UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTA DE INGENIERÍA
CIENCIAS Y SISTEMAS
ARQUITECTURA DE COMPUTADORES Y ENSAMBLADORES 1
DICIEMBRE 2020

Manual de Técnico

Carlos Ojani NG Valladares
201801434

INTRODUCCIÓN

Road Fighter es un juego clásico que consta de un auto que esquiva obstáculos. Este puede recolectar premios a lo largo del nivel, incrementaran su puntaje y al terminar la partida se guarda su resultado. Es un juego muy sencillo basado en únicamente tres niveles, cada uno con diferente dificultad y tiempo.

Como proyecto final, se elaboró este juego en ensamblador, específicamente utilizando MASM para su codificación y DOSBox para su simulación.

ROAD FIGHTER

El juego consta de un menú que permite al usuario registrarse e ingresar a jugar. Para registrarse, en un archivo de datos, se declararon 15 pares de arreglos que constan de un usuario y contraseña, el arreglo de usuario es de tamaño 7 “$” y el de contraseña de 4, que únicamente acepta dígitos.

La contraseña y usuario del administrador se almacenaron de manera similar, pero con datos estáticos. El usuario es “admin” y la contraseña “1234”.

Para registrar usuarios, se debe de verificar que el nombre no exceda de 7 caracteres. Para eso al momento de capturar los caracteres se compara con 7, si es mayor, realiza un salto a una etiqueta que despliega un error, de lo contrario continua a pedir la contraseña. De manera similar, se verifica que no exceda de 4 y se maneja el error de la misma forma.

Cabe mencionar que estos usuarios creados en ejecución son temporales, desaparecen cuando se cierra el juego.

```
userAdmin db 'admin','$'
contraAdmin db '1234','$'
usuario1 db 7 dup('$'),'$'
contra1 db 4 dup('$'),'$'
usuario2 db 7 dup('$'),'$'
contra2 db 4 dup('$'),'$'
usuario3 db 7 dup('$'),'$'
contra3 db 4 dup('$'),'$'
usuario4 db 7 dup('$'),'$'
contra4 db 4 dup('$'),'$'
usuario5 db 7 dup('$'),'$'
contra5 db 4 dup('$'),'$'
usuario6 db 7 dup('$'),'$'
contra6 db 4 dup('$'),'$'
usuario7 db 7 dup('$'),'$'
contra7 db 4 dup('$'),'$'
usuario8 db 7 dup('$'),'$'
contra8 db 4 dup('$'),'$'
usuario9 db 7 dup('$'),'$'
contra9 db 4 dup('$'),'$'
usuario10 db 7 dup('$'),'$'
contra10 db 4 dup('$'),'$'
usuario11 db 7 dup('$'),'$'
contra11 db 4 dup('$'),'$'
usuario12 db 7 dup('$'),'$'
contra12 db 4 dup('$'),'$'
usuario13 db 7 dup('$'),'$'
contra13 db 4 dup('$'),'$'
usuario14 db 7 dup('$'),'$'
contra14 db 4 dup('$'),'$'
usuario15 db 7 dup('$'),'$'
contra15 db 4 dup('$'),'$'
```

Para realizar la parte gráfica del juego se utilizó el modo video o modo 13h, sirve para escribir en memoria y configurar la paleta de colores a mostrar. Básicamente, para desplegar pixeles de distintos colores.

Para ello se parte de declarar un buffer de 64000, ya que esa es la cantidad de pixeles en una pantalla. Las dimensiones de la pantalla son de 320x200. Para configurar modo video se almacena el valor de 13h en AX y se utiliza la interrupción 10h. 13h es el modo gráfico.

Para el manejo de ficheros se utilizó la interrupción 21h, con apoyo de la documentación se logró dicha manipulación. Tanto para leer la ruta de entrada con la función 3FH, pero primeramente ingresando la cadena que representa la ruta, mediante la función 01H que es el ingreso de caracteres con eco. Con la misma interrupción se cierra el programa utilizando la función 4CH.

```
leerCadena macro buffer
    LOCAL ObtenerChar, FinOT ;LOCAL sirve que a la hora de llamar va
    xor si,si   ;limpia contra el mismo, tambien podriamos poner mov
    ObtenerChar:
        leerChar
        cmp al, 0dh ;ascii del salto de linea en hexadecimal
        je FinOT    ;etiqueta que nos dirige fin del macro
        mov buffer[si], al ;mover destino, fuente ======== CON TODO
        inc si  ;si++
        jmp ObtenerChar
    FinOT:
    mov al,24h ;ascii del dolar en hexadecimal = 36d
    mov buffer[si],al ;deja de guardar los valores en el arreglo por
endm

mWrite macro texto
    LOCAL cadena
    .data
        cadena byte texto,'$'
    .code
        push offset cadena
        call WriteString
endm

printReg macro parameter1
        push ax
        mov ax,parameter1
        call toAscii
        imprimir corA
        imprimir Num
        imprimir corC
        pop ax
endm
```

```
leerRuta macro buffer
    LOCAL ObtenerChar, FinOT ;LOCAL sirve que a la hora de llamar varias
    xor si,si   ;limpia contra el mismo, tambien podriamos poner mov si,
    ObtenerChar:
        LeerChar
        cmp al, 0dh ;ascii del salto de linea en hexadecimal
        je FinOT    ;etiqueta que nos dirige fin del macro
        mov buffer[si], al ;mover destino, fuente ======== CON TODO ESTO
        inc si  ;si++
        jmp ObtenerChar
    FinOT:
    mov al, 00h ;ascii del nulo en hexadecimal porque sera el caracter de
    mov buffer[si],al

endm
```

