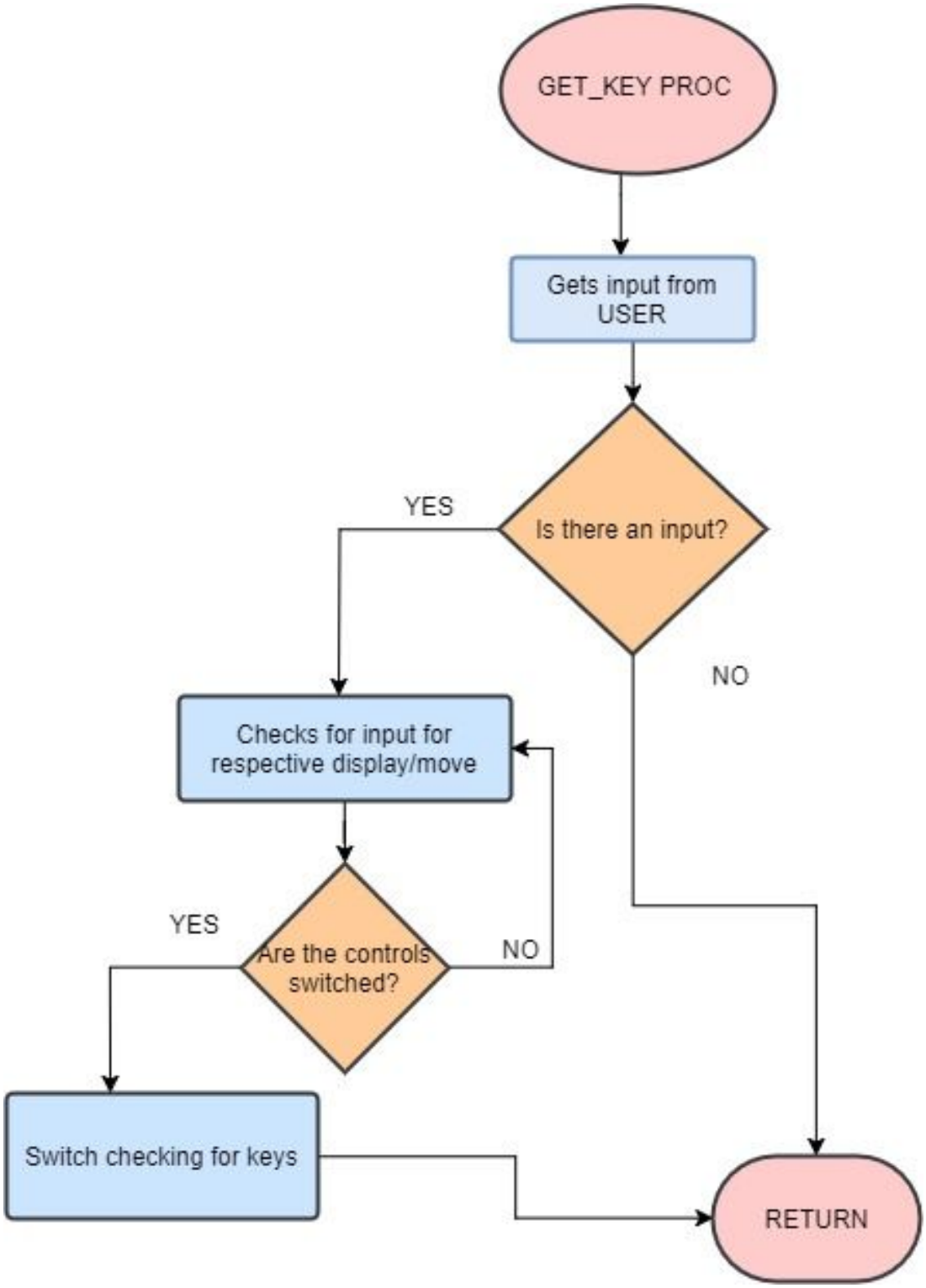
- This procedure terminates the program. This only takes place when the player presses ‘Esc’

B. FLOWCHART

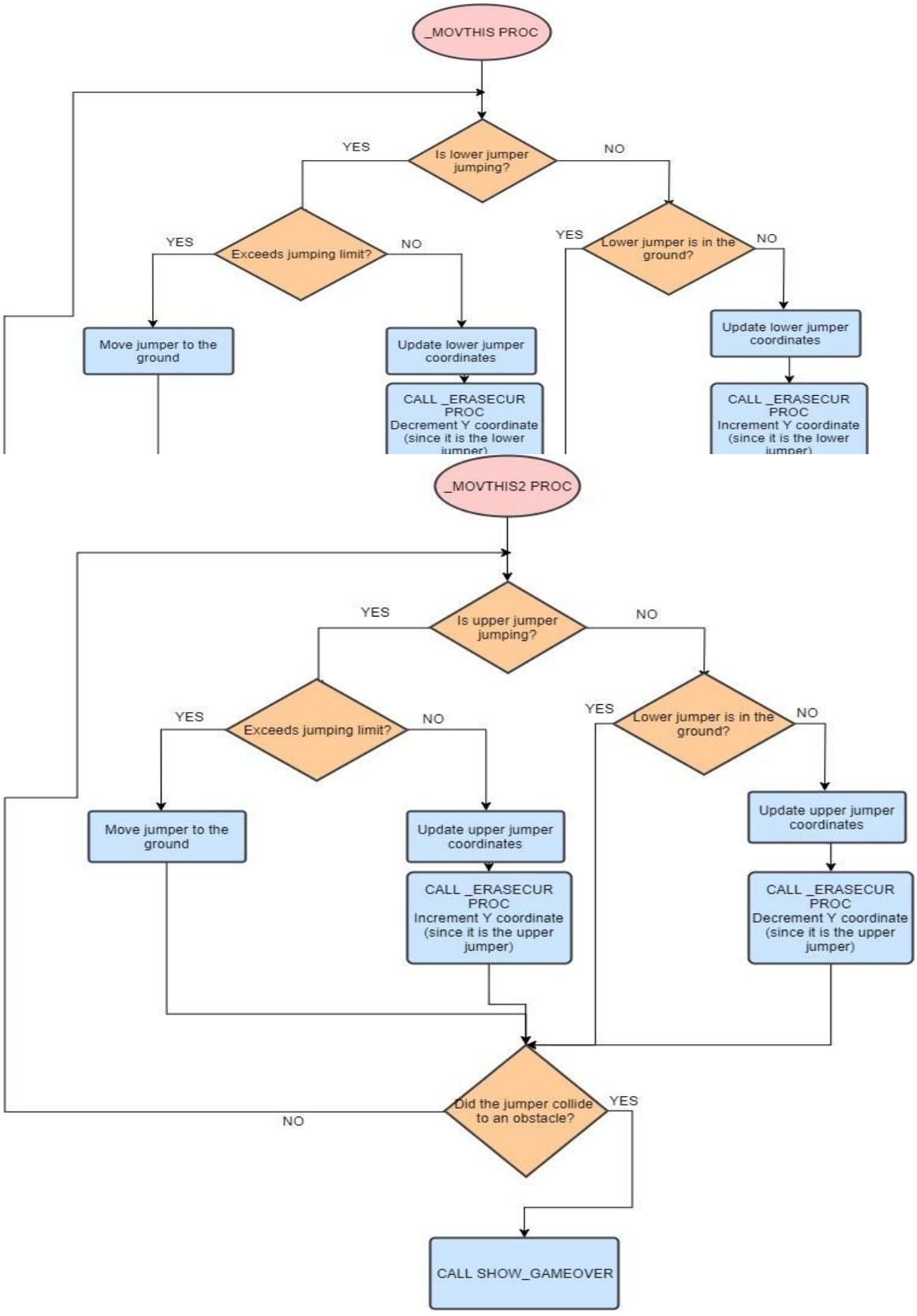
A. MAIN



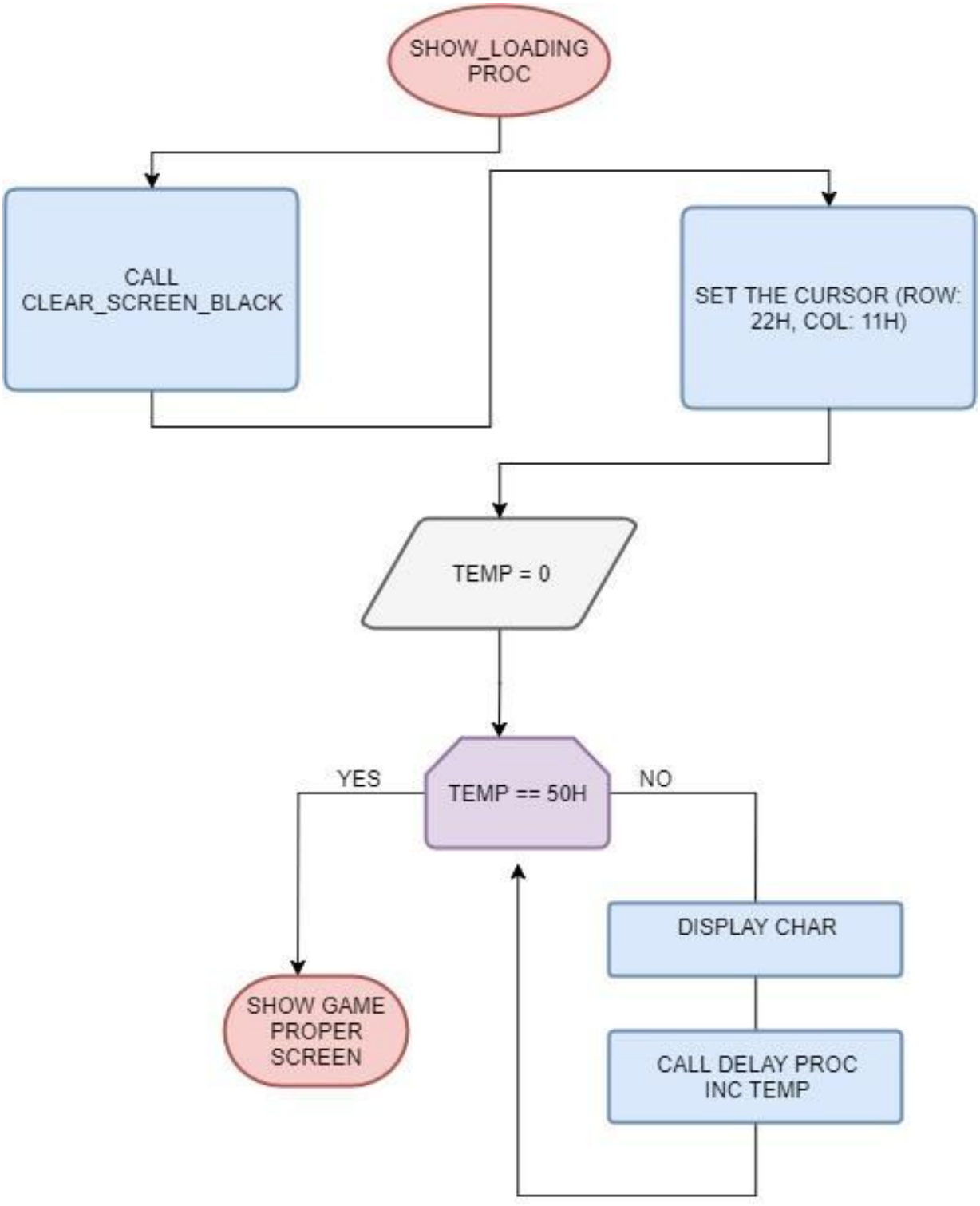
B. GET_KEY


GET_KEY PROC
Gets input from USER
Is there an input?
YES
NO
Checks for input for respective display/move
Are the controls switched?
YES
NO
Switch checking for keys
RETURN

