University Of Dhaka

Department of Computer Science and Engineering

Assembly Lab Project

Submitted to:

Mr. M. Tawfiqul Shobhon

Lecturer

Submitted By:

Salsabil Ahmed Khan (Roll - 11)

Fariha Moomtaheen (Roll – 23)

INDEX

1.Game Overview:

In this game, a brick of random color is generated at the top of the grid . Now, the player uses the arrow keys to move the brick in sideways or downward direction. Whenever a brick is cemented in a position, if three consecutive bricks of same color match horizontally or vertically, then those three bricks are cleared and any bricks above these will fall. Every time three bricks are matched and cleared the score of the player increases. When no more bricks can be generated, the game is over.

This game has been inspired from the traditional Tetris game where blocks of various geometric shapes fall from the top of the grid , which can be rotated 90 degrees and when a horizontal line of 10 blocks of the same color is created, this line disappears.

2.Purpose of the Game:

The purpose of making this game is to improve our efficiency in assembly language. Again by making this game we get to know graphical work in this language. And we use various interrupt function. Thus we get a vivid idea of the effective and proper use of assembly language

3. Main Logic:

To execute the logical part of this game we have used a 2d array. We use numbers to represent each color and zero to represent a blank block. Every time a new number is generated at the specified position of the array and the arrow keys are used to move this number. Whenever a number is cemented we check if three numbers match consecutively. Here, there can be four cases. The cemented number may be leftmost, rightmost, middle of three horizontal numbers or topmost of three vertical numbers. In each case we clear the three numbers by replacing them with zeroes. Then if there are any numbers above any of these three numbers, we move them down. And keep checking for more matches. Finally when there are no more blocks that are matched, we generate a new random number and the process repeats till game over.

This array is then drawn in VGA graphics 320x200 resolution mode.

4. Game Designing & Planning:

Now let us see the implementation of the main logic. As said before we use a 2d array BLOCK[][] to implement the logical part. We use the registers BX and SI to index the row and column respectively of this array. There are several parts of the implementation. We will give a basic idea of the implementation of various parts and at last show a flowchart of the function calls to give a clear view of the algorithm.

4.1 New Block Generation:

We have generated a new number every time at the top row and middle column. To do this we have kept a color array with 50 randomly generated numbers between 1 to 6 and an iterator IT is used to iterate through this array consecutively and everytime the number that is indexed by this iterator is moved to a register and then moved to the specified position of the grid.

We show the logical part of the function NEW_BLOCK_GENERATION.

```
MOV SI, 4
XOR BX, BX
CMP BLOCK[BX][SI],0
JNE LONGBOTTOM
LEA DI, COLOR
ADD DI, IT
MOV AX, [DI]
MOV BLOCK[BX][SI],AX
```

NERATION

4.2: Movement of block:

We use the arrow keys to move the newly generated block in left, right or down position. We take input using the int 16h interrupt. If the player presses escape key then we take them back to the main menu about which we will discuss later. Now, the movement of block may be any three of the following:

Down:

If the player wants to move the block down we at first check if the block below it is zero. If it is, that means we can move it down. Then we move to the next row by adding 10 to BX (as number of column is 5 and each array block is a word, so 5x2=10). The block cannot be moved down if there is already a colored block underneath it or if it has reached the last row. At the same time we clear the previous position of this block by

```
MOV_DOWN:
SUB BX,10
MOV BLOCK[BX][SI],0
MOV TEMP_COL,DX
INC CHANGE_ROW
ADD BX,10
MOV DX,TEMP_COL
MOV BLOCK[BX][SI],DX
```

Left:

In the same way, if a player wants to move left we first check if the block before it is zero. If it is, then we can move left. To do so, we subtract 2 from SI and move the block to BLOCK[BX][SI] clearing the previous block. We cannot move left if the block is in the first column of the grid.

```
ADD SI,2
 MOV BLOCK[BX][SI],0
 SUB SI,2
 MOV BLOCK[BX][SI],DX
```

The right movement is almost the same as left only we add 2 to SI to move to the right block.

```
SUB SI,2
 MOV BLOCK[BX][SI],0
 ADD SI,2
 MOV BLOCK[BX][SI],DX
```

4.3 Check Color Match:

To check if three colors match we consider four cases. These cases are discussed as follows.

