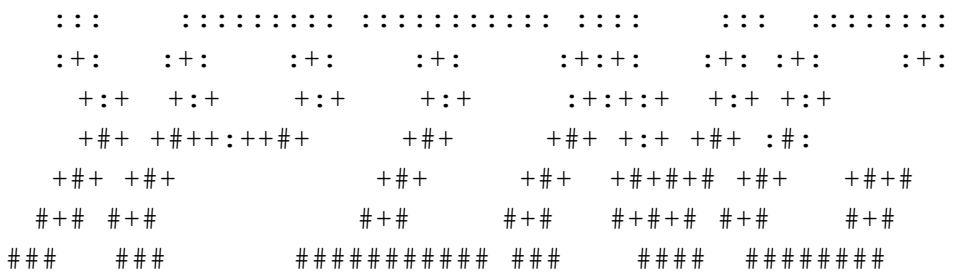


Aiden Justin Bedio

Wina Gen Sotto

BSCS III

TABLE OF CONTENTS

Project Summary / Description 2

List of Procedures 3

Screenshots of Game 8

Project Summary / Description

Lost in the depths of the outer space, you find yourself wandering the vast space in hopes of finding or remembering your way back home. Along the way, you encounter several extraterrestrial entities that try to capture you to satiate their curiosity. With the help of your ship, fight your way against these beings and successfully return to Earth! Ping is a one-player game inspired from the classic Space Impact on Nokia. Shoot your way through countless enemies and bosses who try to stop you while traversing numerous stars and planets to get the highest score. Extra lives could also be obtained from defeating certain enemies.

Github: https://github.com/wgns/Ping

List of Procedures

Key component/s:

```
_This                   equ     es:[bx]
```

- The variable '_This' will be used frequently for easier access to the different structs. The address of the instance of struct that you want to access should be loaded in BX. Then you can now access the different components of the structs.

  Eg.

```
les   bx, myShipAddr
ADD _This.ship_y, 1
SUB _This.ship_x, 1
```

myShipAddr or the address of the myShip (an instance of the SHIP data type) is loaded to bx. BX is then used in _This. To access the components you just use the '.' and the MASM allows to do this. It automatically increments the address given the base address of the struct to know where that certain component is located in the memory.

_This.ship_y is equal to a code like :

es:[myShipAddr].ship_y -> es:[myShipAddr] + 2
(because y is declared second in the SHIP struct)

The following procedures are methods that are specific only for the certain ‘data type’ (struct)

Methods for MENU Data Type

```
MENU                STRUC

  CURRENTFRAME      DB    1
  BGFRAME           DB    1
  SELECTION         DB    1

  X                 DB    30
  Y                 DB    15


  _Display1_        dd    ?
  _Display2_        dd    ?
  _DisplayHi_       dd    ?

MENU                ENDS
```

The following variables will invoke the following procedures when accessed.