C. _MOVTHIS & _MOVTHIS2

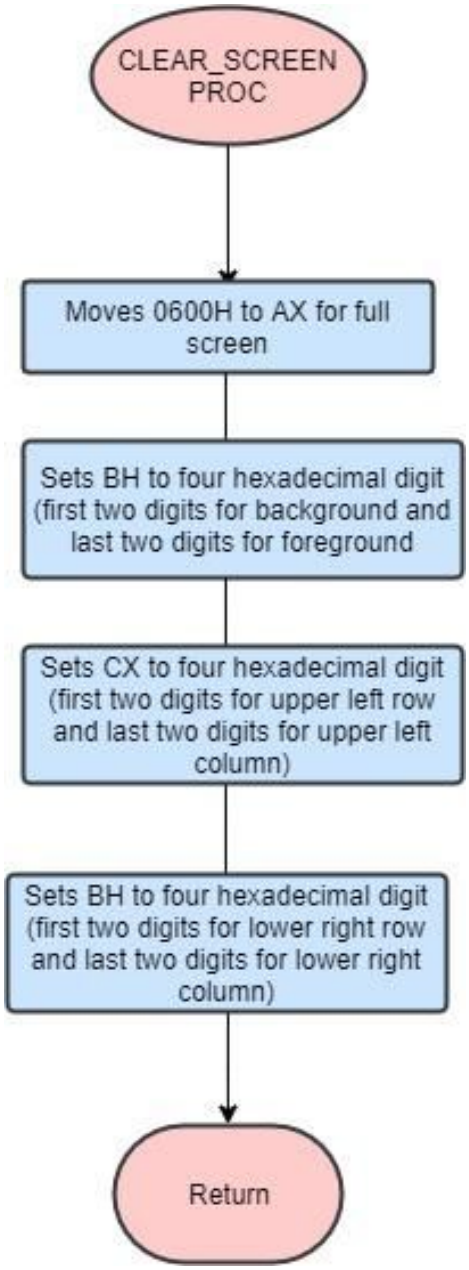


C. SHOW_LOADING

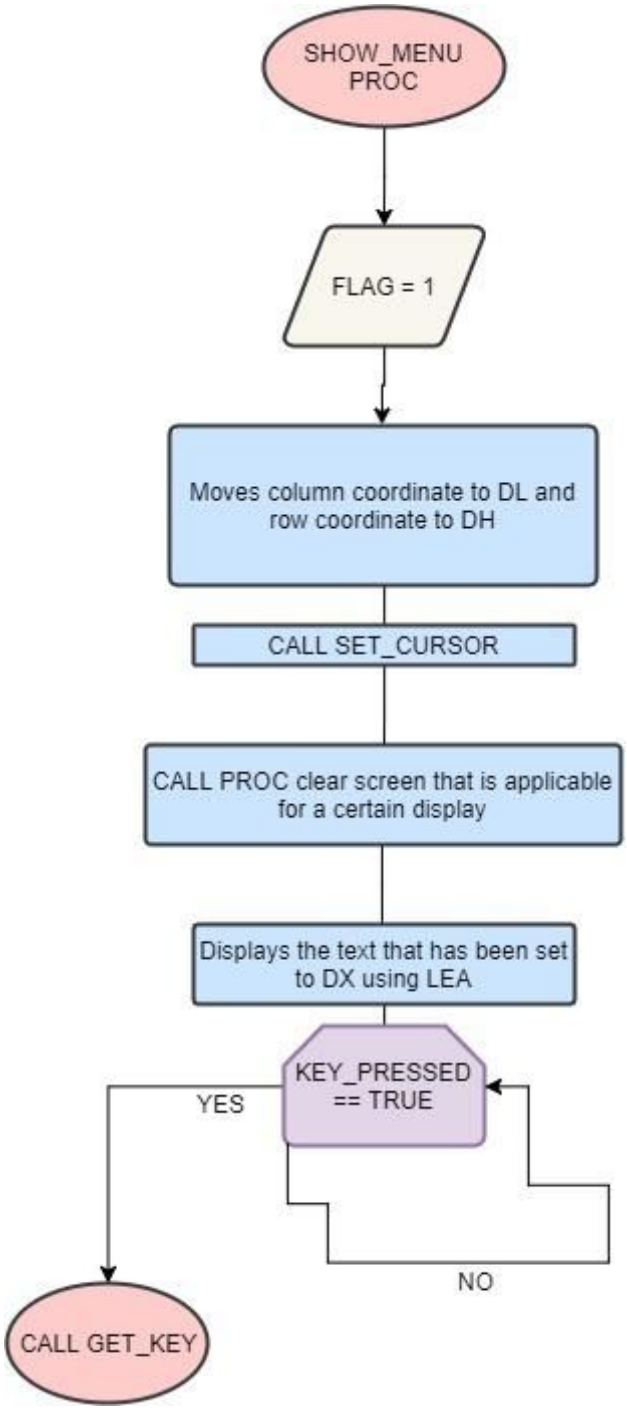


D. CLEAR_SCREEN

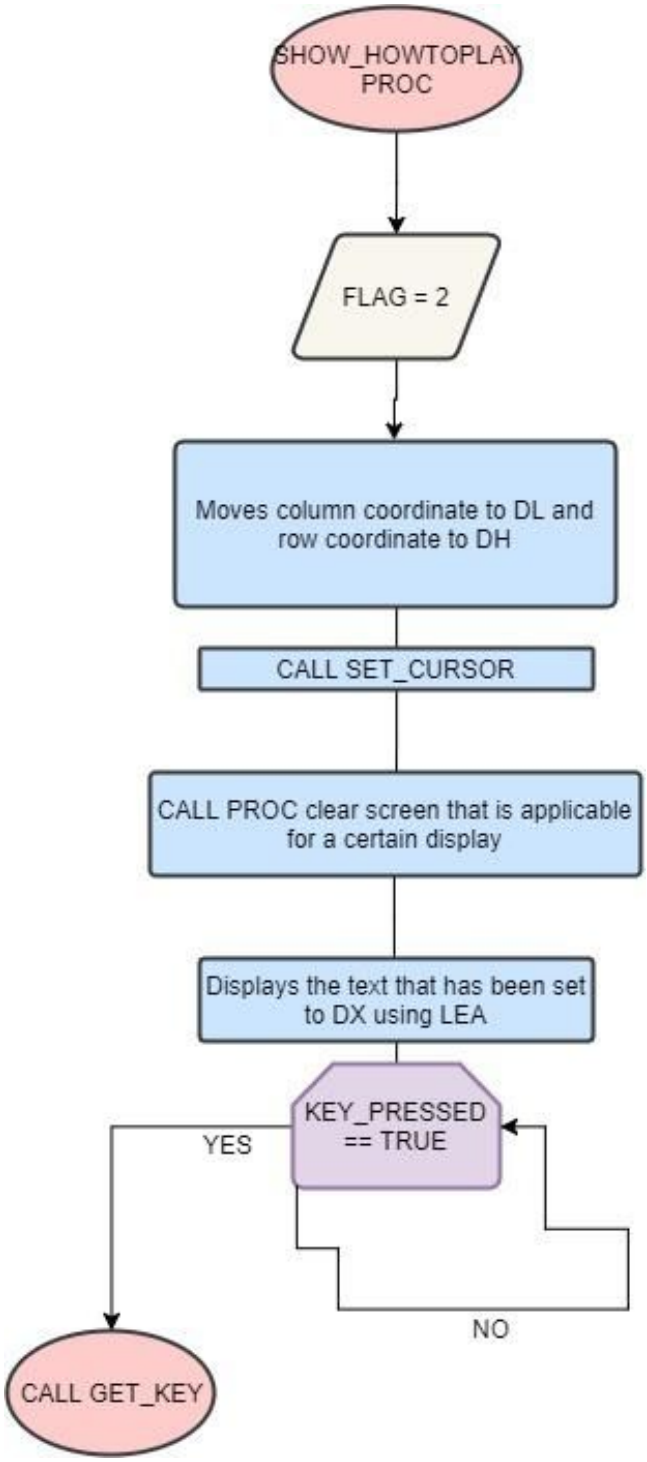
(note: this also works for other clear_screen procedures since only background and foreground color are altered)



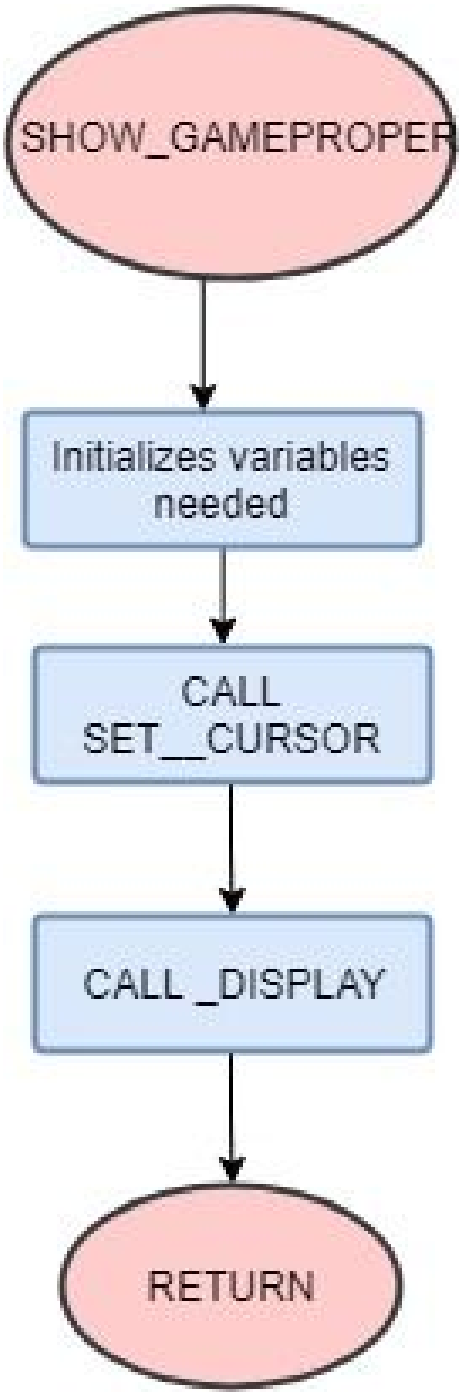
E. SHOW_MENU


SHOW_MENU PROC
FLAG = 1
Moves column coordinate to DL and row coordinate to DH
CALL SET_CURSOR
CALL PROC clear screen that is applicable for a certain display
Displays the text that has been set to DX using LEA
KEY_PRESSED == TRUE
YES
NO
CALL GET_KEY

