**BRICK BREAKER**

**Team Members:**

**Aitzaz Tahir (19P-0012)**

**Abuzar Zulfiqar (19P-0062)**

**Muhammad Abeer (19P-0061)**

**Section:**

**3-B**

**Overview:**

A simple Brick Breaker game designed in NASM 16-Bit Assembly Language.

**Technology & Software Used:**

* VS-Code
* DOS BOX Emulator

**Components Used:**

* Conditional/Unconditional Jumps
* loops
* Interrupts
* Subroutine & Stack
* Video Memory

**General Working:**

* Separate functions have been created for different parts.
* We have used functions to display the bricks, the borders, the text to be displayed on the screen.
* We have used video memory to display the objects across screen.
* Stack has been used to pass parameters in the functions.
* We have included interrupts to begin the program and to terminate the program when the game is over.
* To Run the code, we have to first compile it and then type the output file.com
* example :

*compile*

* + - * 1. *nasm filename.asm -o outputfile.com*
  1. *run*
     1. *outputfile.com*
* The code is written in such a way that first it clears the screen and displays the menu.
* The program asks for interrupt key to start the game and then when the game is over it asks the user to press the key to terminate the program.

**Detailed Working**

* We have defined variables for row, column, incrementing row and column, clock counter, left edge of the screen, right edge of the screen, calculation of location across the screen, left and right limit of position of movable brick, number of lives and score etc.
* We have also declared 2 variables by the name of brick start and end location in which we have assigned positions to 24 bricks across the screen.
* Code starts and we jump to ***start*** function where at first we have written the code to handle the interrupts. Then a function ***main\_menu*** is called.
* We have made a function ***main\_menu*** which first clears the screen and then it displays the welcome messages which we have stored as string.
* Then we have a function  ***menu\_loop*** which checks if we have entered the correct interrupt keys.
* Then we have a function ***start\_game\_here***  which displays the game setup by calling other functions The function are as follows:
  + firstly it calls **clr\_scr** which clears the screen
  + then there comes print string function which print strings like time, lives and score.
  + there comes the function print which print the symbol of lives
  + then we push the location for score in stack
  + then then we push score value to stack
  + we did previous tow steps because we want to print the score on screen to print them we call **print num** function
  + then there a functions which prints bricks in screen it dose not take any parameters.
  + then there a functions which prints **stacker** in screen it dose not take any parameters.
* A function  ***stacker***  is called which moves the movable brick right and left. It helps the ball bounce up .
* Then we have a function  ***game\_inner\_loop*** which checks if the total bricks have finished or not. If yes then the game ends. Furthermore it checks if the number of lives have finished or not.
* Then we have a function ***endgame*** which checks if we have entered the correct interrupt keys .
* There is a function ***last\_menu\_display***  which displays the score and instructions to end the game or to restart the game after the game gets over.
* There is a function ***do\_restart*** which clears the variables and sets them to restart the game.
* Game is restarted when a key is pressed at the end.
* There is a function ***print\_lives***  which prints the life in the game and when ever the ball hits the bottom, it decrements 1 life.
* **Clr\_scr** function does not take any parameter in this function we are just writing on screen space with black background by using es registers

***Command used in these functions***

**cmp;**

to compare two values

**cli;**

to take command line input

**mov;**

to move value from registers to memory location or vice versa

**div;**

to divide the value

the divided value should be in ax

**push / pop ;**

to use stack / to pass the parameters in function

**ret;**

to return from function

**rep stosw;**

* to print the string
* auto-increment mode
* rep cx times, store words ;
* destination is es:
* di inc/dec
* di as well by 2 bytes

**iret;**

**it returns from interrupts**