Dipslay1 => menuDisplay1
Display2 => menuDisplay2
DisplayHi => displayHi

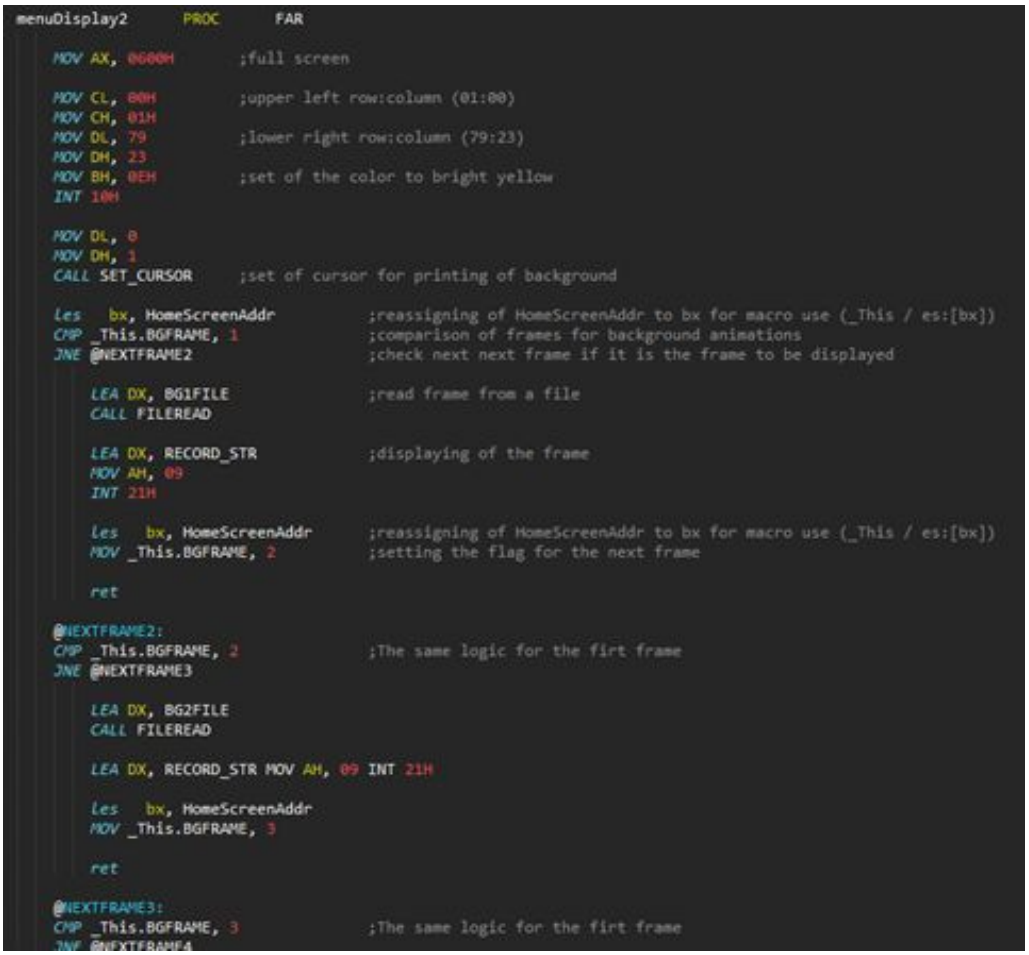
menuDisplay1 PROC FAR

- This procedure is called for displaying the animated background in the main menu.

```
menuDisplay1          PROC      FAR

    MOV DL, 0
    MOV DH, 0
    CALL SET_CURSOR           ;setting of cursor to the upperleft of screen

    les   bx, HomeScreenAddr
    CMP _This.CURRENTFRAME, 1 ;comparison of frames for background animations
    JE @DISPLAY1
    JNE @DISPLAY2

      @DISPLAY1:              ;display frame1

        LEA DX, FRAME1FILE
        CALL FILEREAD         ;read frame1 from a file

        les   bx, HomeScreenAddr      ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
        MOV _This.CURRENTFRAME, 2     ;setting the flag for the next frame

        JMP @PROCEED

      @DISPLAY2:              ;display frame2
        LEA DX, FRAME2FILE    ;read frame2 from a file
        CALL FILEREAD

        les   bx, HomeScreenAddr      ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
        MOV _This.CURRENTFRAME, 1     ;setting the flag for the next frame


        @PROCEED:

        LEA DX, RECORD_STR            ;display of the frame
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr      ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])

        MOV DL, _This.X
        MOV DH, _This.Y
        CALL SET_CURSOR       ;setting of the cursor for the selector

        LEA DX, SELECT        ;display of the selector
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr      ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
        MOV DL, _This.X
        MOV DH, _This.Y
        CALL SET_CURSOR               ;setting of the cursor for the selector

        CALL GET_KEY          ;call of get_key procedure for event listening
        CALL DELAY            ;call of delay for animation purposes

  ret
menuDisplay1     endP
```

menuDisplay2 PROC FAR

- This procedure is called for displaying the animated background in the main menu.



```
menuDisplay2    PROC    FAR

    MOV AX, 0600H       ;full screen

    MOV CL, 00H         ;upper left row:column (01:00)
    MOV CH, 01H
    MOV DL, 79          ;lower right row:column (79:23)
    MOV DH, 23
    MOV BH, 0EH         ;set of the color to bright yellow
    INT 10H

    MOV DL, 0
    MOV DH, 1
    CALL SET_CURSOR     ;set of cursor for printing of background

    les   bx, HomeScreenAddr          ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
    CMP _This.BGFRAME, 1              ;comparison of frames for background animations
    JNE @NEXTFRAME2                   ;check next next frame if it is the frame to be displayed

        LEA DX, BG1FILE               ;read frame from a file
        CALL FILEREAD

        LEA DX, RECORD_STR            ;displaying of the frame
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr      ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
        MOV _This.BGFRAME, 2          ;setting the flag for the next frame

        ret

    @NEXTFRAME2:
    CMP _This.BGFRAME, 2              ;The same logic for the firt frame
    JNE @NEXTFRAME3

        LEA DX, BG2FILE
        CALL FILEREAD

        LEA DX, RECORD_STR MOV AH, 09 INT 21H

        les   bx, HomeScreenAddr
        MOV _This.BGFRAME, 3

        ret

    @NEXTFRAME3:
    CMP _This.BGFRAME, 3              ;The same logic for the firt frame
    JNE @NEXTFRAME4
```

```
        ret


    @NEXTFRAME4:
    CMP _This.BGFRAME, 4              ;The same logic for the firt frame
    JNE @NEXTFRAME5

        LEA DX, BG4FILE
        CALL FILEREAD

        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr
    @NEXTFRAME6:
    CMP _This.BGFRAME, 6              ;The same logic for the firt frame
    JNE @NEXTFRAME7

        LEA DX, BG6FILE
        CALL FILEREAD

        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr
        MOV _This.BGFRAME, 7

        ret

    @NEXTFRAME7:
    CMP _This.BGFRAME, 7              ;The same logic for the firt frame
    JNE @NEXTFRAME8

        LEA DX, BG7FILE
        CALL FILEREAD

        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr
        MOV _This.BGFRAME, 8

        ret

    @NEXTFRAME8:
    CMP _This.BGFRAME, 8              ;The same logic for the firt frame
    JNE @NEXTFRAME9

        LEA DX, BG8FILE
        CALL FILEREAD

        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr
        MOV _This.BGFRAME, 9

        ret

    @NEXTFRAME9:
    CMP _This.BGFRAME, 9              ;The same logic for the firt frame
    JNE @NEXTFRAME10

        LEA DX, BG9FILE
        CALL FILEREAD

        LEA DX, RECORD_STR
        MOV AH, 09
        INT 21H

        les   bx, HomeScreenAddr
        MOV _This.BGFRAME, 1

        ret

    @NEXTFRAME10:

  ret
menuDisplay2      endp
```

displayHi PROC FAR

- This procedure is called for displaying the 'HI SCORE' to the upper right of the screen

```
displayHi     PROC     FAR

  MOV DL, 63
  MOV DH, 0
  CALL SET_CURSOR          ;set of cursor for printing "HI SCORE: " on the upper right

  LEA DX, HISCORE          ;display of "HI SCORE: "
  MOV AH, 09
  INT 21H

  LEA DX, HIFILE           ;loading of file to that will be read

  MOV AX, 3D00H            ;requst open file           ;read only; 01 (write only); 10 (read/write)
  INT 21H

  MOV FILEHANDLE, AX

  MOV AH, 3FH              ;request read record
  MOV BX, FILEHANDLE       ;file handle
  MOV CX, 10               ;record length
  LEA DX, HISCORE_STR      ;address of input area
  INT 21H

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

  LEA DX, HISCORE_STR      ;display of the actual hi score
  MOV AH, 09
  INT 21H

  ret
displayHi     endP
```

Methods for the SHIP Data Type

drawShip PROC FAR

- This procedure is called for displaying the 'ship model' to the upper right of the screen. Works together with DRAW_SHIPS1 procedure. Basically in this procedure is where the comparison to what type of ship will be drawn. Then another function will be called for the actual drawing of the ship