Leftmost:

If the block that is cemented is the leftmost of the 3 horizontal blocks then we have to check the two blocks to its right to see if the colors match.

```
LEFTMOST:
    MOV DX,BLOCK[BX][SI]
L1:
    CMP SI,8
    JE RIGHTMOST
    CMP SI,6
    JE RIGHTMOST
    ADD SI,2
    CMP DX,BLOCK[BX][SI]
    JE L2
    SUB SI,2
    MOV DX,BLOCK[BX][SI]
    JMP RIGHTMOST
L2:
    ADD SI,2
    CMP DX,BLOCK[BX][SI]
    JE JMP_LEFT
    SUB SI,2
    SUB SI,2
    JMP RIGHTMOST
```

Rightmost:

If the block that is cemented is the rightmost of the 3 horizontal blocks then we have to check the two blocks to its left to see if the colors match.

```
RIGHTMOST:
    MOV DX,BLOCK[BX][SI]
R1:
    CMP SI,0
    JE TOP
    CMP SI,2
    JE MIDDLE
    SUB SI,2
    CMP DX,BLOCK[BX][SI]
    JE R2
    ADD SI,2
    MOV DX,BLOCK[BX][SI]
    JMP MIDDLE
R2:
    SUB SI,2
    CMP DX,BLOCK[BX][SI]
    JE JMP_RIGHT
    ADD SI,2
    ADD SI,2
    JMP MIDDLE
```

Middle:

If the block that is cemented is the middle of the 3 horizontal blocks then we have to check the two blocks to its left and right each to see if the colors match.

```
MIDDLE:
    CMP SI,0
    JE TOP
    CMP SI,8
    JE TOP
    MOV DX,BLOCK[BX][SI]
LM1:
    SUB SI,2
    CMP DX,BLOCK[BX][SI]
    JE RM1
    ADD SI,2
    MOV DX,BLOCK[BX][SI]
    JMP TOP
RM1:
    ADD SI,2
    ADD SI,2
    CMP DX,BLOCK[BX][SI]
    JE JMP_MIDDLE
    SUB SI,2
    JMP TOP
```

Top:

To check if the cemented block is the top of three vertical blocks we check the lower two blocks underneath this to see if the colors match.

```
TOP:
    CMP BX,40
    JE END_COL_TOP
    CMP BX,30
    JE END_COL_TOP
    MOV DX,BLOCK[BX][SI]
T1:
    ADD BX,10
    CMP DX,BLOCK[BX][SI]
    JE T2
    SUB BX,10
    MOV DX,BLOCK[BX][SI]
    JMP END_COL_MATCH
T2:
    ADD BX,10
    CMP DX,BLOCK[BX][SI]
    JE JMP_TOP
    SUB BX,10
    SUB BX,10
    JMP END_COL_MATCH
```

4.4 Clear Blocks:

Now if three colors match we have to clear those blocks. So we place zero in place of the numbers in those blocks. In case of leftmost match, along with that block we have to clear the two right blocks, in case of rightmost the two left blocks, in case of middle one to the left and one to the right and in case of top, the two blocks below it. We show only the code of clearing the middle.

```
CLEAR_MIDDLE:
    MOV RR, BX
    MOV CC, SI
    MOV BLOCK[BX][SI],0
    SUB SI,2
    MOV BLOCK[BX][SI],0
    ADD SI,2
    ADD SI,2
    MOV BLOCK[BX][SI],0
    ADD SCORE, 1
    CALL SHOW_SCORE
    MOV MF, 1
```

Everytime 3 colors match we have to move any block that may have been above these three blocks. To do so we move down each row by one unit by traversing upward until a row with zero is reached.

Let us talk about a single case where we can do this, for example, the middle case. In this case we at first check if the flag MF is 1, indicating that we cleared three blocks in middle case. Now there is a function move array in which it knows the

row and column of a cleared block and it moves any colored block above this until zero is reached. This is done three times in every case.

```
MIDDLE_CASE:
    CMP MF, 1
    JNE END_UA
    ADD CC, 2
    CALL MOVE_ARRAY
    MOV CX, TMP_RR
    MOV RR, CX
    SUB CC, 4
    CALL MOVE_ARRAY
```

```
MOVE_ARRAY PROC
   PUSH_ALL
   MOV BX, RR
   MOV SI, CC
   SUB BX, 10
LOOP_MA:
   CMP BLOCK[BX][SI], 0
   JE END_MA
   MOV DX, BLOCK[BX][SI]
   MOV BLOCK[BX][SI], 0
   XCHG BX, RR
   XCHG SI, CC
   MOV BLOCK[BX][SI], DX
   XCHG BX, RR
   XCHG SI, CC
   SUB RR, 10
   SUB BX, 10
   JMP LOOP_MA
END_MA:
   POP_ALL
   RET
MOVE_ARRAY ENDP
```

4

T ny other blocks match after the update of the array. This is done by scanning the entire array each time by

Three blocks, at first row wise and the column wise. If any three match we call the update array function again. This continues until no 3 blocks are found that match.