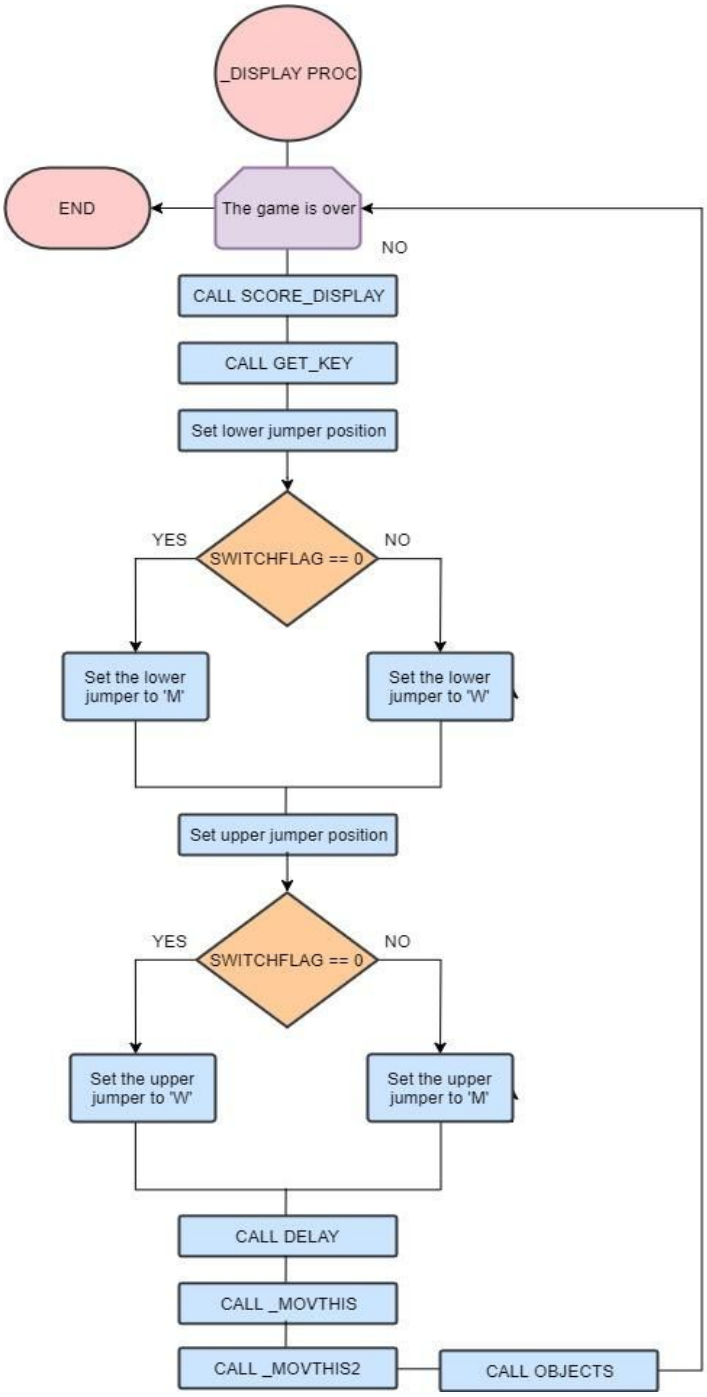
F. SHOW_HOWTOPLAY



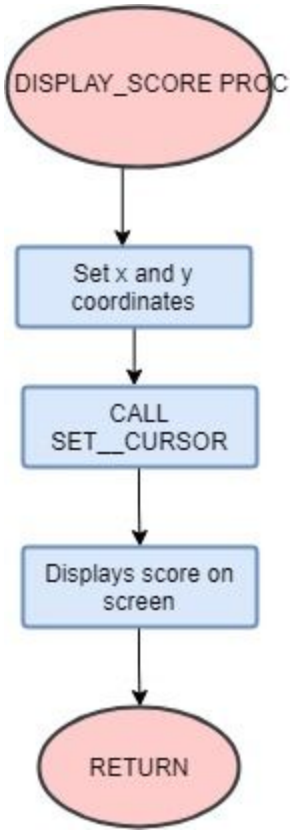
G. SHOW_GAMEPROPER



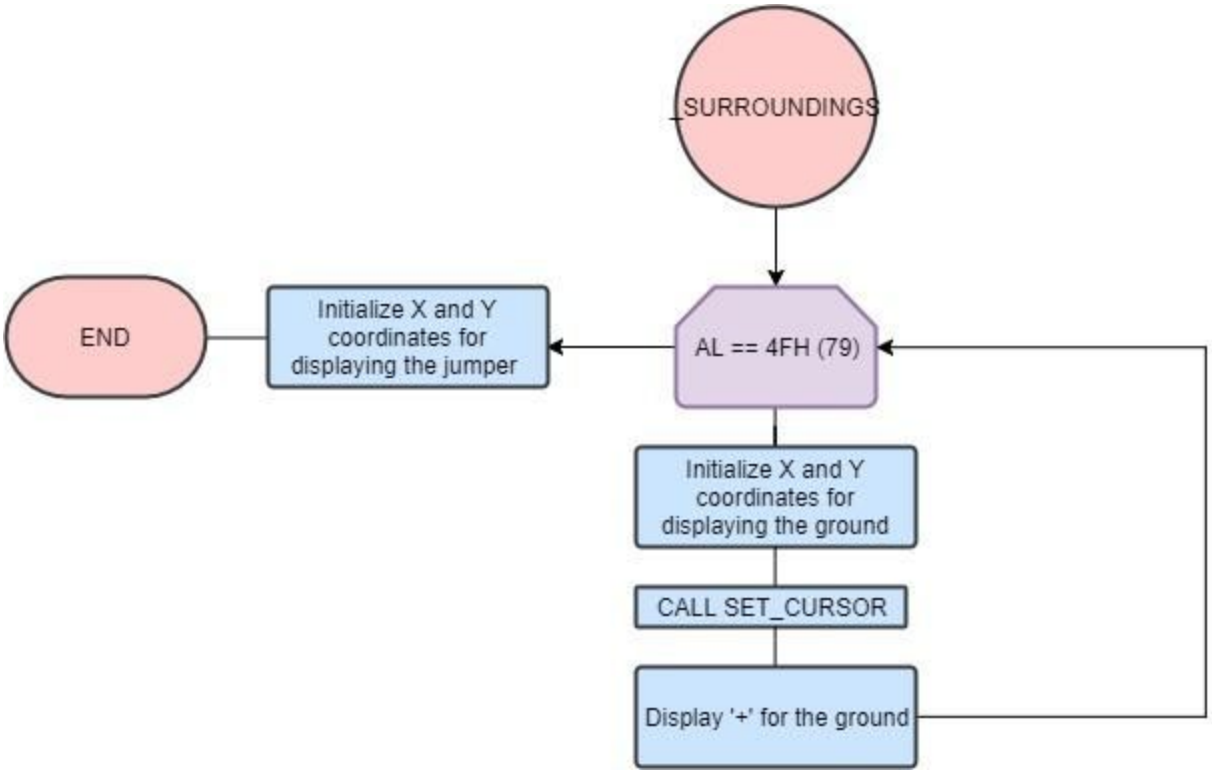
H. _DISPLAY


_DISPLAY PROC
END
The game is over
NO
CALL SCORE_DISPLAY
CALL GET_KEY
Set lower jumper position
YES
SWITCHFLAG == 0
NO
Set the lower jumper to 'M'
Set the lower jumper to 'W'
Set upper jumper position
YES
SWITCHFLAG == 0
NO
Set the upper jumper to 'W'
Set the upper jumper to 'M'
CALL DELAY
CALL _MOVTHIS
CALL _MOVTHIS2
CALL OBJECTS