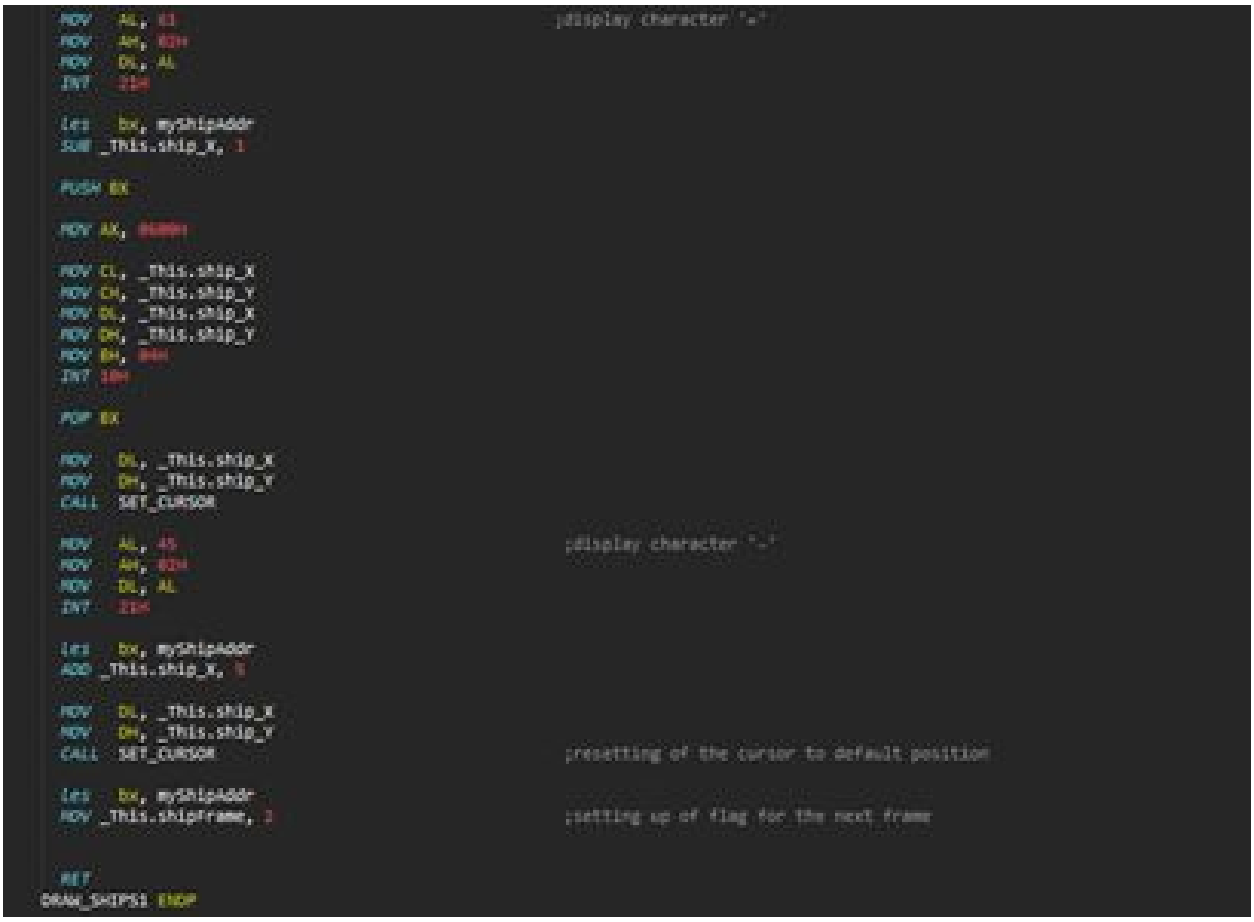
DRAW_SHIPS1 PROC NEAR

- One of the procedures that will actually draw the ship. This is invoked by the drawShip procedure.

```
DRAW_SHIPS1     PROC    NEAR

    ;[Overall this series of repetitive task will form the shape of the ship]

    PUSH BX                     ;push the current ship's address stored in bx because we will use the bx register for
    color purposes

    MOV AX, 0600H               ;full screen

    MOV CL, _This.ship_X        ;upper left row:column (depends:depends)  depends equ ship's current x and y coordinate
    MOV CH, _This.ship_Y
    MOV DL, _This.ship_X        ;upper left row:column (depends:depends)  depends equ ship's current x and y coordinate
    MOV DH, _This.ship_Y
    MOV BH, 0FH
    INT 10H

    POP BX                      ;re-acquire ship's current address

    MOV   DL, _This.ship_X
    MOV   DH, _This.ship_Y
    CALL  SET_CURSOR            ;set cursor to ship's current x and y axis

    MOV   AL, 219               ;display box part of the ship
    MOV   AH, 02H
    MOV   DL, AL
    INT   21H


    les   bx, myShipAddr        ;load myShipAddr to bx for repositioning of cursor
    ADD _This.Ship_x, 1         ;reposition cursor

    PUSH BX                     ;same process as before but for a different character and position

    MOV AX, 0600H

    MOV CL, _This.ship_X
    MOV CH, _This.ship_Y
    MOV DL, _This.ship_X
    MOV DH, _This.ship_Y
    MOV BH, 0FH
    INT 10H

    POP BX

    MOV   DL, _This.ship_X
    MOV   DH, _This.ship_Y
    CALL  SET_CURSOR

    MOV   AL, 91                ; display of the character "["
    MOV   AH, 02H
    MOV   DL, AL
    INT   21H

    les   bx, myShipAddr
    ADD _This.Ship_x, 1

    PUSH BX

    MOV AX, 0600H

    MOV CL, _This.ship_X
    MOV CH, _This.ship_Y
    MOV DL, _This.ship_X
    MOV DH, _This.ship_Y
    MOV BH, 04H
    INT 10H

    POP BX

    MOV   DL, _This.ship_X
    MOV   DH, _This.ship_Y
    CALL  SET_CURSOR

    MOV   AL, 62                ; display of the character ">"
    MOV   AH, 02H
    MOV   DL, AL
    INT   21H

    les   bx, myShipAddr
    SUB _This.Ship_x, 3

    PUSH BX

    MOV AX, 0600H

    MOV CL, _This.ship_X
    MOV CH, _This.ship_Y
    MOV DL, _This.ship_X
    MOV DH, _This.ship_Y
    MOV BH, 0FH
    INT 10H

    POP BX

    MOV   DL, _This.ship_X
    MOV   DH, _This.ship_Y
    CALL  SET_CURSOR

    MOV   AL, 221               ; display of the character slim box
    MOV   AH, 02H
    MOV   DL, AL
    INT   21H

    les   bx, myShipAddr
    SUB _This.Ship_y, 1

    PUSH BX

    MOV AX, 0600H
```

```
MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 193                   ; display of the character wingup
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
ADD _This.ship_y, 2

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 218                   ; display of the character wingdown
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
SUB _This.ship_X, 1

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 124                    ; display of the character double pipe
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
SUB _This.ship_y, 2

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 124                    ; display of the character double pipe
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
ADD _This.ship_y, 1
SUB _This.ship_X, 1


;240 - equivalence
;205 - =
;196 - -

les   bx, myShipAddr            ;load myShipAddr to bx for repositioning of cursor
CMP _This.shipFrame, 1          ;check if what frame of thrusters is being displayed (for animation purposes)
JE @Thruster1                   ;if so then print frame1 of thrusters
```

```
JE @Thruster1                   ;if so then print frame1 of thrusters

@Thruster2:                     ;else print the thrusters2

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 0EH
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 61                    ;display of the character "="
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
SUB _This.ship_X, 1

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 45                    ;display of the character "-"
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
ADD _This.ship_X, 4

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR                ;resetting of the cursor to default position

les   bx, myShipAddr
MOV _This.shipFrame, 1          ;set up flag for the next frame

