CMSC 131 FINAL PROJECT

RUN BB

DEVELOPED BY:
Liu,May-En
Cristina Dayuday

Table of Contents

Project Summary

The game's concept is similar to that of many 90's arcade games where repetition and fast reaction is the key to reaching the highest possible score. As the player's score increases per block left behind, the player also decreases his chances of winning by leaving himself a smaller area to work with each time.

Similar to the Snake series, the game rules don't allow the player to walk over his/her own blocks. The player *can* if he/she wishes to exit the game quickly and start over, if not uninstall the game entirely due to anger management issues, but that is not recommended.

To add to the game's excitement, the developers included a 'free roaming object'. Its path and behavior designed to make the game both more challenging and fun. The roaming object can only move diagonally. At an angle, it has the ability to bounce off a block, loop around the block, or ignore it completely. This adds to the unpredictability of the roaming object's movements as the player makes his/her rounds across the map.

However, not all things were created to amuse the developers alone. So to make the game fair, the developers also made the roaming object able to consume each block it bounces off of. Giving the player more room to work with and even a pixel sized escape route if done right. All that said, meeting the free roaming object head on is not recommended as it will also cause the player to exit the game, and perhaps to some extent uninstall as well.

In addition to the game's next generation features, one can also pause the game by having the player touch any of the borders. This causes the whole game, enemy and all, to freeze as the player contemplates on his next move(enemy disappears during the pause and reapppears once player is ready again). The developers consider this border as a 'safe spot' where the player can pass, but the enemy can't. The developers recommend that players will have to think carefully before blocking it off, and only pass along that area as a last resort.

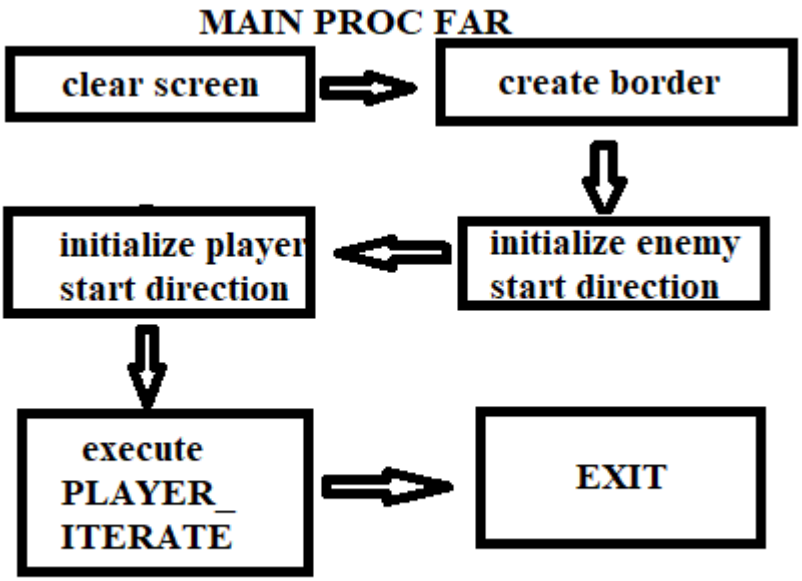
The game uses the DOSBOX's masm as its platform and was made from scratch by emulating the 80386 processor to compile and run the game.

Github Link: https://github.com/bittertongue96/CMSC131FINALPROJECT

List of Procedures

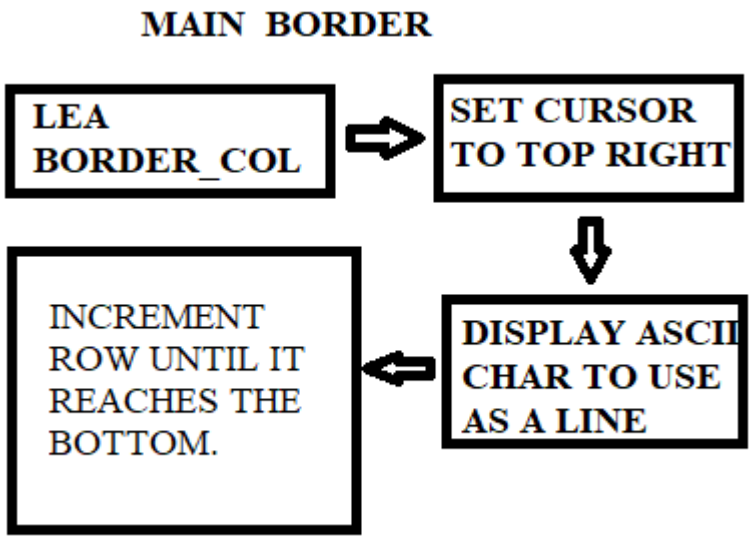
MAIN clears the screen, initializes location for the border, then prompts a procedure to make the border, initializes enemy starting direction, initializes player starting direction, then proceeds to execute the Player_Iterate proc to start the game. If gameover conditions have been met, control returns to main label ENDING.

```
47  MAIN PROC FAR
48    MOV AX, @data
49    MOV DS, AX
50
51   ;clear the screen
52    CALL CLEAR_SCREEN
53   ;BORDER
54       MOV     DL, 9H
55       MOV     DH, 3H
56       CALL GENERAL_SET_CURSOR
57       CALL MAIN_BORDER
58
59    NEXT:
60    ;ENEMY
61    MOV ENEMY_MAX,01 ; set max to 1, meaning going up towards the right.
62    MOV DL, 0CH ;
63    MOV DH, 12
64    CALL ENEMY_SET_CURSOR
65
66    MOV ENEMY_PARTY_COL, 0AH  ;coordinates of where enemy starts out
67    MOV ENEMY_PARTY_ROW, 12
68
69   ;PLAYER
70   MOV DL, 0CH ; MIDDLE LEFTMOST OF THE SCREEN
71   MOV DH, 24H
72   CALL PLAYER_SET_CURSOR
73
74   MOV PLAYER_DIS,0H; initialize as zero to signify starting from col
75   MOV PLAYER_MAX,1H ; initialize to 1 to signify starting movement towards the right
76
77   MOV PLAYER_PARTY_COL, 24H  ; starting coordinates of player to be ~middle of the screen
78   MOV PLAYER_PARTY_ROW, 12
79   JMP PLAYER_ITERATE ; Activate player loop
80      ;PLAYER
81
82
83   ENDING:
84   MOV AH, 4CH
85   INT 21H
86
87  MAIN ENDP
```