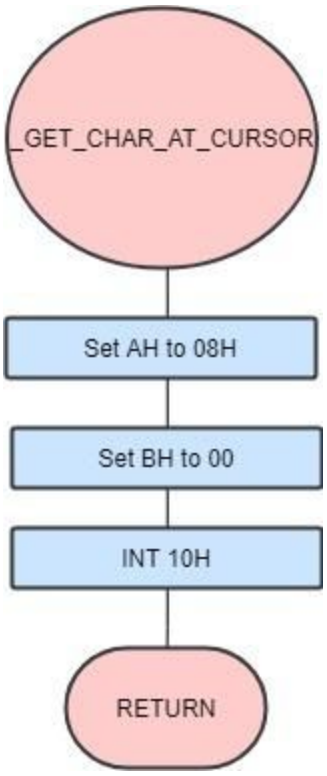
I. _SCORE_DISPLAY



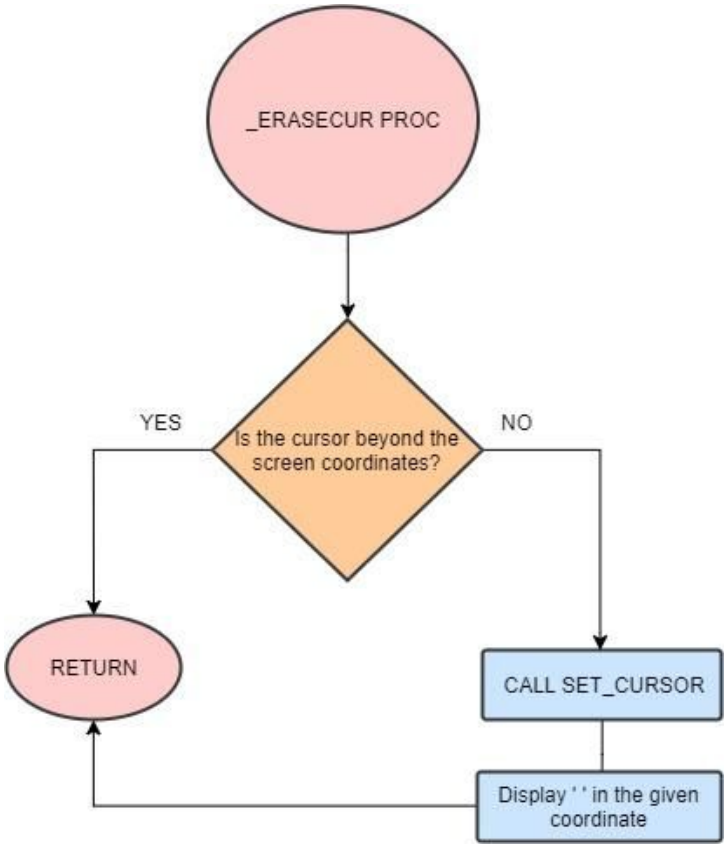
J. _SURROUNDINGS



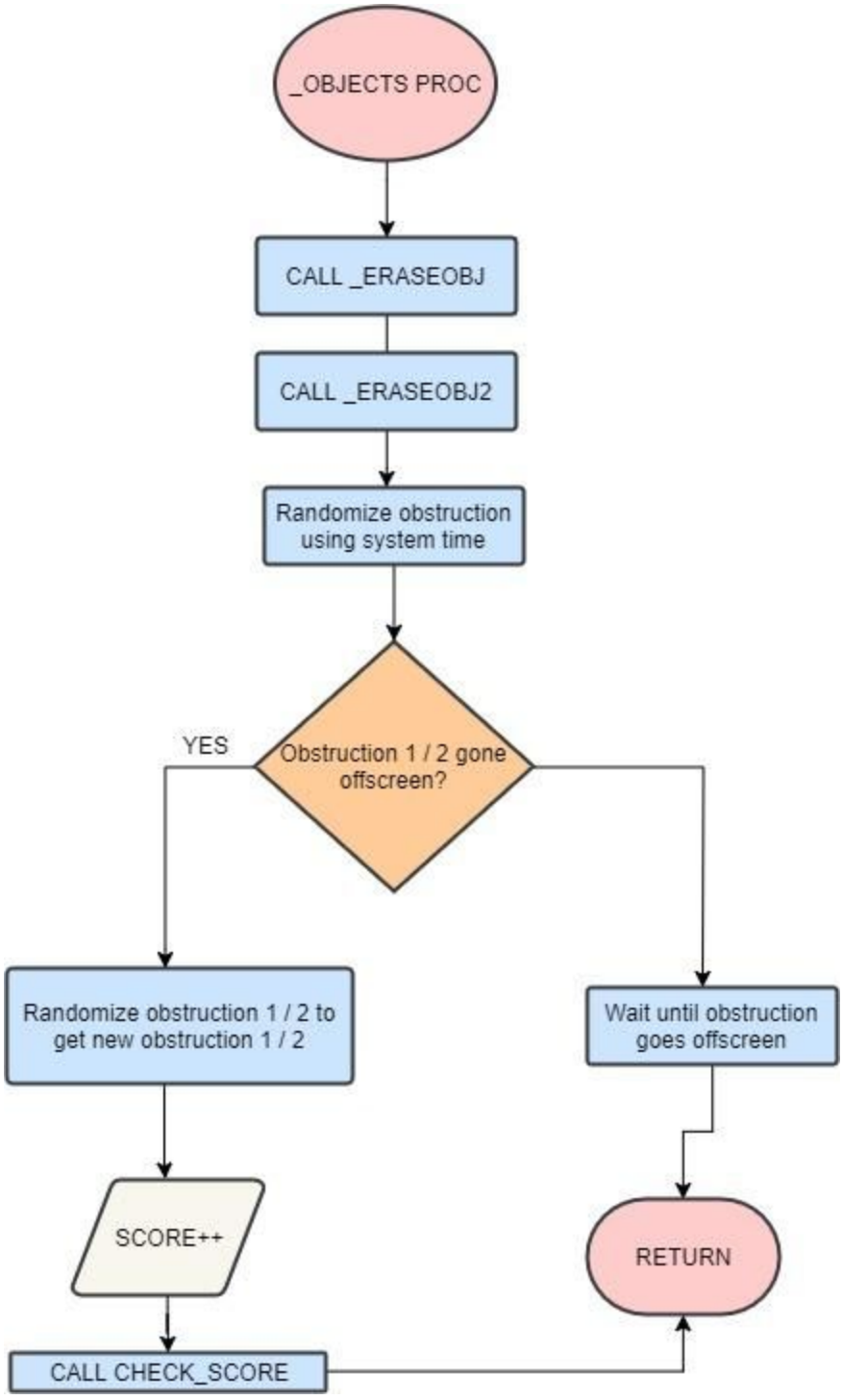
K. _GET_CHAR_AT_CURSOR



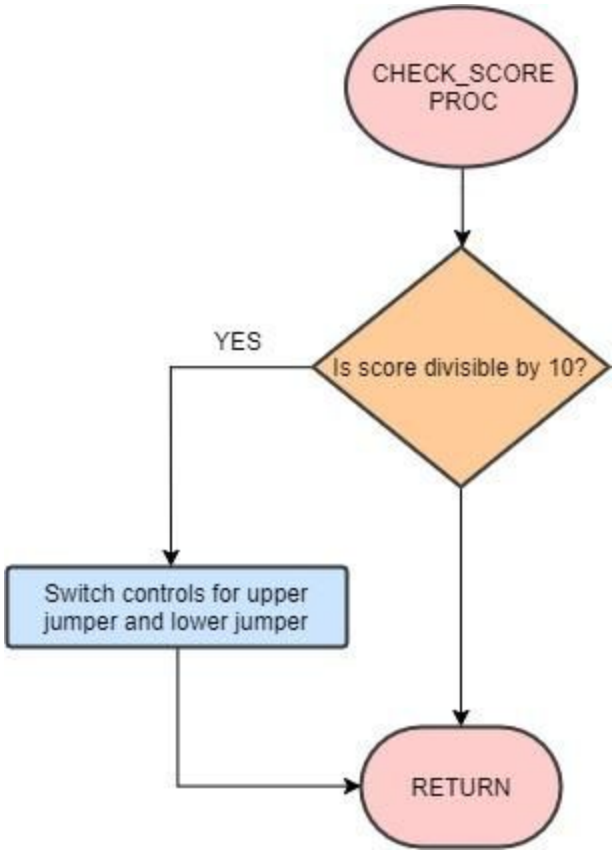
L. _ERASECUR



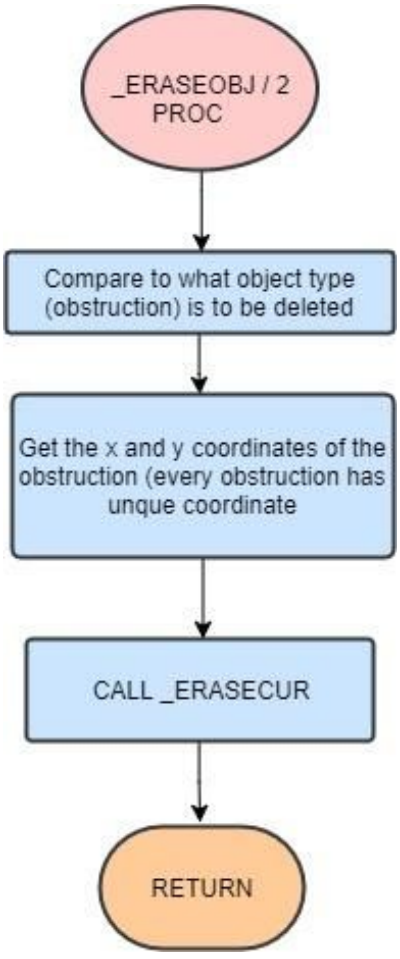
M. _OBJECTS


_OBJECTS PROC
CALL _ERASEOBJ
CALL _ERASEOBJ2
Randomize obstruction using system time
Obstruction 1 / 2 gone offscreen?
YES
Randomize obstruction 1 / 2 to get new obstruction 1 / 2
Wait until obstruction goes offscreen
SCORE++
RETURN
CALL CHECK_SCORE