RET


@Thruster1:

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 04H
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 240                               ;display of character 'equivalence'
MOV   AH, 02H
MOV   DL, AL
INT   21H

les   bx, myShipAddr
SUB _This.ship_X, 1

PUSH BX

MOV AX, 0600H

MOV CL, _This.ship_X
MOV CH, _This.ship_Y
MOV DL, _This.ship_X
MOV DH, _This.ship_Y
MOV BH, 0EH
INT 10H

POP BX

MOV   DL, _This.ship_X
MOV   DH, _This.ship_Y
CALL  SET_CURSOR

MOV   AL, 61                                ;display character '='
```



drawScore PROC FAR

- This procedure is called for displaying the current score on the upper center side of the screen

drawHeart PROC FAR

- This procedure is called for displaying the current health of the ship. It is seen on the upper left side of the screen.

```
drawHeart       PROC      FAR

  MOV DL, 1
  MOV DH, 0
  CALL SET_CURSOR                 ;setting the cursor to the upperleft for the string "HP: "

  MOV   AL, 72  ; "H"             ; display of the character "H"
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  MOV   AL, 80  ; "P"             ; display of the character "P"
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  MOV   AL, 58  ; ":"             ; display of the character ":"
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  les   bx, myShipAddr            ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])

  MOV CX, _This.health            ;setting the loop to the number of health left

  @HEARTLOOP:                     ;start of loop for printing the 'payting/heart'

    PUSH CX                       ;push the current value of CX because the CX register will be used in clearscreen for color purposes

    MOV AX, 0600H                 ;full screen

    MOV CL, _This.health_X        ;upper left row:column (00:depends) / initially on 5
    MOV CH, 00H
    MOV DL, _This.health_X        ;lower right row:column (00:depends)
    MOV DH, 00H
    MOV BH, 04H
    INT 10H                       ; we call this the 'unit clearscreen'

    POP CX                        ;re-acquire value for the CX register for looping

    les   bx, myShipAddr          ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    MOV DL, _This.health_X
    MOV DH, 0
    CALL SET_CURSOR               ;setting of cursor for the printing of heart

    MOV   AL, 03                  ; display of the "payting" symbol
    MOV   AH, 02H
    MOV   DL, AL
    INT   21H

    les   bx, myShipAddr          ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    ADD _This.health_X, 1         ;add x coordinate for the next heart symbol that will be printed

    LOOP @HEARTLOOP               ;end of loop for printing the 'payting/heart'

    les   bx, myShipAddr          ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    MOV _This.health_X, 5         ;resetting the value of the x coordinate to 5

    ;CALL DELAY

  ret
drawHeart       endp
```

drawBomb PROC FAR

- This procedure is called for displaying the current number of 'bombs' a ship currently has. It is seen on the lower left side of the screen.

```
drawBomb      PROC      FAR

  les   bx, myShipAddr              ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
  MOV DL, 1
  MOV DH, 24
  CALL SET_CURSOR                   ;setting the cursor to the lower right of the screen (01:x , 24:y)

  MOV   AL, 237  ; "Ø"              ;display of the character 'Ø' as symbol for bomb
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  MOV   AL, 00  ; "space/' '"       ;display of the character ' '
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  MOV   AL, 120  ; "x"              ;display of the character 'x'
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  MOV   AL, 00  ; "space/' '"       ;display of the character ' '
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  les   bx, myShipAddr              ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])

  ADD _This.Bomb, 48                ;adding character '0' so that it will display its decimal equivalent (To be
  optimized/ will not work greater than 10)
  LEA DX, _This.Bomb                ;display of the current 'bomb' a ship has
  MOV AH, 09H
  INT 21H

  SUB _This.Bomb, 48                ;subtract back '0' so that it will not keep on adding because this function is
  always called (normalization)

  ret
drawBomb      endp
```