MAIN_BORDER displays characters in BORDER that stores duplicated columns(top and bottom), initializes location to start building the rows for the border. Data is duplicated for left and right before returning control to main proc.

```
90   MAIN_BORDER PROC NEAR
91     MOV AH, 09H
92      LEA DX, BORDER
93      INT 21H
94
95      MOV     BORDER_COL, 46H
96      MOV     BORDER_ROW, 4
97          ITERATE:
98              ;set cursor
99              MOV     DL, BORDER_COL
100             MOV     DH, BORDER_ROW
101             CALL    GENERAL_SET_CURSOR
102
103             ;display char from register
104             MOV     AL, 0BAH
105             MOV     AH, 02H
106             MOV     DL, AL
107             INT     21H
108
109             INC     BORDER_ROW
110             CMP     BORDER_ROW, 21
111             JE      NEXT
112             JMP ITERATE
113
114  RET
115  MAIN_BORDER ENDP
```



ENEMY_ITERATE procedure that deals mainly enemy mechanics/movement. Sets cursor to where the enemy will be, calls read cursor, displays the enemy character, gives the direction of the enemy using ENEMY_MAX value(1-4) to see where to go.

```
120  ENEMY_ITERATE PROC NEAR
121
122    MOV DL, ENEMY_PARTY_COL
123    MOV DH, ENEMY_PARTY_ROW ; set coordinates in spot when arrow  was pressed.
124    CALL ENEMY_SET_CURSOR
125    CALL ENEMY_READ_CURSOR
126
127    ;DISPLAY SMILEY CHAR
128    MOV     AL, 02H
129    MOV     AH, 02H
130    MOV     DL, AL
131    INT     21H
132    CALL DELAY
133
134
135   CMP ENEMY_MAX,01
136   JE UPRIGHT
137
138   CMP ENEMY_MAX,02
139   JE DOWNRIGHT
140
141   CMP ENEMY_MAX,03
142   JE DOWNLEFT
143
144   CMP ENEMY_MAX,04
145   JE UPLEFT
```