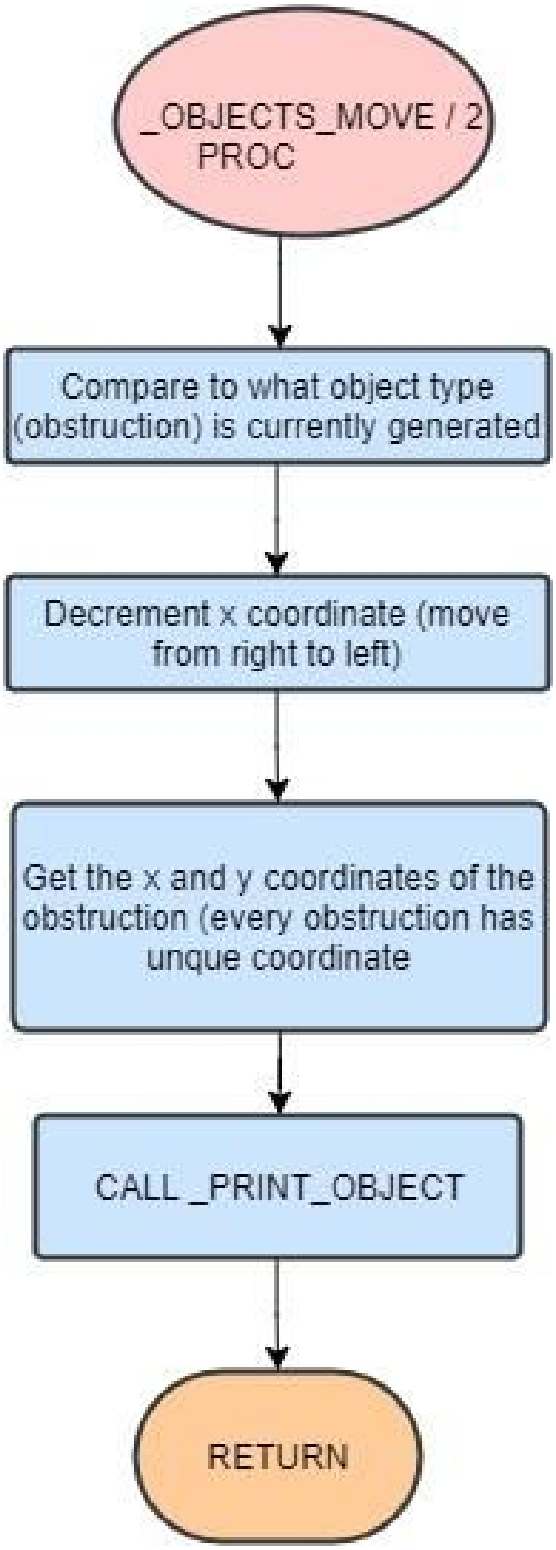
N. _CHECKSCORE



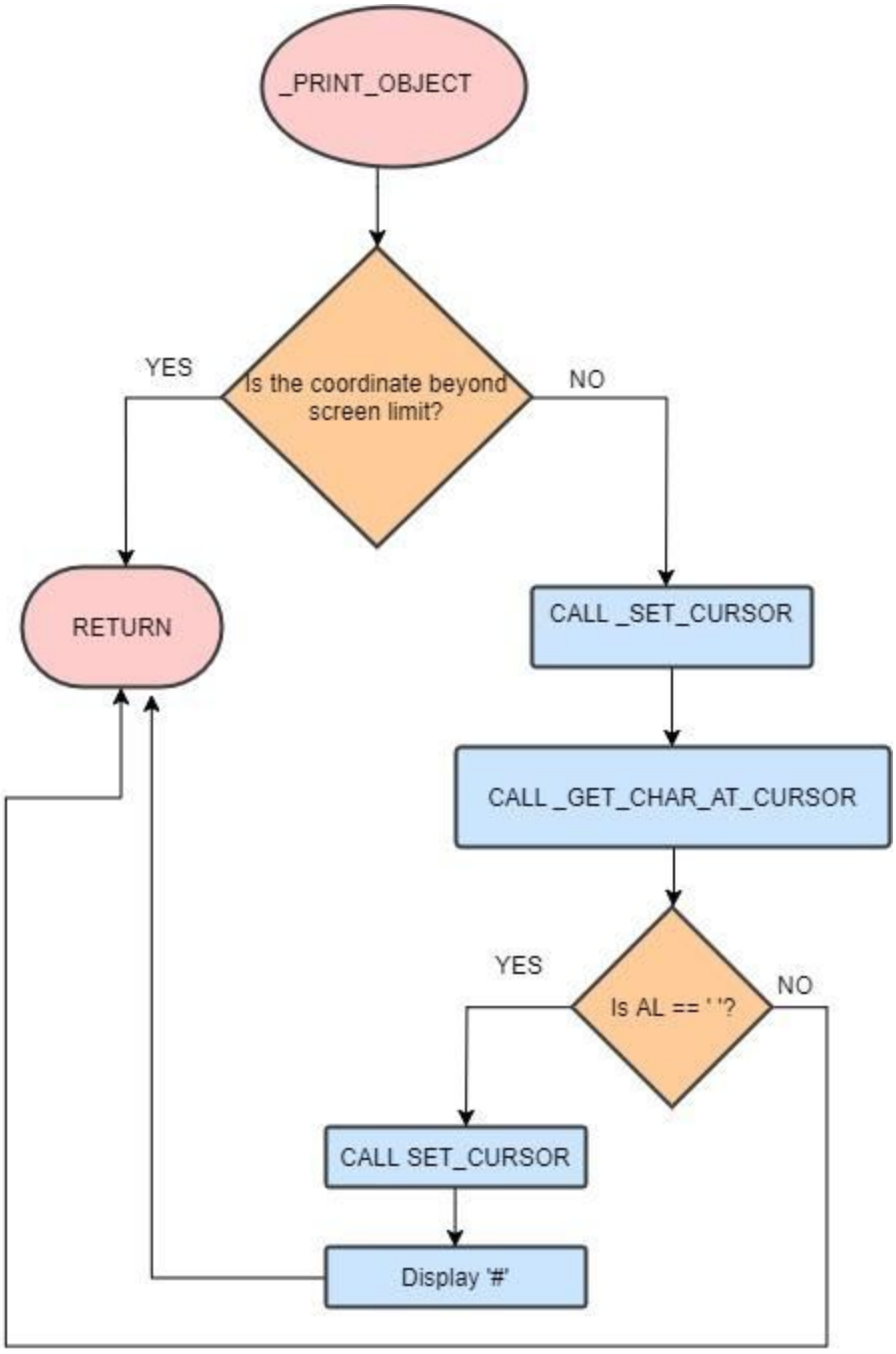
O. _ERASEOBJ & _ERASEOBJ2



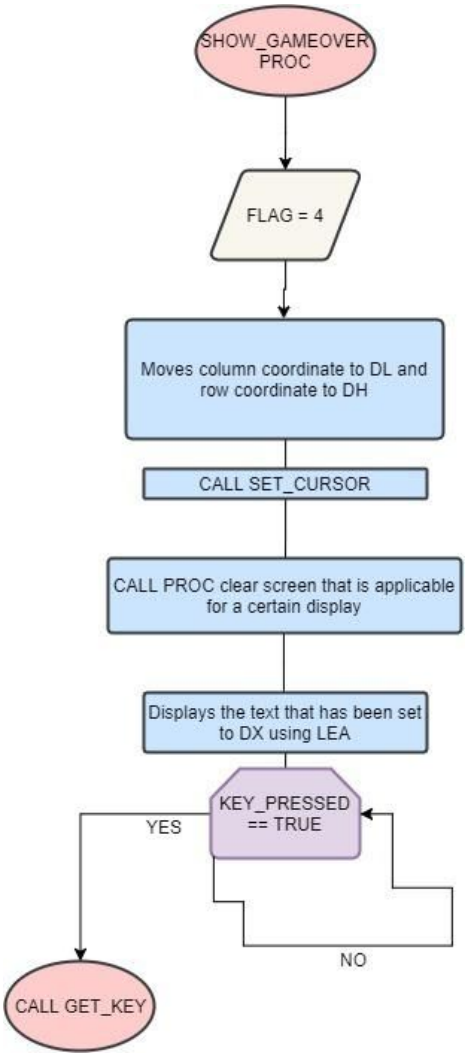
P. _OBJECTS_MOVE & _OBJECTS_MOVE2



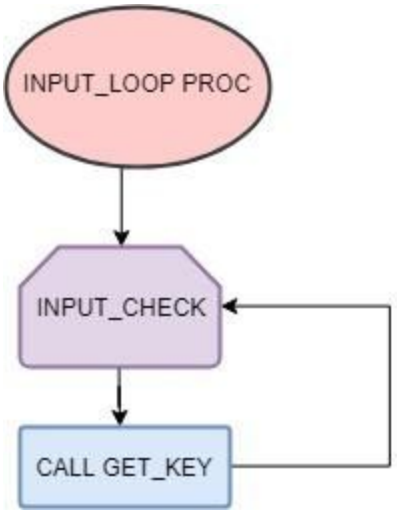
Q. _PRINT_OBJECT



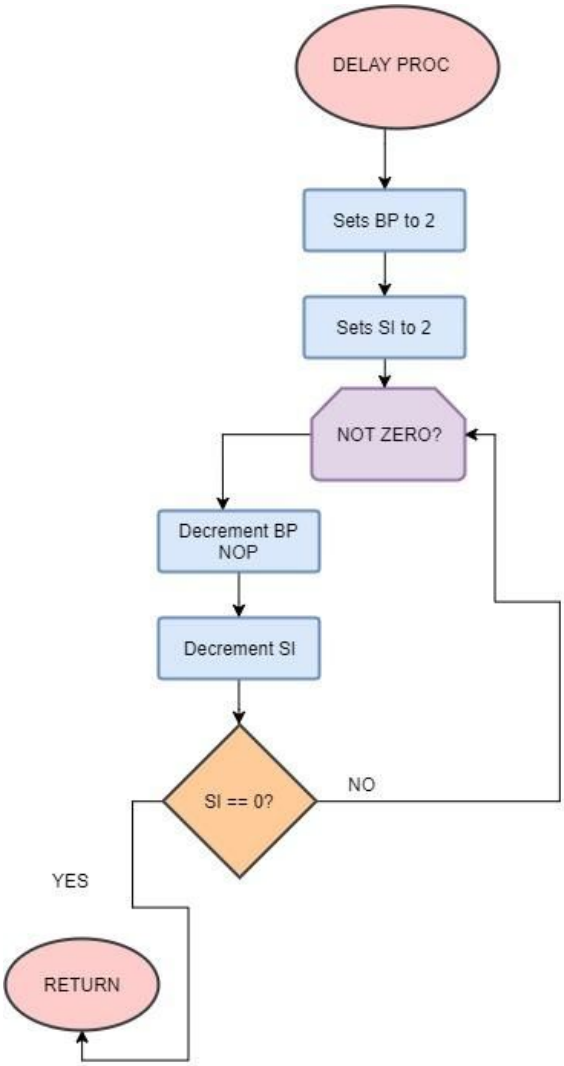
R. SHOW_GAMEOVER



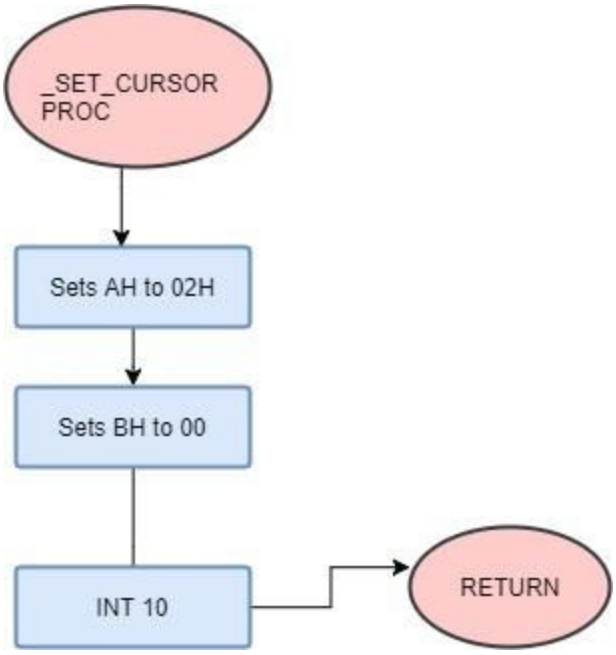
S. INPUT_LOOP



T. DELAY



U. SET_CURSOR



V. TERMINATOR

C. SOURCE CODE OF PROCEDURES

A. MAIN

```
MAIN PROC FAR                          ;Main code
    MOV AX, @data
    MOV DS, AX

        DISPLAY_MENU:                  ;Displays menu
        CALL SHOW_MENU

        DISPLAY_LOADING:               ;Displays loading
        CALL SHOW_LOADING

        DISPLAY_HOWTOPLAY:             ;Displays instruction
        CALL SHOW_HOWTOPLAY

EXIT:
    CALL TERMINATOR


MAIN ENDP
```