Methods for the BULLET Data Type

bSetXY PROC FAR

- This procedure is called every time a 'spacebar' is pressed or simply a bullet is fired. It initializes the initial position of the bullet depending on the current position of the ship.

```
bSetXY      PROC      FAR

  PUSH BX                           ;push the current bullet's address stored in bx because we will use the bx register
  temporarily

  les bx, myShipAddr                ;load myShipAddr to bx to get its current x coordinate

  MOV AL, _This.ship_X              ;get ship's current x coordinate (store in AL)
  ADD AL, 4                         ;add 4 units to make it in front of the ship

  POP BX                            ;re-acquire current bullet's address
  PUSH BX                           ;push it back, same process will happen but this time it is the y-axis

  MOV _This.Bullet_X, AL            ;assign the value stored in AL as the bullet's new X coordinate

  les bx, myShipAddr                ;load myShipAddr to bx to get its current y coordinate

  MOV AL, _This.ship_Y              ;get ship's current y coordinate (store in AL)

  POP BX                            ;re-acquire current bullet's address
  MOV _This.Bullet_Y, AL            ;assign the value stored in AL as the bullet's new Y coordinate

  ret
bSetXY      endp
```

drawBullet PROC FAR

- This procedure is called for displaying the instance of the bullet.

```
drawBullet   PROC      FAR

                       ;this should have some comparison here to which type of bullet is displayed (power
                       ups not yet implemented)

  PUSH BX              ;push the current bullet's address stored in bx because we will use the bx register
  for color purposes

  MOV AX, 0600H        ;full screen

  MOV CL, _This.Bullet_X   ;upper left row:column (depends:depends)  depends equ bullet's current x and y
  coordinate
  MOV CH, _This.Bullet_Y
  MOV DL, _This.Bullet_X   ;lower right row:column (depends:depends) depends equ bullet's current x and y
  coordinate
  MOV DH, _This.Bullet_Y
  MOV BH, 09H          ;set to color blue
  INT 10H              ;unit clearscreen

  POP BX               ;re-acquire current bullet's address

  MOV DL, _This.Bullet_X
  MOV DH, _This.Bullet_Y
  CALL SET_CURSOR      ;mov cursor to the bullet's position

  MOV   AL, 21  ; "§ "
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H            ;display of the bullet

  les bx, myShipAddr   ;load myShipAddr to bx for resetting the cursor to default position

  MOV   DL, _This.ship_X
  MOV   DH, _This.ship_Y
  CALL  SET_CURSOR     ;set the cursor to the ship (default cursor position)

  ret
drawBullet     endp
```

updateBullet PROC FAR

- This procedure is called for updating the current position and state of the bullet. It sets the new X and Y coordinate of the bullet. It also checks here if it has gone outside of the screen (no enemies were hit) or enemies were hit. Either way it resets the state of the instance of the bullet to 'inactive'.

```
updateBullet      PROC      FAR

  CMP _This.Bullet_X, 75          ;checking of the bullets if it goes outside the screen
  JL @KEEPUPDATING                ;if not yet then keep updating

    MOV _This.onair, 0            ;else then set the current bullet's active flag to false

    ret

  @KEEPUPDATING:
  ADD _This.Bullet_X, 5           ;move the bullet by updating its X coordinate

  ret
updateBullet    endp
```

Methods for the INTERCEPTOR Data Type

icSetXY PROC FAR

- This procedure is called every time an instance of the the INTERCEPTOR data type is generated. It initializes the initial position of the enemy.

```
icSetXY       PROC      FAR
  ; should be something random (3 - 21)
  MOV _This.ic_Y, 12          ;setting of the y coordinate of the enemies (temporarily set to a fixed coordinate for
  development purposes)
  ret
icSetXY       endP
```

drawIC PROC FAR

- This procedure is called for displaying the instance of the Interceptors(enemy).

```
drawIC      PROC      FAR

  PUSH BX                     ;push the current bullet's address stored in bx because we will use the bx register
  for color purposes

  MOV AX, 0600H               ;full screen

  MOV CL, _This.ic_X          ;upper left row:column (depends:depends)  depends equ bullet's current x and y
  coordinate
  MOV CH, _This.ic_Y
  MOV DL, _This.ic_X          ;lower right row:column (depends:depends) depends equ bullet's current x and y
  coordinate
  MOV DH, _This.ic_Y
  MOV BH, 0DH                 ;set enemies to color magenta/purple
  INT 10H                     ;unit clearscreen

  POP BX                      ;re-acquire current bullet's address

  MOV DL, _This.ic_X
  MOV DH, _This.ic_Y
  CALL SET_CURSOR             ;mov cursor to the bullet's position

  MOV   AL, 232  ; "Þ"        ;display of the INTERCEPTOR enemy character
  MOV   AH, 02H
  MOV   DL, AL
  INT   21H

  les bx, myShipAddr          ;load myShipAddr to bx for resetting the cursor to default position

  MOV   DL, _This.ship_X
  MOV   DH, _This.ship_Y
  CALL  SET_CURSOR            ;set the cursor to the ship (default cursor position)

  ret
drawIC      endP
```

updateIC PROC FAR

- This procedure is called for updating the current position and state of the Interceptor(enemy). It sets the new X and Y coordinate of the Interceptor. It also checks here if it has gone outside of the screen (ship was not hit) or this was not hit by a bullet. Either way it resets the state of the instance of the Interceptor(enemy) to 'inactive'.

```
updateIC      PROC      FAR

  CMP _This.ic_X, 5             ;checking of the bullets if it goes outside the screen
  JG @KEEPUPDATINGIC            ;if not yet then keep updating

    MOV _This.ic_X, 79          ;reset current enemies default x coordinate to '79'
    MOV _This.active, 0         ;else then set the current bullet's active flag to false

    ret

  @KEEPUPDATINGIC:
  SUB _This.ic_X, 1             ;move the bullet by updating its X coordinate

  ret
updateIC      endP
```

HELPER METHODS

CLEAR_SCREEN PROC NEAR

- This procedure called every time a screenwide clear screen is invoked

DELAYEVENT PROC NEAR

- This procedure is called everytime a new frame is displayed (For animation purposes). DELAYEVENT is the timing during actual gameplay.