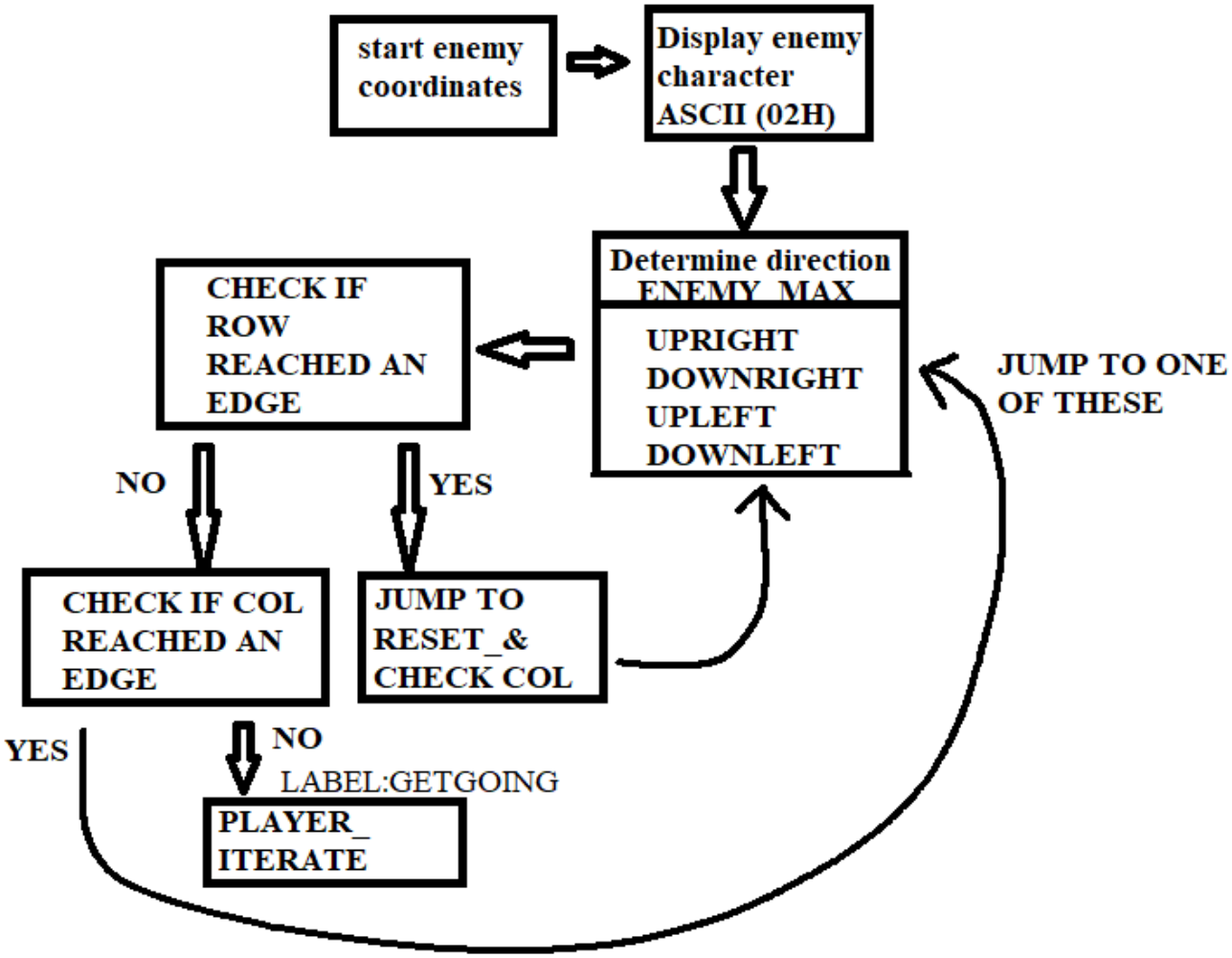
Depending on ENEMY_MAX value, the enemy may go upright(1), downright(2), upleft(4),or downleft(3) and the value is moved to ENEMY_MAX to keep it going the same direction,and ENEMY_ROW and ENEMY_COL are incremented and decremented accordingly

```
148     UPRIGHT: ; GOING UP TO RIGHT
149     CALL ENEMY_DELETE_PREV
150     MOV ENEMY_MAX,01
151     DEC ENEMY_PARTY_ROW
152     INC ENEMY_PARTY_COL
153         CMP ENEMY_PARTY_ROW,04H ;REACHED THE TOP
154         JE RESET_UR
155         CMP ENEMY_PARTY_COL,45H   ;REACHED THE RIGHT.
156         JE UPLEFT
157         JMP GETGOING ; WHOLE LOOP
158
159     DOWNRIGHT:
160     CALL ENEMY_DELETE_PREV
161     MOV ENEMY_MAX,02
162     INC ENEMY_PARTY_ROW
163     INC ENEMY_PARTY_COL
164         CMP ENEMY_PARTY_ROW,14H ;REACHED THE BOTTOM
165         JE RESET_DR
166         CMP ENEMY_PARTY_COL,45H
167         JE DOWNLEFT
168         JMP GETGOING
169
170     DOWNLEFT:
171     CALL ENEMY_DELETE_PREV
172     MOV ENEMY_MAX,03
173     INC ENEMY_PARTY_ROW
174     DEC ENEMY_PARTY_COL
175         CMP ENEMY_PARTY_ROW,14H
176         JE RESET_DL
177         CMP ENEMY_PARTY_COL,0AH; REACHED THE LEFT
178         JE DOWNRIGHT
179         JMP GETGOING
181     UPLEFT:
182     CALL ENEMY_DELETE_PREV
183     MOV ENEMY_MAX,04
184     DEC ENEMY_PARTY_ROW
185     DEC ENEMY_PARTY_COL
186         CMP ENEMY_PARTY_ROW,04H; REACHED THE TOP
187         JE RESET_UL
188         CMP ENEMY_PARTY_COL,0AH; REACHED THE LEFT
189         JE UPRIGHT
190         JMP GETGOING
```

For control to be passed here, the maximum/minimum value of ENEMY_ROW has to be reached. If it has, the ENEMY_RESET then checks if ENEMY_COL has reached maximum/minimum value as well before giving a new direction for the enemy to go.

```
    RESET_DL: ;  RIGHT TO LEFT MOTION

     CMP ENEMY_PARTY_COL,0AH
     JE UPRIGHT
     JMP UPLEFT

    RESET_DR:;  RIGHT TO LEFT MOTION

     CMP ENEMY_PARTY_COL,45H
     JE UPLEFT
     JMP UPRIGHT

    RESET_UR:

     CMP ENEMY_PARTY_COL,45H ;REACHED THE UPRIGHT
     JE DOWNLEFT
     JMP DOWNRIGHT

    RESET_UL: ; REACHED UP LEFT

     CMP ENEMY_PARTY_COL,0AH
     JE DOWNRIGHT
     JMP DOWNLEFT
   RET

ENEMY_ITERATE ENDP
```

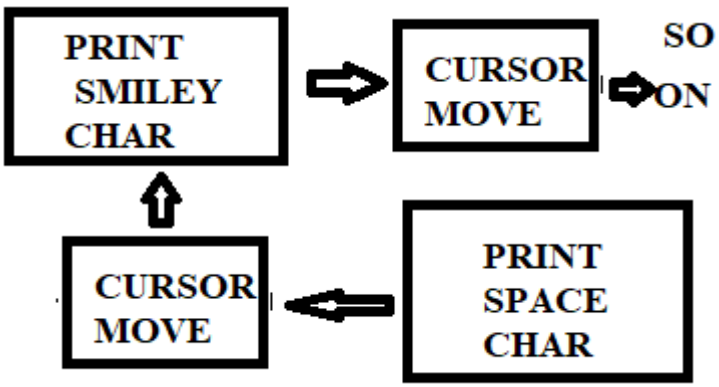
If it fails to reach any of the highest or lest side of the screen, it then checks to see if it reached the farthest end or beginning of the screen. Then it jumps to another direction inside ENEMY_ITERATE.