B. GET_KEY

```
GET_KEY  PROC  NEAR                    ;Gets for input
    MOV   AH, 01H
    INT   16H

    JZ    __LEAVETHIS                  ;No input recorded

    MOV   AH, 00H                      ;get input  MOV AH, 10H; INT 16H
    INT   16H

    CMP FLAG, 4                        ;Signifies flag
    JE _JMPGAMESCREEN

    CMP AL, 20H
    JE DISPLAY_MENU

    CMP AL, 0DH                            ;Enter key for loading screen
    JE DISPLAY_LOADING

    CMP AL, 1BH                            ;Escape key for exit
    JE EXIT

    CMP AL, 08H                            ;Backspace key for how to play screen
    JE DISPLAY_HOWTOPLAY

    JMP __LEAVETHIS                        ;No significant key press

    _JMPGAMESCREEN:                        ;Game screen controls
        CMP SWITCHFLAG, 1                  ;Checks if switch has been activated
        JE __MODEFLAGONE

        CMP AH, 50H                        ;Controls for players
        JE __JUMP

        CMP AH, 48H
        JE __JUMP2
```

```
    JMP __LEAVETHIS

    __EXTENDER_EXT:
    JMP __LEAVETHIS

    __MODEFLAGONE:              ;Jumps here if switch is activated
    CMP AH, 50H
    JE __JUMP2

    CMP AH, 48H
    JE __JUMP

    JMP __LEAVETHIS

    __JUMP:
        CMP PLAYXLETTER, 'D'  ;disables jump if current jumper is descending
      JE __LEAVETHIS

      CMP PLAYXLETTER, 'j'      ;disables jump if jumper is jumping
      JE __LEAVETHIS

    MOV PLAYXLETTER, 'j'
    JE __EXTENDER_UP
    JMP __LEAVETHIS

    __JUMP2:
        CMP PLAYXLETTER2, 'D'
        JE __LEAVETHIS

        CMP PLAYXLETTER2, 'j'
        JE __LEAVETHIS


    MOV PLAYXLETTER2, 'j'
    JE __UP2

        __LEAVETHIS:
          RET
        _EXTENDER_EXIT:
        CALL SHOW_GAMEOVER
GET_KEY  ENDP
```

```
    __DOWN2:
    CMP PLAY2Y, 2H              ;Compares jump limit
    JE __LANDED2                ;Descends if not landed

    MOV DL, PLAY2X              ;Erases previous lovation
    MOV DH, PLAY2Y
    CALL _ERASECUR

    DEC PLAY2Y
    MOV DL, PLAY2X              ;GAMEOVER
    MOV DH, PLAY2Y
    CALL _SET_CURSOR
    CALL _GET_CHAR_AT_CURSOR
    CMP AL, 23H
    JE _EXTENDER_EXTENDER_EXIT  ;HIT OBJECT

    JMP __EXT
    _EXTENDER_EXTENDER_EXIT:
    JMP _EXTENDER_EXIT

    __LANDED2:                  ;check if colision
    MOV PLAYXLETTER2, 17H
    JMP __EXT2

    __EXT2:
    ret
_MOVTHIS2 ENDP
```

C. _MOVTHIS & _MOVTHIS2

```
_MOVTHIS2  PROC NEAR            ;Moves character

    CMP PLAYXLETTER2, 'j'       ; Jumps Character
    JE __UP2

    CMP PLAYXLETTER2, 'D'       ; DOWN Character
    JE __DOWN2

    JMP __EXT2
    __EXTENDER_UP:
    JMP __UP

    __UP2:
    CMP PLAY2Y, 8H              ;Upper limit for jump
    JE __DESCEND2               ;Descends if upper limit is hit
    MOV DL, PLAY2X
    MOV DH, PLAY2Y
    CALL _ERASECUR              ;Erases previous location

    INC     PLAY2Y              ;Moves to new location

    MOV DL, PLAY2X              ;Checks collision at new location
    MOV DH, PLAY2Y
    CALL _SET_CURSOR
    CALL _GET_CHAR_AT_CURSOR
    CMP AL, 23H
    JE _EXTENDER_EXTENDER_EXIT  ;GAMEOVER if obstacle is hit

    JMP __EXT2

    __DESCEND2:                 ;Descends jumper
    MOV PLAYXLETTER2, 44H
    JMP __EXT2
_MOVTHIS PROC NEAR              ;Moves character
    CMP PLAYXLETTER, 6AH        ; Jumps Character
    JE __UP

    CMP PLAYXLETTER, 44H        ; DOWN Character
    JE __DOWN

    JMP __EXT

    __UP:
    CMP PLAYY, 10H              ;Compares jump limit
    JE __DESCEND                ;Descends if lapas
    MOV DL, PLAYX
    MOV DH, PLAYY
    CALL _ERASECUR

    DEC PLAYY                   ;Moves to new location

    MOV DL, PLAYX               ;Checks for collision
    MOV DH, PLAYY
    CALL _SET_CURSOR
    CALL _GET_CHAR_AT_CURSOR
    CMP AL, 23H
    JE _EXTENDER_EXTENDER_EXIT  ;HIT OBJECT
    JMP __EXT

    __DESCEND:                  ;Movement for jumper
    MOV PLAYXLETTER, 'D'
    JMP __EXT

    __DOWN:                     ;Moves jumper down
    CMP PLAYY, 16H              ;Descends if it has not hit the ground
    JE __LANDED
    MOV DL, PLAYX
    MOV DH, PLAYY
    CALL _ERASECUR
```

```
    INC     PLAYY               ; Moves to new location
    MOV DL, PLAYX                   ;GAMEOVER
    MOV DH, PLAYY
    CALL _SET_CURSOR
    CALL _GET_CHAR_AT_CURSOR
    CMP AL, 23H
    JE _EXTENDER_EXTENDER_EXIT      ;HIT OBJECT
    JMP __EXT
    _EXTENDER_EXTENDER_EXTENDER_EXIT:
    JMP _EXTENDER_EXTENDER_EXIT

    __LANDED:
    MOV PLAYXLETTER, 17H
    JMP __EXT

    __EXT:
    ret
_MOVTHIS ENDP
```

D. SHOW_LOADING

```
SHOW_LOADING    PROC    NEAR            ;Calls loading screen
        CALL CLEAR_SCREEN_BLACK
        MOV     DL, 22H
        MOV     DH, 11
        CALL    _SET_CURSOR
        MOV TEMP, 0

    ITERATE:
        ;set cursor
        MOV     DL, TEMP
        MOV     DH, 12
        CALL    _SET_CURSOR

        ;display char from register
        MOV     AL, 0DBH
        MOV     AH, 02H
        MOV     DL, AL
        INT     21H

        CALL    DELAY

        INC     TEMP
        CMP     TEMP, 50H
        JE      GAME_PROPER ;CHANGE THIS TO ACTUAL GAME

    JMP     ITERATE

    GAME_PROPER:
    CALL SHOW_GAMEPROPER
    RET
SHOW_LOADING ENDP
```

E. CLEAR_SCREEN

```
CLEAR_SCREEN PROC NEAR ;for the title
  MOV AX, 0600H   ;full screen
  MOV BH, 0BH     ;black background (0), cyan foreground (B)
  MOV CX, 0300H   ;upper left row:column (0:0)
  MOV DX, 184FH   ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN ENDP
```

F. _CLEAR_SCREEN

```
_CLEAR_SCREEN PROC  NEAR            ;clears screen
    MOV     AX, 0600H       ; scroll up
    MOV     BH, 00H         ;color
    MOV     CX, 0000H
    MOV     DX, 184FH
    INT     10H

    MOV     AX, 0600H       ; scroll up
    MOV     BH, 71H         ;color
    MOV     CX, 0101H
    MOV     DX, 174EH

    INT     10H
    RET
_CLEAR_SCREEN ENDP
;-------------------------------------------------------------
```

G. CLEAR_SCREEN_BLACK

```
CLEAR_SCREEN_BLACK PROC NEAR
  MOV AX, 0600H   ;full screen
  MOV BH, 0FH     ;black background (0), black foreground (0)
  MOV CX, 0000H   ;upper left row:column (0:0)
  MOV DX, 184FH   ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN_BLACK ENDP
```

H. CLEAR_SCREEN1

```
CLEAR_SCREEN1 PROC NEAR
  MOV AX, 0600H    ;full screen
  MOV BH, 8EH      ;black background with blink (8), yellow foreground (E)
  MOV CX, 0000H    ;upper left row:column (0:0)
  MOV DX, 184FH    ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN1 ENDP
```

I. CLEAR_SCREEN2

```
CLEAR_SCREEN2 PROC NEAR
  MOV AX, 0600H    ;full screen
  MOV BH, 8EH      ;black background with blink (8), yellow foreground (0)
  MOV CX, 1500H    ;upper left row:column (0:0)
  MOV DX, 184FH    ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN2 ENDP
```

J. CLEAR_SCREEN_UP

```
CLEAR_SCREEN_UP PROC NEAR
  MOV AX, 0600H    ;full screen
  MOV BH, 8AH      ;black background with blink (8), green foreground (A)
  MOV CX, 0500H    ;upper left row:column (0:0)
  MOV DX, 1011H    ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN_UP ENDP
```

K. CLEAR_SCREEN_DOWN

```
CLEAR_SCREEN_DOWN PROC NEAR
  MOV AX, 0600H    ;full screen
  MOV BH, 8EH      ;black background with blink(8), yellow foreground (E)
  MOV CX, 0841H    ;upper left row:column (0:0)
  MOV DX, 184EH    ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN_DOWN ENDP
```

L. CLEAR_SCREEN_UPPERHALF

```
CLEAR_SCREEN_UPPERHALF PROC NEAR
  MOV AX, 0600H   ;full screen
  MOV BH, 8EH     ;black background with blink(8), yellow foreground (E)
  MOV CX, 0200H   ;upper left row:column (0:0)
  MOV DX, 054FH   ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN_UPPERHALF ENDP
```

M. CLEAR_SCREEN_LOWERHALF

```
CLEAR_SCREEN_LOWERHALF PROC NEAR
  MOV AX, 0600H   ;full screen
  MOV BH, 8AH     ;black background with blink(8), green foreground (A)
  MOV CX, 0600H   ;upper left row:column (0:0)
  MOV DX, 084FH   ;lower right row:column (24:79)
  INT 10H

  RET
CLEAR_SCREEN_LOWERHALF ENDP
```

N. SHOW_MENU

```
SHOW_MENU   PROC NEAR                ;shows menu
        MOV FLAG, 1
        CALL CLEAR_SCREEN1
        CALL GET_KEY

        MOV   DL, COL
        MOV   DH, ROW
        CALL _SET_CURSOR

        LEA DX, DES1
        MOV AH, 09
        INT 21H

        MOV DL, COL1
        MOV DH, ROW1
        CALL _SET_CURSOR

        CALL CLEAR_SCREEN
        LEA DX, MESSAGE1
        MOV AH, 09
        INT 21H

        LEA DX, MESSAGE6
        MOV AH, 09
        INT 21H

        LEA DX, MESSAGE11
        MOV AH, 09
        INT 21H

        LEA DX, PLAY
        MOV AH, 09
        INT 21H


        MOV   DH, ROW2
        CALL _SET_CURSOR

        CALL CLEAR_SCREEN2

        LEA DX, DES4
        MOV AH, 09
        INT 21H

        CALL INPUT_LOOP
    RET
SHOW_MENU ENDP
```

O. SHOW_HOWTOPLAY

```
SHOW_HOWTOPLAY PROC NEAR                ;shows how to play page
        MOV FLAG, 2
        CALL CLEAR_SCREEN_BLACK
        MOV   DL, COL
        MOV   DH, ROW
        CALL _SET_CURSOR

        LEA DX, HOWTOPLAY1
        MOV AH, 09
        INT 21H

        CALL CLEAR_SCREEN_UP

        LEA DX, UPKEY1
        MOV AH, 09
        INT 21H

        CALL CLEAR_SCREEN_DOWN

        LEA DX, DOWNKEY1
        MOV AH, 09
        INT 21H

        LEA DX, KEY
        MOV AH, 09
        INT 21H

            CALL INPUT_LOOP
    RET
SHOW HOWTOPLAY ENDP
```