DELAY PROC NEAR

- This procedure is called everytime a new frame is displayed (For animation purposes). DELAY is the timing during main menu.

FILEREAD PROC NEAR

- This procedure is called for reading the certain files that are needed during run time. (FRAMES and HI SCORE)

DISPLAY_TIMER PROC NEAR

- This is for debugging purposes only. (A timer/counter of some sort)

SET_CURSOR PROC NEAR

- This procedure is called for setting the position of the cursor.

GET_KEY PROC NEAR

- This method is called as listener for keyboard events

MAIN MENU METHODS

START PROC FAR

- The start of the GAME program.

```
START PROC FAR
  MOV AX, @data
  MOV DS, AX


  @MAINMENU:

    CALL CLEAR_SCREEN                   ;calling of clearscreen

    les   bx, HomeScreenAddr            ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
    _DISPLAY1                           ;display of the background for MAINMENU


    CMP KEY_INPUT, 00                   ;check if there was input
    JNE @ISACTION                       ;if so then check what action

        JMP @MAINMENU                   ;else loop back

    @ISACTION:
    CALL ACTION                         ;call the action button

    JMP @MAINMENU                       ;loop back

  MOV AH, 4CH
  INT 21H

START ENDP
```

ACTION PROC NEAR

- This procedure is called every time a key was pressed. Here it checks and execute the different functions of each pressable buttons in the game (main menu actions).

ACTUAL GAME METHODS

MAIN_GAME PROC NEAR

- The main game loop procedure. This procedure is invoked every time 'play game' is selected from the main menu screen.

```
MAIN_GAME PROC NEAR

  @FIXEDUPDATE:

    CALL CLEAR_SCREEN                   ;calling of clearscreen

    les   bx, HomeScreenAddr            ;reassigning of HomeScreenAddr to bx for macro use (_This / es:[bx])
    _DISPLAY2                           ;display of the background for MAINMENU
    _DISPLAYHI                          ;display of the HI SCORE
    ;CALL DISPLAY_TIMER                 ;display of a counter (appears on the bottom right[for debuggin
    purposes])

    les   bx, myShipAddr                ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    _DRAWBOMB                           ;display of the current bomb a ship has (bottom left)
    les   bx, myShipAddr                ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    _DRAWHEART                          ;display of the current health a ship has (upper left)
    les   bx, myShipAddr                ;reassigning of myShipAddr to bx for macro use (_This / es:[bx])
    _DRAWSHIP                           ;display the ship
    _DrawScore                          ;display the current score

    CALL GEN                            ;generate enemies (psuedo generate)
    CALL DRAWICS                        ;display generated enemies
    CALL UPDATEICS                      ;update generated enemies' position and status

    CALL DRAWBULLETS                    ;display generated bullets
    CALL UPDATE_BULLETS                 ;update generated bullets' position and status

    CALL GET_KEY                        ;call of get_key procedure for event listening
    CALL DELAYEVENT                     ;call of delay for animation purposes

    CMP KEY_INPUT, 00                   ;check if there was input
    JNE @ISEVENT                        ;if so then check what event

    JMP @FIXEDUPDATE                    ;loop back (main game loop)

    @ISEVENT:
      CALL EVENT                        ;call event procedure (event listening)


      JMP @FIXEDUPDATE                  ;loop back (main game loop)

  RET
MAIN_GAME ENDP
```

EVENT PROC NEAR

- This procedure is called every time a key is pressed. Here it checks and execute the different functions of each pressable buttons in the game (actual game actions).

```
EVENT    PROC   NEAR

  CMP KEY_INPUT, 01H                    ;check if input is 'ESC'
  JE @EXITPRG1                          ; if so then exit the program
  JNE @SHIPLISTENER                     ; else jmp to ship action listener

    @EXITPRG1:                          ;exit program
    MOV AH, 4CH
    INT 21H

  @SHIPLISTENER:

    Les   bx, myShipAddr                ;load myShipAddr to bx for resetting the cursor to default position

    CMP KEY_INPUT, 48H                  ;check if input is up
    JE @MOVEUP

    CMP KEY_INPUT, 50H                  ;check if input is down
    JE @MOVEDOWN

    CMP KEY_INPUT, 4BH                  ;check if input is left
    JE @MOVELEFT

    CMP KEY_INPUT, 4DH                  ;check if input is right
    JE @MOVERIGHT

    CMP KEY_INPUT, 39H                  ;check if input is space
    JE @FIRE
    JNE @PORTAL

                                        ;needs one more key for the 'bomb' (not yet implemented)
    @MOVEUP:
      DEC _This.ship_Y                  ;move the ship upwards
      JMP @PORTAL                       ;flush key input (default 00)

    @MOVEDOWN:
      INC _This.ship_Y                  ;move the ship downwards
      JMP @PORTAL                       ;flush key input (default 00)

    @MOVELEFT:
      DEC _This.ship_X                  ;move the ship backwards
      JMP @PORTAL                       ;flush key input (default 00)

    @MOVERIGHT:
      INC _This.ship_X                  ;move the ship forward
      JMP @PORTAL                       ;flush key input (default 00)

    @FIRE:
      CALL FIREBULLET                   ;call FIREBULLET procedure for generating (psuedo-generate) a bullet


  @PORTAL:
  MOV KEY_INPUT, 00                     ;flush key input (default 00)

  ret
EVENT endp
```

FIREBULLET PROC NEAR

- Invoked every time a ‘spacebar’ is pressed. It generates(pseudo generate) an instance of the bullet.

DRAWBULLETS PROC NEAR

- This procedure is called every cycle. This determines which bullets are to be displayed base from bullet’s flag called onAir which determines if it was fired. It then invoke the drawBullet procedure of the instance of that bullet if it was fired.

UPDATE_BULLETS PROC NEAR

- This procedure is also called every cycle. This determines which bullets are to be updated(update only if instance of bullet is fired/onAir). It then invokes the updatebullet procedure of the instance of that bullet if it was fired.