Otherwise, it loops back to the PLAYER_ITERATE to keep the game moving as is.

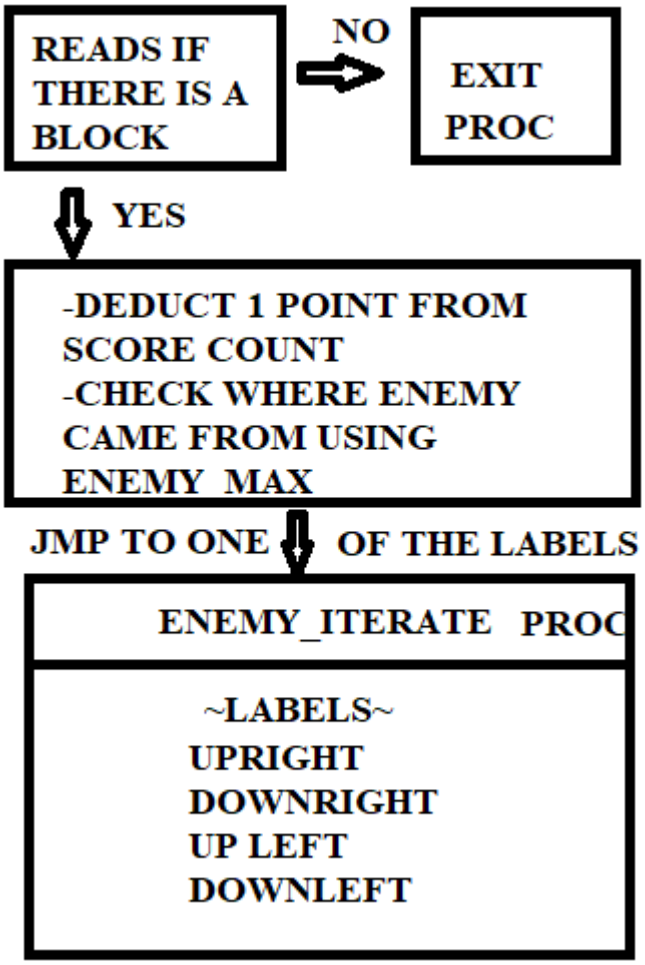
ENEMY_DELETE_PREV this ‘deletes’ the trails left behind by the setting the cursor to where the enemy was and overwriting with ascii spaces(20H).

```
506  DELETE_PREV PROC NEAR
507
508    MOV AH, 02H
509    MOV BH, 00
510    MOV DH, PLAYER_PARTY_ROW
511    MOV DL, PLAYER_PARTY_COL
512    INT 10H
513
514
515      MOV      AL, 16H
516      MOV      AH, 02H
517      MOV      DL, AL
518       INT     21H
519
520    RET
521
522  DELETE_PREV ENDP
```



ENEMY_READ_CURSOR the proc reads information that the cursor passes through. It then determines whether or not the enemy has passed an ascii block (16H) . If it doesn't detect that it has passed the ascii block, it carries on normally. If it detects the ascii block, a score from the player is deducted and it now checks the previous direction of the enemy by using ENEMY_MAX before deciding on the new direction to give.

```
257  ENEMY_READ_CURSOR PROC NEAR
258      MOV AH, 08H
259      MOV BH, 00
260      INT 10H
261
262      CMP AL, 16H
263      JE SORTER
264      JMP NEVERMIND
265
266       SORTER:
267       DEC SCORECOUNT
268       CMP ENEMY_MAX,01 ;  UPRIGHT
269       JE DOWNRIGHT
270       CMP ENEMY_MAX,02 ;  DOWNRIGHT
271       JE DOWNLEFT
272       CMP ENEMY_MAX,03;DOWNLEFT
273       JE UPRIGHT
274       CMP ENEMY_MAX,04 ;UPLEFT
275       JE UPRIGHT
276
277
278     NEVERMIND:
279       RET
280  ENEMY_READ_CURSOR ENDP
```


READS IF THERE IS A BLOCK
NO
EXIT PROC
YES
-DEDUCT 1 POINT FROM SCORE COUNT
-CHECK WHERE ENEMY CAME FROM USING ENEMY_MAX
JMP TO ONE OF THE LABELS
ENEMY_ITERATE PROC
~LABELS~
UPRIGHT
DOWNRIGHT
UP LEFT
DOWNLEFT