P. SHOW_GAMEPROPER

```
SHOW_GAMEPROPER PROC NEAR
    MOV FLAG, 4
    MOV     PLAYX, 10       ;Initial Position player 1 x
    MOV     PLAYY, 16H      ;Initial Position player 1 y

    MOV     PLAY2X, 10      ;Initial Position player 2 x
    MOV     PLAY2Y, 2H      ;Initial Position player 2 y

    CALL _CLEAR_SCREEN      ;Clears screen

    MOV DRAWERX, 0H         ;Sets surroundings x
    MOV DRAWERY, 1H         ;Sets surroundings y
    CALL _SURROUNDINGS      ;Prints surroundings

    MOV DRAWERX, 0H         ;Sets surroundings x
    MOV DRAWERY, 17H        ;Sets surroundings y
    CALL _SURROUNDINGS      ;Prints surroundings

    MOV PLAYXLETTER, 'O'    ;Initializes player 1 status
    MOV PLAYXLETTER2, 'O'   ;Initializes player 2 status

    MOV OBJECTSX, 40H       ;Initializations for obstructions
    MOV OBJECTSY, 16H
    MOV OBJECTSTYPE, 10H

    MOV OBJECTS2X, 40H      ;Initializations for obstructions second
    MOV OBJECTS2Y, 2H
    MOV OBJECTSTYPE2, 30H

    MOV SCORE, 30H          ;Initializing score

    MOV SWITCHFLAG, 1
    CALL _DISPLAY
    RET
SHOW_GAMEPROPER ENDP
```

Q. _DISPLAY

```
_DISPLAY PROC NEAR                      ; Displays players

    __ITERATE:                          ; Iterates until game over
    CALL _SCORE_DISPLAY
    ;Clears everything and gets input
    CALL    GET_KEY

    ;Set cursor                         ; Sets player 1 position
    MOV     DL, PLAYX
    MOV     DH, PLAYY
    CALL    _SET_CURSOR
    ;Display char from register

    CMP SWITCHFLAG, 0                   ;Checker if switch boolean triggered
    JE __UNSWITCHED1

    MOV     DL, 'W'
    INT     21H
    JMP __CONT_FIRST

    __UNSWITCHED1:
    MOV     DL, 'M'
    INT     21H

    __CONT_FIRST:
    MOV     DL, PLAY2X                  ; Sets player 2 position
    MOV     DH, PLAY2Y
    CALL    _SET_CURSOR

    CMP SWITCHFLAG, 0                   ;Checks if switched or unswitched
    JE __UNSWITCHED2

    MOV     DL, 'M'
    INT     21H
```

R. _SCORE_DISPLAY

```
_SCORE_DISPLAY PROC NEAR    ;Displays and checks the score
    MOV DL, 25H
    MOV DH, 12H
    CALL _SET_CURSOR

    MOV DL, SCORE

    MOV AH, 2H
    INT 21H
    RET
_SCORE_DISPLAY ENDP
```

S. _SURROUNDINGS

```
_SURROUNDINGS   PROC    NEAR            ;Sets surroundings of game

    __SURROUNDINGS_ITERATE:             ;sets cursor
    MOV     DL, DRAWERX
    MOV     DH, DRAWERY
    CALL    _SET_CURSOR

    ;Display char from register
    MOV     DL, 2BH                     ;specific character to be diplayed
    INT     21H
    INC     DRAWERX

    MOV AH, 00
    MOV AL, DRAWERX
    CMP AL, 4FH
    JE  __SURROUNDINGS_EXIT
    JMP     __SURROUNDINGS_ITERATE

    __SURROUNDINGS_EXIT:
    MOV     DL, PLAYX
    MOV     DH, PLAYY
_SURROUNDINGS   ENDP
```

T. _GET_CHAR_AT_CURSOR

```
_GET_CHAR_AT_CURSOR PROC    NEAR    ;Gets character at cursor
            MOV     AH, 08H
            MOV     BH, 00
            INT     10H
            RET
_GET_CHAR_AT_CURSOR ENDP
```

U. _ERASECUR

```
_ERASECUR PROC  NEAR                ;Erases character at cursor
    CMP DL, 50H
    JAE __EXIT_ERASECUR
    CMP DL, 0H
    JB __EXIT_ERASECUR

    CALL    _SET_CURSOR

    ;display char from register
    MOV     DL, 20H
    INT     21H

    __EXIT_ERASECUR:
    RET
_ERASECUR ENDP
```

V. _OBJECTS

```
_OBJECTS PROC NEAR                ; move objects
    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    CALL _ERASEOBJ                ;erase current objects

    MOV DL, OBJECTS2X
    MOV DH, OBJECTS2Y
    CALL _ERASEOBJ2               ;erase current objects

    CMP OBJECTSX, 1
    JE __RESET_OBJECTSX
    __AFTER_RESET1:

    CALL _OBJECTS_MOVE

    CMP OBJECTS2X, 1
    JE __EXTENDER_RESET_OBJECTS2X

    __AFTER_RESET2:
    CALL _OBJECTS_MOVE2

    JMP __OBJECTS_EXT

        __RESET_OBJECTSX:         ;Resets position of objects
        MOV AH, 2CH               ;Get the current system time
        INT 21H

        CMP DL, 10H               ;Sets obstacles based on time
        JBE __ZERO

        CMP DL, 14H
        JBE __ONE

        CMP DL, 1EH
        JBE __TWO
```

```
CMP DL, 28H
JBE __THREE

CMP DL, 32H
JBE __FOUR

CMP DL, 3CH
JBE __FIVE

CMP DL, 46H
JBE __SIX

CMP DL, 50H
JMP __SEVEN

__ZERO:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '0'             ;print a number
JMP __random_found
__EXTENDER_RESET_OBJECTS2X:
JMP __RESET_OBJECTS2X

__ONE:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '1'             ;print a number
JMP __random_found

__TWO:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '2'             ;print a number
JMP __random_found
```

```
__THREE:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '3'             ;set sign of obstacle
JMP __random_found

__FOUR:
MOV OBJECTSX, 55H       ;reset back at the far end of the  screen
MOV DL, '4'             ;set sign of obstacle
JMP __random_found

__FIVE:
MOV OBJECTSX, 60H       ;reset back at the far end of the  screen
MOV DL, '5'             ;set sign of obstacle
JMP __random_found

__SIX:
MOV OBJECTSX, 70H       ;reset back at the far end of the  screen
MOV DL, '6'             ;set sign of obstacle
JMP __random_found

__SEVEN:
MOV OBJECTSX, 54H       ;reset back at the far end of the  screen
MOV DL, '7'             ;set sign of obstacle
JMP __random_found

__EIGHT:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '8'             ;set sign of obstacle
JMP __random_found

__NINE:
MOV OBJECTSX, 50H       ;reset back at the far end of the  screen
MOV DL, '9'             ;set sign of obstacle

__random_found:
INC SCORE               ;Score is incremented if object is evaded
```

```
MOV OBJECTSTYPE, DL
MOV DL, 20H
MOV DH, 10H
CALL _SET_CURSOR
MOV DL, OBJECTSTYPE

MOV AH, 2H
INT 21H

CALL _CHECKSCORE        ;Checks score

JMP __AFTER_RESET1

__RESET_OBJECTS2X:
MOV AH, 2CH                  ; get the current system time as randomizer
INT 21H

CMP DL, 10H                ;sets different obstacles based on time
JE  __ZERO2

CMP DL, 14H
JBE __ONE2

CMP DL, 1EH
JBE __TWO2

CMP DL, 28H
JBE __THREE2

CMP DL, 32H
JBE __FOUR2

CMP DL, 3CH
JBE __FIVE2
```

```
CMP DL, 46H
JBE __SIX2

CMP DL, 50H
JMP __SEVEN2

__ZERO2:
MOV OBJECTS2X, 50H      ;reset back at the far end of the  screen
MOV DL, '0'             ;set sign of obstacle
JMP __random_found2

__ONE2:
MOV OBJECTS2X, 50H      ;reset back at the far end of the  screen
MOV DL, '1'             ;set sign of obstacle
JMP __random_found2

__TWO2:
MOV OBJECTS2X, 50H      ;reset back at the far end of the  screen
MOV DL, '2'             ;set sign of obstacle
JMP __random_found2

__THREE2:
MOV OBJECTS2X, 50H      ;reset back at the far end of the  screen
MOV DL, '3'             ;set sign of obstacle
JMP __random_found2

__FOUR2:
MOV OBJECTS2X, 55H      ;reset back at the far end of the  screen
MOV DL, '4'             ;set sign of obstacle
JMP __random_found2

__FIVE2:
MOV OBJECTS2X, 60H      ;reset back at the far end of the  screen
MOV DL, '5'             ;set sign of obstacle
JMP __random_found2
```

```
        __SIX2:
        MOV OBJECTS2X, 60H      ;reset back at the far end of the  screen
        MOV DL, '6'             ;set sign of obstacle
        JMP __random_found2

        __SEVEN2:
        MOV OBJECTS2X, 56H      ;reset back at the far end of the  screen
        MOV DL, '7'             ;set sign of obstacle
        JMP __random_found2

        __random_found2:
        INC SCORE
        MOV OBJECTSTYPE2, DL
        MOV DL, 20H
        MOV DH, 8H
        CALL _SET_CURSOR
        MOV DL, OBJECTSTYPE2

        MOV AH, 2H
        INT 21H

        CALL _CHECKSCORE

        JMP __AFTER_RESET2
    __OBJECTS_EXT:
    ret
_OBJECTS ENDP
```

W. _CHECKSCORE

```
_CHECKSCORE PROC NEAR               ;Checks score for switching controls
    MOV DX, 0
    MOV AH, 0
    MOV AL, SCORE
    MOV BX, 10
    DIV BX

    CMP DX, 0
    JE __SWITCH
    RET

    __SWITCH:
    CMP SWITCHFLAG, 0
    JE __FLAGONE
    MOV SWITCHFLAG, 0
    RET

    __FLAGONE:
    MOV SWITCHFLAG, 1
    RET
_CHECKSCORE ENDP
```