The loop of the check again function (row wise) is as follows:

```
CHK_ROW:
   MOV BX,0
   MOV SI,0
```

```
    JMP LCS2
LOOP_LCS1:
    ADD BX,10
    XOR SI,SI
    JMP LCS1
    CMP CHK_FLAG, 0
    JE CHK_COLUMN
```

4.7 Drawing the array in VGA graphics mode:

Now, the array is updated everytime as the game continues. And each time the array is updated we draw the array in graphics mode with draw array function. In this function we traverse the array, by starting from the block in first row and first column of the array. For each block we call draw block function that draws the block in graphics mode. We at first decide the row and column where this block will be drawn and then increase this by 12 and using a nested loop draw the block. This is done for every block in the array with the help of a loop. We use interrupt int 10h to draw this block.

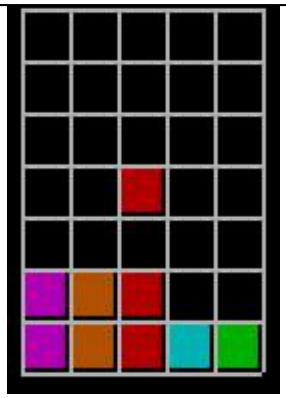
Let us see a portion of this draw block function.

```
MOV AH, 0BH
    MOV BH, 1
    MOV BL, 1
    INT 10h
    MOV AH, 0CH
    MOV AL, TMP
    MOV BH, 0
```

In the above code we have at first selected color, palette and using int 10h implemented to draw the block. Then by moving 0CH to AH we call the function to write pixel in page zero (BH = 0). These pixels are drawn with the help of a loop in the following way.

```
LOOP1_DB:
   CMP DX, D
   JE END1_DB
LOOP2_DB:
   CMP CX, C
   JE L1_DB
   INT 10H
   INC CX
   JMP LOOP2_DB
L1_DB:
   INC DX
   MOV CX, INIT_C
   JMP LOOP1_DB
```

aphics mode at any random moment.

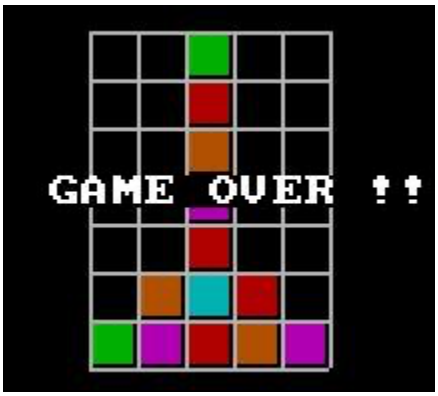


4.8 Score:

Whenever the player matches three colors their score increases by 1. At first this score is added to a word variable. But to print in graphics mode we have to print it as a string. So we get each digit of the score in reverse order by dividing it by 10 and storing the remainder in a temporary array call temp_score. This string is then reversed again and moved to an array called scores to get the score in forward order. Then we print this string using interrupt int 10h and 21h.

4.9 Game Over:

The condition for game over is that if no new block can be generated. This check is done by seeing that whether the block in the top row middle column already has a block. If it does, then no new block can be generated. Thus ending the game. Let us see this situation.



4.10 Menu:

There is a main menu screen where we have provided with two options. The first is new game which whenever chosen a new game will begin and the other is exit which exits from the game completely. The menu screen has been drawn in text mode. This is done by moving 3 to AX and then using the interrupt int 10h.

The menu screen looks like below.

5.Winning condition and Game Over:

In this game there is actually no winning condition. The aim of the player is to score as many points as possible and if no new block can be generated, then the game is over.

6.Limitations:

There are certain limitations to this game:

6.1. We have not used timer to ensure the falling of the blocks. Instead we use the down arrow key.

6.2. We have designed this game in such a way that only three colors will be cleared even if more than three consecutive blocks match.

6.3. The game can be run only inside the dosbox or similar virtual environment.

7.Future Plan:

We plan on overcoming all our limitations of this game. We will try to include timer so as to control the falling of the blocks. And we will also try to change the algorithm in such a way that more than 3 blocks can be cleared if they match.

8. Discussion:

Designing and making this game has given us a deeper knowledge of the Assembly Programming Language. Though due to the low level nature of the language there were a lot of limitations. But it also gave us a clear insight as to how the computer thinks and functions. And building a graphical game with Assembly has obviously given us the basic idea of computer graphics.

9.References:

1. - Ytha Yu & Charles Marut . “Assembly Language Programming and Organization of IBM PC” International Edition 1992