PLAYER_ITERATE this procedure deals mainly with the player mechanics. This procedure is also the main loop of the entire game. It first displays the score of player as soon as the game start. Then it checks whether or not a player has inputted a value key(arrows and escape),

```
285   PLAYER_ITERATE PROC NEAR
286  ;DISPLAY SCORE SA SCREEN
287     MOV DL, 10
288     MOV DH, 22
289     CALL GENERAL_SET_CURSOR
290     MOV AH, 09
291     LEA DX, SCOREMSG
292     INT 21H
293 ;DISPLAY SCORECOUNT
294     MOV DL, 20
295     MOV DH, 22
296     CALL GENERAL_SET_CURSOR
297     MOV AL, SCORECOUNT
298     DEC AL
299     XOR AH,AH
300     CALL DISPNUM
301
302     CALL GET_KEY
303
304     CMP PLAYER_INPUT,1BH ;CHECK IF INPUT MATCHES ESCAPE KEY. IF NOT, CONITNUE MOVING. IF IT DOES, JUMP TO TERMINATE POC TO EXIT
305     JE ENDING
306
307     CMP AX,4800H; UP
308     JE UP
309
310     CMP AX,5000H; DOWN
311     JE DOWN
312
313     CMP AX,4B00H;LEFT
314     JE LEFT
315
316     CMP AX,4D00H; RIGHT
317     JE RIGHT
```


if not it continues to loop on itself. Displaying ascii block (16H) per loop.Using the initialized/previous value of PLAYER_DIS(1&0 for row or col) and PLAYER_MAX (1&0 for left or right) and PLAYER_MAX2 (1&0 for up or down) for direction until a key is inputted or until it reaches the border and stops the game to wait for a key input.

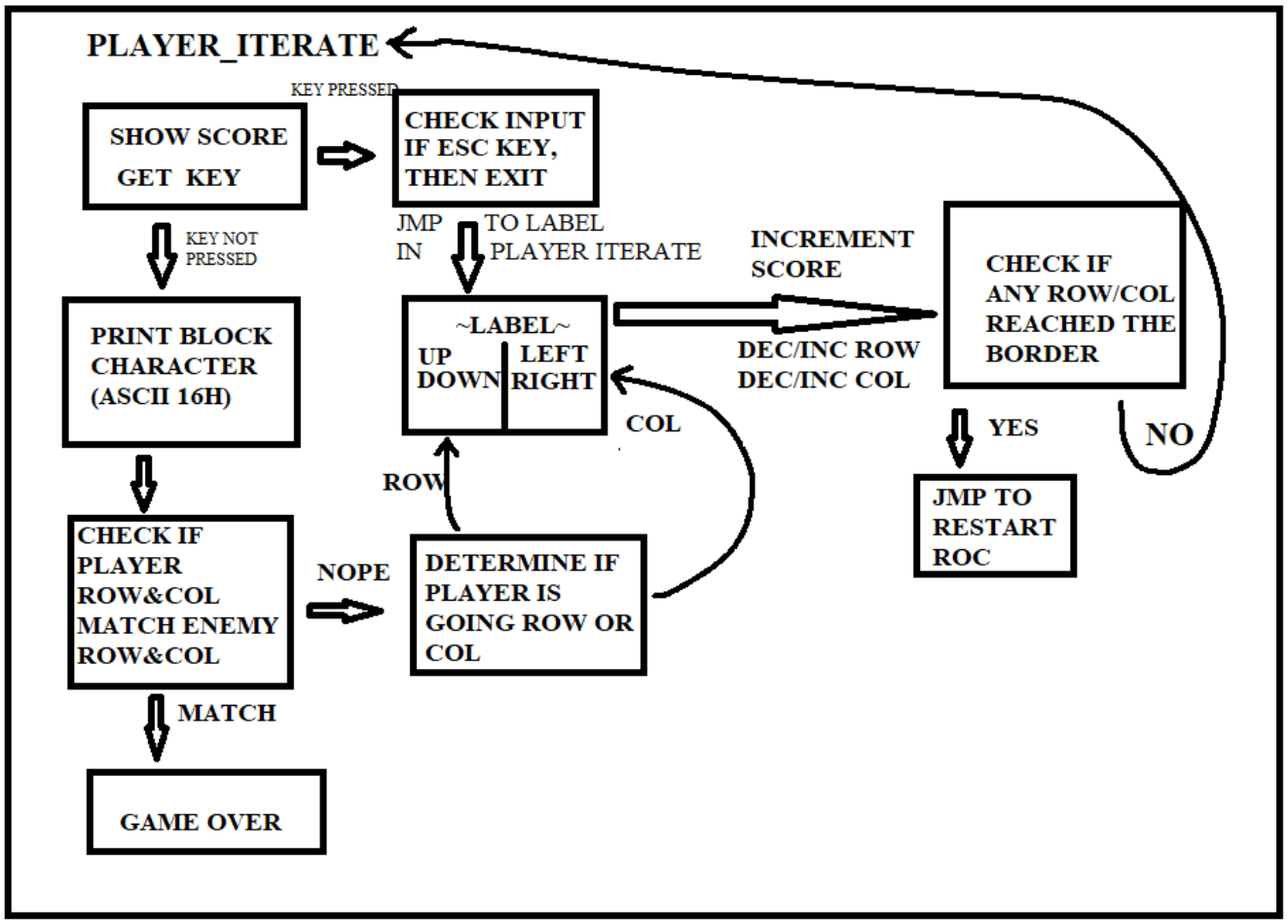
It also checks whether or not the player has met the enemy. Comparing the columns of enemy and player before comparing the rows too. If both conditions are satisfied, it means that the player and enemy are on the same spot. This ends the game. Other wise it carries on.

```
325     MOV DL, PLAYER_PARTY_COL
326     MOV DH, PLAYER_PARTY_ROW ; set coordinates in spot when arrow  was pressed.
327     CALL PLAYER_SET_CURSOR
328     CALL PLAYER_READ_CURSOR
329
330     ;DISPLAY BLOCK CHAR
331     MOV     AL, 16H
332     MOV     AH, 02H
333     MOV     DL, AL
334     INT     21H
335      ;CALL DELAY
336
337     MOV AL, ENEMY_PARTY_COL
338     CMP PLAYER_PARTY_COL, AL
339     JE NEXT_CMP1
340     JNE DIR
341
342     NEXT_CMP1:
343     MOV AL, ENEMY_PARTY_ROW
344     CMP PLAYER_PARTY_ROW, AL
345     JE ENDING
346     JNE DIR
347
348
349     DIR: ;direction to go either row or column
350     CMP PLAYER_DIS,0H
351     JE COL
352     JNE ROW
```