GENERATEIC PROC NEAR

- This procedure is called every time an enemy is to be generated. It generates(pseudo generate) an instance of the Interceptor.

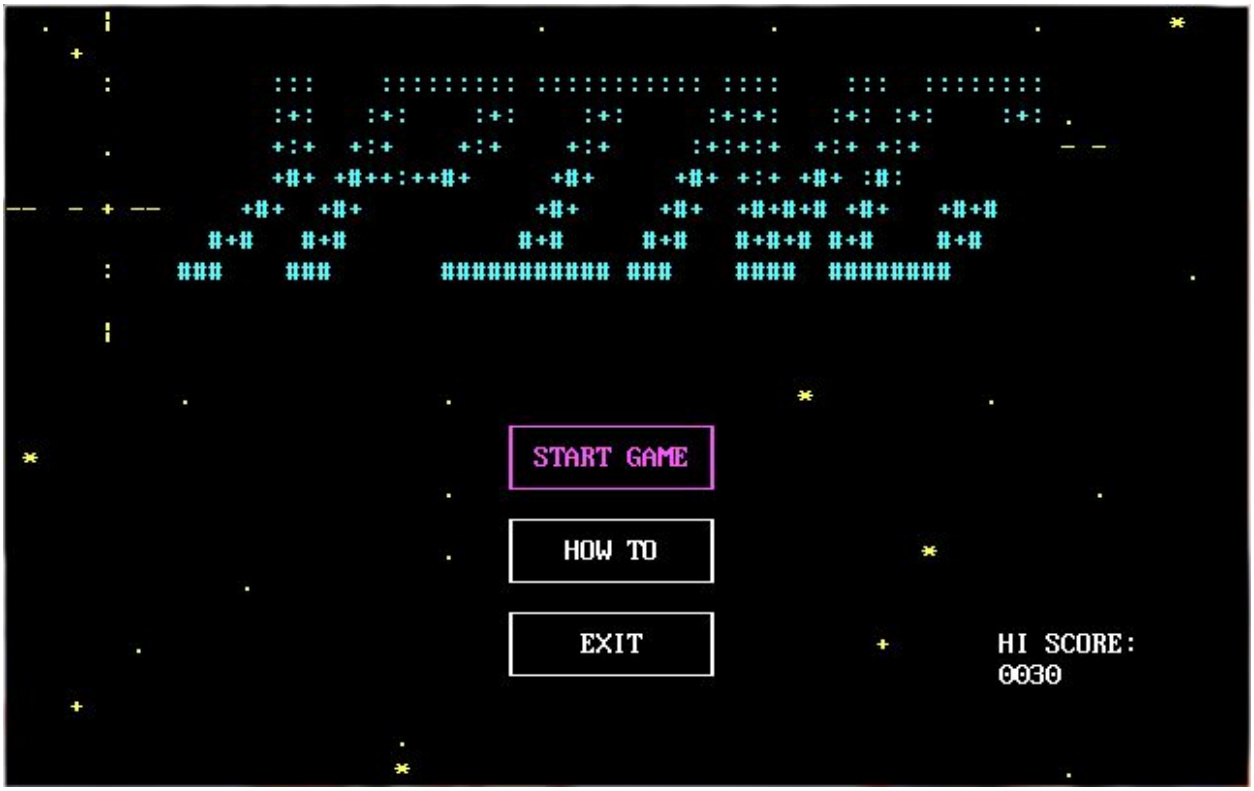
DRAWICS PROC NEAR

- This procedure is called every cycle. This determines which interceptors are to be displayed base from interceptor’s flag called active which determines if it is active. It then invoke the drawIC procedure of the instance of that interceptor if it was active.

UPDATEICS PROC NEAR

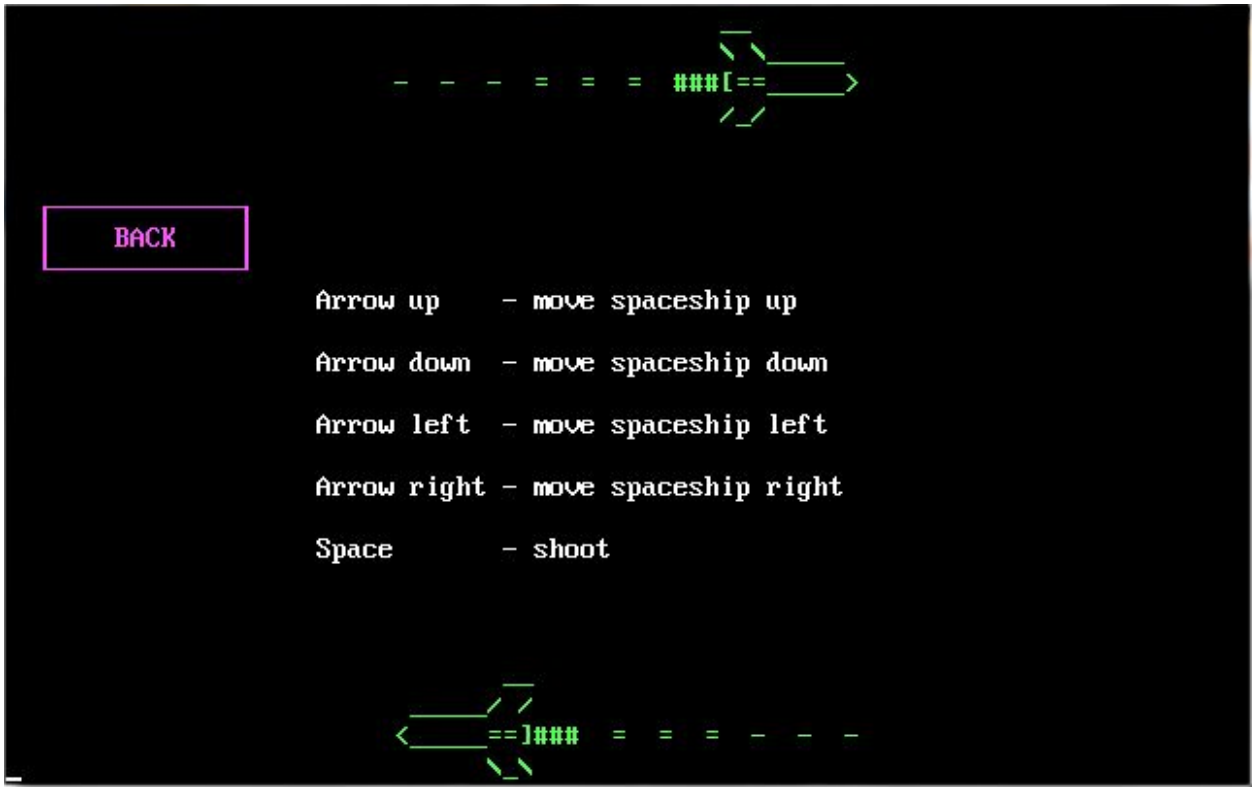
- This procedure is also called every cycle. This determines which interceptors are to be updated(update only if instance of interceptor is active). It then invokes the updateIC procedure of the instance of that interceptor if it was fired.