X. _ERASEOBJ & _ERASEOBJ2

```
_ERASEOBJ PROC  NEAR                    ;Erases specific obstacle to be erased for jumper 1
    CMP OBJECTSTYPE, "0"                ;Identifies which obstacle
    JE  __ZERO_ERASE

    CMP OBJECTSTYPE, "1"
    JE  __ONE_ERASE

    CMP OBJECTSTYPE, "2"
    JE  __TWO_ERASE

    CMP OBJECTSTYPE, "3"
    JE  __THREE_ERASE

    CMP OBJECTSTYPE, "4"
    JE  __EXTENDER_FOUR_ERASE

    CMP OBJECTSTYPE, "5"
    JE  __EXTENDER_FIVE_ERASE

    CMP OBJECTSTYPE, "6"
    JE __EXTENDER_SIX_ERASE

    CMP OBJECTSTYPE, "7"
    JMP __EXTENDER_SEVEN_ERASE

    __ZERO_ERASE:                       ;Specifies coordinates to erase
    CALL _ERASECUR
    JMP  __CONT_ERASE

    __ONE_ERASE:                        ;Erase first object
    CALL _ERASECUR
    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    DEC DH
    CALL _ERASECUR
```

```
    JMP   __CONT_ERASE
    __EXTENDER_FOUR_ERASE:

    JMP __FOUR_ERASE
    __EXTENDER_FIVE_ERASE:

    JMP __FIVE_ERASE
    __EXTENDER_SIX_ERASE:

    JMP __SIX_ERASE
    __EXTENDER_SEVEN_ERASE:

    JMP __SEVEN_ERASE

    __TWO_ERASE:                              ;Erase second object
    CALL _ERASECUR

    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    DEC DH
    CALL _ERASECUR

    MOV DL, OBJECTSX
    ;MOV AX, 0
    MOV AL, OBJECTSY
    SUB AL, 2
    MOV DH, AL
    CALL _ERASECUR

    JMP   __CONT_ERASE

    __THREE_ERASE:                            ;Erase third object
    CALL _ERASECUR

    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    DEC DH
```

```
    CALL _ERASECUR

    MOV DL, OBJECTSX
    MOV AL, OBJECTSY
    SUB AL, 2
    MOV DH, AL
    CALL _ERASECUR

    MOV DL, OBJECTSX
    MOV AL, OBJECTSY
    SUB AL, 3
    MOV DH, AL
    CALL _ERASECUR

    JMP  __CONT_ERASE

    __FOUR_ERASE:                          ;Erase fourth object
    CALL _ERASECUR

    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    CALL _ERASECUR

    MOV DH, OBJECTSY
    MOV DL, OBJECTSX
    DEC DL
    CALL _ERASECUR

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 2
    MOV DL, AL
    CALL _ERASECUR
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 4
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 5
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 6
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 7
MOV DL, AL
CALL _ERASECUR
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 8
MOV DL, AL
CALL _ERASECUR

JMP  __CONT_ERASE

__FIVE_ERASE:                          ;Erase fifth object
CALL _ERASECUR

MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _ERASECUR

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 2
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _ERASECUR
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 8
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 9
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 10
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 11
MOV DL, AL
CALL _ERASECUR

JMP  __CONT_ERASE

__SIX_ERASE:                          ;Erase sixth object
CALL _ERASECUR
```

```
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _ERASECUR

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 2
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 12
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 13
MOV DL, AL
CALL _ERASECUR
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 14
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 15
MOV DL, AL
CALL _ERASECUR

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 17H
MOV DL, AL
CALL _ERASECUR

JMP  __CONT_ERASE

__SEVEN_ERASE:                          ;Erase seventh object
CALL _ERASECUR

MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _ERASECUR

MOV DH, OBJECTSY
MOV DL, OBJECTSX
DEC DL
CALL _ERASECUR
```

```
    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 2
    MOV DL, AL
    CALL _ERASECUR

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 3
    MOV DL, AL
    CALL _ERASECUR

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 4
    MOV DL, AL
    CALL _ERASECUR

    JMP __CONT_ERASE

    __CONT_ERASE:
    RET
_ERASEOBJ ENDP
```

Y. _OBJECTS_MOVE & _OBJECTS_MOVE2

```
_OBJECTS_MOVE PROC NEAR
    CMP OBJECTSTYPE, 30H
    JE  __ZERO_MOVE

    CMP OBJECTSTYPE, 31H
    JE  __ONE_MOVE

    CMP OBJECTSTYPE, 32H
    JE  __TWO_MOVE

    CMP OBJECTSTYPE, 33H
    JE  __EXTENDER_THREE_MOVE

    CMP OBJECTSTYPE, 34H
    JE  __EXTENDER_FOUR_MOVE

    CMP OBJECTSTYPE, 35H
    JE  __EXTENDER_FIVE_MOVE

    CMP OBJECTSTYPE, 36H
    JE __EXTENDER_SIX_MOVE

    CMP OBJECTSTYPE, 37H
    JMP __EXTENDER_SEVEN_MOVE

    __ZERO_MOVE:
    DEC OBJECTSX
    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    CALL _PRINT_OBJECT
    JMP  __CONT_MOV

    ;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    __EXTENDER_THREE_MOVE:
    JMP   THREE_MOVE
```

```
__EXTENDER_FOUR_MOVE:
JMP __FOUR_MOVE
__EXTENDER_FIVE_MOVE:
JMP __FIVE_MOVE
__EXTENDER_SIX_MOVE:
JMP __SIX_MOVE
__EXTENDER_SEVEN_MOVE:
JMP __SEVEN_MOVE

__ONE_MOVE:
DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DH                  ;SECOND OBJECT
CALL _PRINT_OBJECT

JMP  __CONT_MOV

__TWO_MOVE:
DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DH                  ;SECOND OBJECT
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV AL, OBJECTSY
SUB AL, 2
```

```
MOV DH, AL
CALL _PRINT_OBJECT

JMP  __CONT_MOV

__THREE_MOVE:
DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DH                    ;SECOND OBJECT
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV AL, OBJECTSY
SUB AL, 2
MOV DH, AL
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV AL, OBJECTSY
SUB AL, 3
MOV DH, AL
CALL _PRINT_OBJECT

JMP  __CONT_MOV

__FOUR_MOVE:
DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
```

```
CALL _PRINT_OBJECT
MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DL                  ;SECOND OBJECT
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 2
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 4
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 5
MOV DL, AL
CALL _PRINT_OBJECT
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 6
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 7
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 8
MOV DL, AL
CALL _PRINT_OBJECT
JMP  __CONT_MOV

__FIVE_MOVE:
DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DL                  ;SECOND OBJECT
CALL _PRINT_OBJECT
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 2
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 8
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 9
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 10
MOV DL, AL
CALL _PRINT_OBJECT
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 11
MOV DL, AL
CALL _PRINT_OBJECT

JMP  __CONT_MOV

__SIX_MOVE:

DEC OBJECTSX
MOV DL, OBJECTSX
MOV DH, OBJECTSY
CALL _PRINT_OBJECT

MOV DL, OBJECTSX
MOV DH, OBJECTSY
DEC DL                  ;SECOND OBJECT
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 2
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 3
MOV DL, AL
CALL _PRINT_OBJECT
```

```
MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 12
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 13
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 14
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 15
MOV DL, AL
CALL _PRINT_OBJECT

MOV DH, OBJECTSY
MOV AX, 0
MOV AL, OBJECTSX
SUB AL, 17H
MOV DL, AL
CALL _PRINT_OBJECT
```

```
    JMP   __CONT_MOV

    __SEVEN_MOVE:
    DEC OBJECTSX
    MOV DL, OBJECTSX
    MOV DH, OBJECTSY

    CALL _PRINT_OBJECT
    MOV DL, OBJECTSX
    MOV DH, OBJECTSY
    DEC DL                    ;SECOND OBJECT
    CALL _PRINT_OBJECT

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 2
    MOV DL, AL
    CALL _PRINT_OBJECT

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 3
    MOV DL, AL
    CALL _PRINT_OBJECT

    MOV DH, OBJECTSY
    MOV AX, 0
    MOV AL, OBJECTSX
    SUB AL, 4
    MOV DL, AL
    CALL _PRINT_OBJECT

    JMP __CONT_MOV
    __CONT_MOV:
    RET

    __EXIT2:
        CALL SHOW_GAMEOVER
    RET
_OBJECTS_MOVE ENDP
```

Z. _PRINT_OBJECT

```
    _PRINT_OBJECT PROC NEAR
    CMP DL, 50H
    JAE __EXIT_PRINT_OBJECT

    CMP DL, 0H
    JB __EXIT_PRINT_OBJECT

    CALL    _SET_CURSOR

    CALL _GET_CHAR_AT_CURSOR
    CMP AL, 20H
    JNE __EXIT3 ;HIT OBJECT

    CALL    _SET_CURSOR

    MOV     DL, 23H
    INT     21H

    __EXIT_PRINT_OBJECT:
    ret
_PRINT_OBJECT ENDP
```

AA. SHOW_GAMEOVER

```
SHOW_GAMEOVER PROC NEAR
    MOV FLAG, 3
    CALL CLEAR_SCREEN_BLACK
    MOV   DL, COL
    MOV   DH, ROW1
    CALL _SET_CURSOR

    CALL CLEAR_SCREEN_UPPERHALF
    CALL CLEAR_SCREEN_LOWERHALF

    LEA DX, END1
    MOV AH, 09
    INT 21H

    MOV   DL, COL
    MOV   DH, 10
    CALL _SET_CURSOR

    LEA DX, LINE
    MOV AH, 09
    INT 21H

    LEA DX, MSGCURRENTSCORE
    MOV AH, 09
    INT 21H

    MOV   DL, COL
    MOV   DH, 11H
    CALL _SET_CURSOR
```

```
    LEA DX, LINE
    MOV AH, 09
    INT 21H

    LEA DX, MSGHIGHSCORE
    MOV AH, 09
    INT 21H

    MOV   DL, COL
    MOV   DH, 18H
    CALL _SET_CURSOR

    LEA DX, MSGPLAYAGAIN
    MOV AH, 09
    INT 21H

    CALL INPUT_LOOP
    RET
SHOW_GAMEOVER ENDP
```

BB. INPUT_LOOP

```
INPUT_LOOP PROC NEAR
    INPUT_CHECK:
        CALL GET_KEY
        JMP INPUT_CHECK
INPUT_LOOP ENDP
```

CC. 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
```

DD. _SET_CURSOR

```
_SET_CURSOR PROC NEAR

  MOV   AH, 02H
  MOV   BH, 00
  INT   10H

  RET
_SET_CURSOR ENDP
```

EE.TERMINATOR

```
TERMINATOR PROC NEAR
    MOV AH, 4CH
    INT 21H
    RET
TERMINATOR ENDP
```

3. SCREENCAP WITH GAME DESCRIPTION

1) **Game Title Screen-** This screen shows the title of the game. The player can choose between three options: learn to play, play the game or quit.

2) **How to Play-** This screen shows the instructions for the game. Pres the arrow up and arrow down key to move the jumpers and avoid obstacles.

3) **Game Over Screen** This screen serves as our game-over screen as well as display for the current and high score. It gives the player the option to exit, play again or go to the menu.

4) **Game Screen** This is our game screen. The two objects on the left are the jumpers which the player controls. The two approaching objects are obstacles that the player must evade by jumping. The score can be seen in the lower center. Players are scored for every obstacle they successfully evade.