The player only needs to push a key once to run in one of the four directions. When the maximum/minimum row/column has been reached, the control then jumps to the restart proc. If not, it will continue to iterate PLAYER_ITERATE to keep the player moving.

```
363     ROW:
364     CMP PLAYER_MAX2,0H
365     JE UP
366
367     CMP PLAYER_MAX2,1H
368     JE DOWN
369
370
371     RIGHT:
372     CALL DELETE_PREV
373     MOV PLAYER_DIS,0H; TO DISTINGUISH BETWEEN ROL AND COL
374     MOV PLAYER_MAX,1H
375     INC PLAYER_PARTY_COL
376     INC SCORECOUNT
377     CMP PLAYER_PARTY_COL,46H ;RIGHT ARROW
378     JE RESTART
379     JMP PLAYER_ITERATE
380
381     LEFT:
382     CALL DELETE_PREV
383     MOV PLAYER_DIS,0H; TO DISTINGUISH BETWEEN ROL AND COL
384     MOV PLAYER_MAX,0H
385     DEC PLAYER_PARTY_COL
386     INC SCORECOUNT
387     CMP PLAYER_PARTY_COL,0AH
388     JE RESTART
389     JMP PLAYER_ITERATE
390
```

```
391     UP:
392     CALL DELETE_PREV
393     MOV PLAYER_DIS,1H ; TO DISTINGUISH BETWEEN ROL AND COL
394     MOV PLAYER_MAX2,0H;TO DISTINGUISH BETWEEN UP AND DOWN
395     DEC PLAYER_PARTY_ROW
396     INC SCORECOUNT
397     CMP PLAYER_PARTY_ROW,03H
398     JE RESTART
399     JMP PLAYER_ITERATE
400
401     DOWN:
402     CALL DELETE_PREV
403     MOV PLAYER_DIS,1H ; TO DISTINGUISH BETWEEN ROL AND COL
404     MOV PLAYER_MAX2,1H;TO DISTINGUISH BETWEEN UP AND DOWN
405     INC PLAYER_PARTY_ROW
406     INC SCORECOUNT
407     CMP PLAYER_PARTY_ROW,15H
408     JE RESTART
409     JMP PLAYER_ITERATE
410
411
412        RET
413
414 PLAYER_ITERATE ENDP
```