Screenshots of Game



Game Title Screen

This is our title screen and here we can see our game's title and the different menu items you can choose from, namely: Start Game, How To and Exit. You can navigate through them by pressing the arrow up, arrow down and enter keys. Choosing 'Start Game' lets you begin playing the game wherein you shoot incoming enemies to increase your score. The main goal of the game is to get as far and kill as many enemies as you can in order to get the highest score among other players. Choosing next item, 'How To', let's you display the different controls used in the game. Lastly, choosing 'Exit' lets you quit the game.

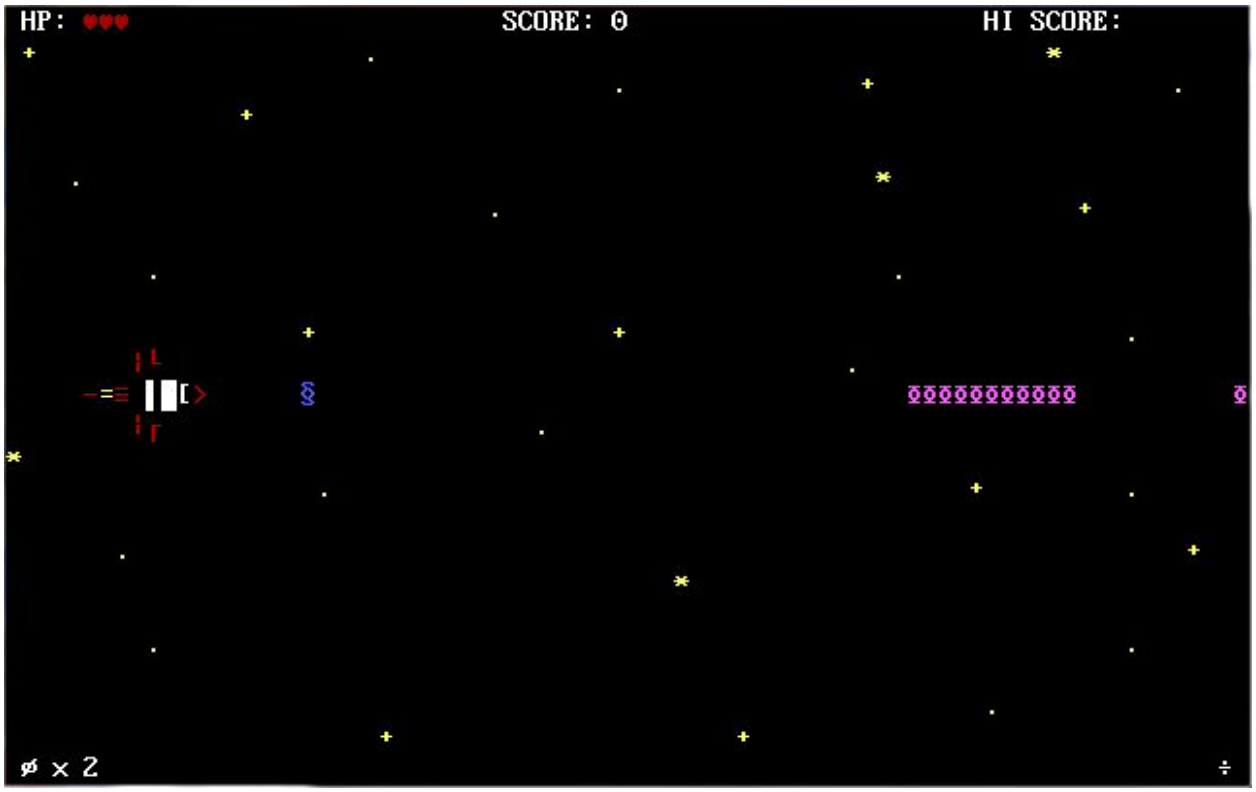


How to Play Screen

This is our game's How to Play Screen. It shows the different keys needed to control your ship ingame. As shown in the picture, the four different arrow keys lets you navigate the ship in any direction. Whenever there's an incoming enemy you need to shoot, press Space to fire. There's also a back button to go back to the Title Screen and choose the other menu items.

Game Over Screen

This Game Over Screen shows up when you have no more lives left to continue playing. It displays your last score you achieved before you lost all your lives. Under it is a back button to go back to the Title Screen to play again or choose other menu items.



Actual Game Screen

This is the screen that appears when you choose ‘Start Game’ from the title screen. On the upper left, you can the see the number of lives you have which is represented by the hearts. Once there are no more hearts left, you can’t continue playing anymore and will be directed to the Game Over Screen. The upper center shows you current score ingame while on the upper right shows the current highest score someone achieved while playing the game. On the center part on the left side, you can see the ship that you’ll be controlling. When you press Space, you’ll be able to fire bullets which is represented by the § symbol in blue. The enemies in series are shown by the ¤ symbol which are colored pink/magenta. Other enemies will be possibly represented using different characters and colors.