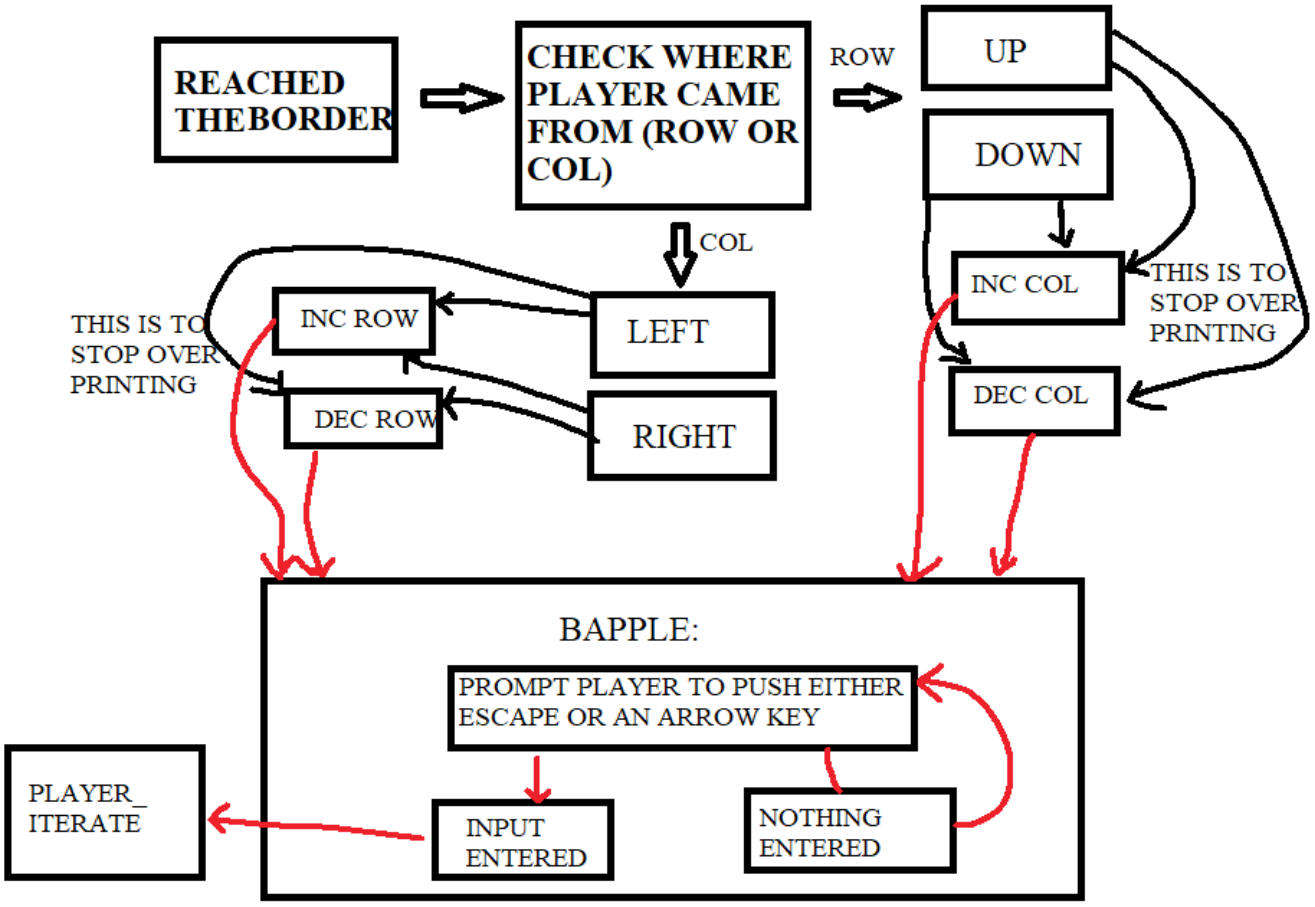

PLAYER_ITERATE
KEY PRESSED
SHOW SCORE
GET KEY
CHECK INPUT
IF ESC KEY,
THEN EXIT
KEY NOT
PRESSED
JMP
IN
TO LABEL
PLAYER ITERATE
INCREMENT
SCORE
CHECK IF
ANY ROW/COL
REACHED THE
BORDER
PRINT BLOCK
CHARACTER
(ASCII 16H)
~LABEL~
UP
DOWN
LEFT
RIGHT
DEC/INC ROW
DEC/INC COL
YES
NO
JMP TO
RESTART
ROC
COL
ROW
CHECK IF
PLAYER
ROW&COL
MATCH ENEMY
ROW&COL
NOPE
DETERMINE IF
PLAYER IS
GOING ROW OR
COL
MATCH
GAME OVER

RESTART this procedures deals on what will happen once the player has reached the border. First it determines where the player was going and the last key pressed. Then it will give the direction on where the player should go next.

```
RESTART  PROC NEAR

        CMP PLAYER_DIS,0H ;DISTINGUISH WHETHER INFO CAME
        JE ROW_ONLY; IF FROM DIS 0--WHICH IS COL, JUMP TO
        JNE COL_ONLY

         ROW_ONLY:
         MOV PLAYER_DIS,1H
         CMP PLAYER_MAX2,1H
         JE DOWNY
         JNE UPPY

            UPPY:
         MOV PLAYER_MAX2,1H    ;GOING UP/DECREMENTING
         CMP PLAYER_PARTY_COL,46H
         JE DECREMENT_COL
         CMP PLAYER_PARTY_COL,0AH
         JE INCREMENT_COL

            DOWNY:
         MOV PLAYER_MAX2,0H    ;GOING DOWN/INCREMENTING
         CMP PLAYER_PARTY_COL,46H
         JE DECREMENT_COL
         CMP PLAYER_PARTY_COL,0AH
         JE INCREMENT_COL

         DECREMENT_COL:
         DEC PLAYER_PARTY_COL
         JMP BAPPLE

         INCREMENT_COL:
         INC PLAYER_PARTY_COL
         JMP BAPPLE

COL_ONLY:
MOV PLAYER_DIS,0H
CMP PLAYER_MAX,1H
JE RIGHTY
JNE LEFTY

RIGHTY:
MOV PLAYER_MAX,1H; GOING RIGHT, INC
CMP PLAYER_PARTY_ROW,15H
JE DECREMENT_ROW

CMP PLAYER_PARTY_ROW,03H
JE INCREMENT_ROW

LEFTY:
MOV PLAYER_MAX,0H ;GOING LEFT, DECR
CMP PLAYER_PARTY_ROW,03H
JE INCREMENT_ROW

CMP PLAYER_PARTY_ROW,15H
JE DECREMENT_ROW

DECREMENT_ROW:
DEC PLAYER_PARTY_ROW
JMP BAPPLE

INCREMENT_ROW:
INC PLAYER_PARTY_ROW
JMP BAPPLE
```

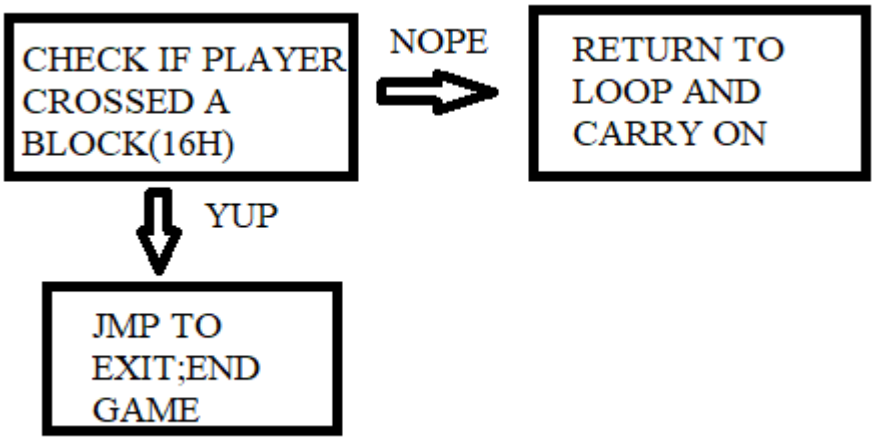
If no key is pressed, it will proceed to the BAPPLE loop which pauses the entire game until an input is entered. Once an input has been entered, it will return control back to PLAYER_ITERATE proc and carries on.

```
        BAPPLE:

         MOV     AH, 01H     ;check for input.
         INT     16H

         JZ      BAPPLE  ;NO INPUT DETECTED, TRY TO DETECT INPUT AGAIN



        JMP PLAYER_ITERATE ; SEE WHAT TO DO NEXT

RET
RESTART ENDP
```

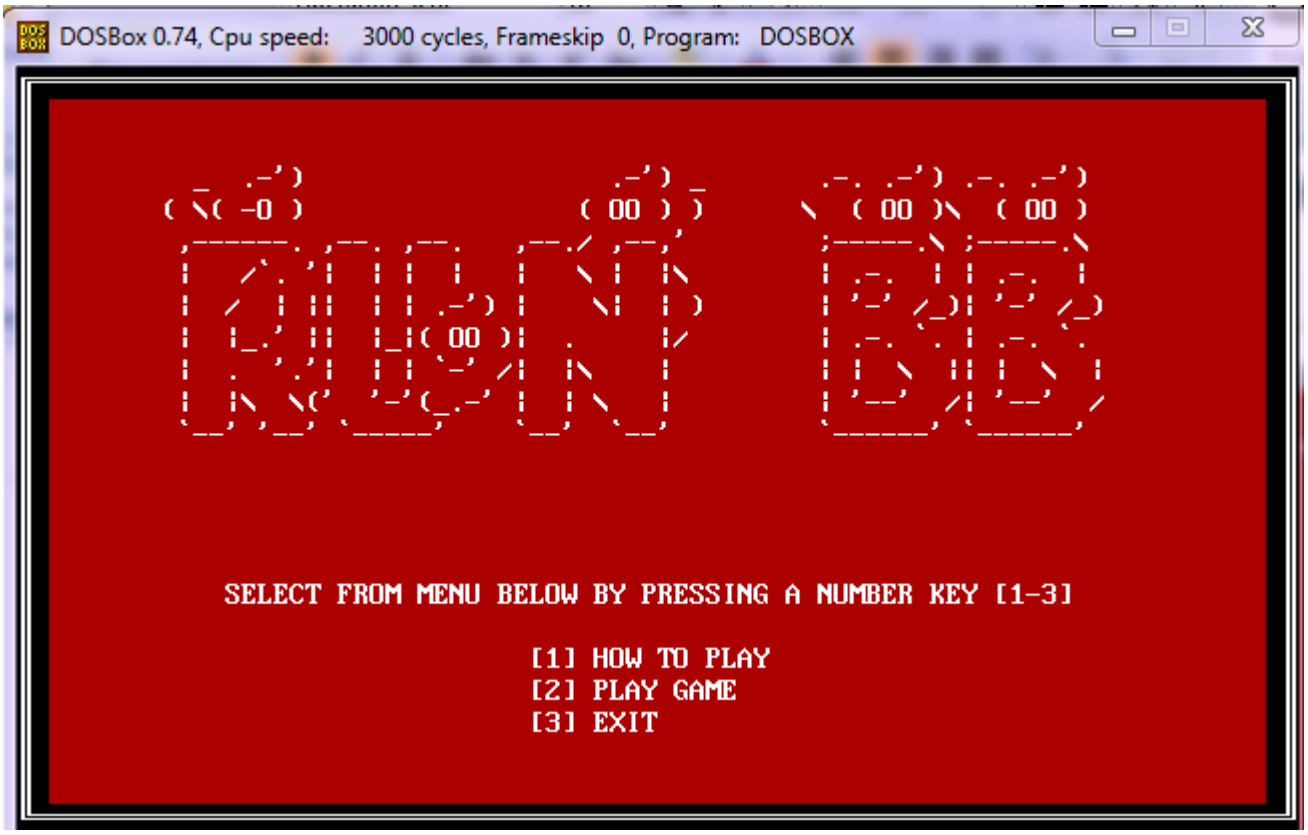

REACHED THEBORDER
CHECK WHERE PLAYER CAME FROM (ROW OR COL)
ROW
UP
DOWN
COL
INC COL
THIS IS TO STOP OVER PRINTING
LEFT
INC ROW
THIS IS TO STOP OVER PRINTING
DEC ROW
RIGHT
DEC COL
BAPPLE:
PROMPT PLAYER TO PUSH EITHER ESCAPE OR AN ARROW KEY
PLAYER_ ITERATE
INPUT ENTERED
NOTHING ENTERED

PLAYER_READ_CURSOR this procedure reads the information from the set procedure. It checks whether or not the player has ran into its own blocks. If not, it continues on to the PLAYER_ITERATE other wise exits the game.

```
;---------------------------------------------------------
PLAYER_READ_CURSOR PROC NEAR
    MOV AH, 08H
    MOV BH, 00
    INT 10H

    CMP AL, 16H
    JE DO
    JNE THISONE
    DO:
   MOV DL, PLAYER_PARTY_COL
   MOV DH, PLAYER_PARTY_ROW ; set coordinates in spot when a


    JMP ENDING

    THISONE:
    RET
PLAYER_READ_CURSOR ENDP
```



SCREENSHOTS



STARTING SCREEN

GAMEPLAY