Para pintar los márgenes del juego, se basa de la formula x*320 + y para calcular en donde empezar a pintar. Se utilizó un macro que se encarga de este trabajo, que utiliza di como registro para almacenar el color a utilizar.

```
PintarMargenes macro color
    LOCAL Primera, Segunda, Tercera, Cuarta
    mov dl, color
    mov di,6450
    Primera:
        mov es:[di],dl ;[di] contiene el color de esa posicion de la memoria de graficos
        inc di
        cmp di,6669   ;(20,269) = 320*20+269 = 6669
        jne Primera

    ;barra horizontal inferior
    ;empieza en pixel (i,j) = (190,50) = 320*190+50 = 60850
    mov di,60850
    Segunda:
        mov es:[di],dl
        inc di
        cmp di, 61069
        jne Segunda

    mov di, 6450 ;(20,50)
    Tercera:
        mov es:[di], dl
        add di,320
        cmp di,60850 ;(190,50)
        jne Tercera

    ; Derecha
    mov di,6669  ;(20,269)
    Cuarta:
        mov es:[di], dl
        add di,320
        cmp di,61069 ;(190,269) = 320*190+269= 61069
        jne Cuarta
        mov es:[di],dl
endm
```

Para pintar el carro cada vez que se mueve, se realiza un proceso similar. Está vez se utiliza un proc o procedure llamado pintarCarro. Este proc devuelve almacenados los siguientes valores en estos registros: en di la posición, dl el color, cx la altura y en ax la posición en el instante. A su vez, pinta las llantas y la carrocería del auto. A continuación el fragmento de código utilizado:

```
pintarCarro proc
    PINTARCARROZA:
        mov es:[di], dl
        mov es:[di+1],dl
        mov es:[di+2],dl
        mov es:[di+3],dl
        mov es:[di+4],dl
        mov es:[di+5],dl
        mov es:[di+6],dl
        mov es:[di+7],dl
        mov es:[di+8],dl
        mov es:[di+9],dl
        mov es:[di+10],dl
        mov es:[di+11],dl
        mov es:[di+12],dl
        mov es:[di+13],dl
        mov es:[di+14],dl
        mov es:[di+15],dl
        mov es:[di+16],dl
        mov es:[di+17],dl
        mov es:[di+18],dl
        mov es:[di+19],dl
    SIGFILA:
        dec cx
        cmp cx, 0d
        je LLANTA1
        add di,320
        jmp PINTARCARROZA
    ;PINTAR LLANTAS DEL CARRO
    LLANTA1:    ;SUPERIOR IZQUIERDA
        mov di, ax  ;punto inicial
        add di,1920 ;6 pixeles abajo entonces 6(320) = 1920
        sub di,3    ;le resto 3 para que este sobre la llanta y tenga grosor de 3px
        mov dl, Gris
        ;ALTURA DE LLANTA DE 8PX
        ;PINTAR GROSOR 1
        mov es:[di],dl
        mov es:[di+320],dl
        mov es:[di+640],dl
        mov es:[di+960],dl
        mov es:[di+1280],dl
        mov es:[di+1600],dl
        mov es:[di+1920],dl
        mov es:[di+2240],dl
        ;PINTAR GROSOR 2
        mov es:[di+1],dl
        mov es:[di+321],dl
        mov es:[di+641],dl
        mov es:[di+961],dl
        mov es:[di+1281],dl
        mov es:[di+1601],dl
        mov es:[di+1921],dl
```

Para las demás llantas se realiza el mismo procedimiento, pero variando su posición a pintar.

Para almacenar la configuración del juego, luego de haber leído la ruta de entrada y esta ser válida. Se utiliza un proc llamado Config, que mediante un buffer que almacena los bytes que se leen del archivo, el buffer ya tiene un tamaño lo suficiente grande para leer los tres niveles. Este, mediante “;” para separando los datos y los va capturando para almacenarlos en las variables definidas por nivel, ya que solo son 3 niveles, estás variables son estáticas.

Para regresar al modo texto, se utiliza un macro con el mismo nombre, que utiliza la interrupción 10h y mueve la función 003h a AX.

```
ModoTexto macro
    mov ax, 0003h
    int 10h
endm

ModoVideo macro
    mov ah, 00h
    mov al, 13h
    int 10h
    mov ax, 0A000h
    mov es, ax  ; DS = A000h (memoria de graficos).
endm
```

Para pintar los premios y obstáculos, se utilizó un macro MovimientoLineal, que toma como parámetro las coordenadas a posicionar, guarda en AX la posición de reinicio. A su vez, este macro hace llamado a un proc llamado Lineal que lo que devuelve la coordenada en AX, ya mapeada. Estos macros básicamente realizan el mapeo lexicográfico de la “pantalla”.

```
lineal proc
        push bp                 ;almacenamos el puntero base
        mov  bp,sp              ;ebp contiene la direccion de esp
        push bx
        push dx
        push cx
    ;fin Surutina prologo

    ;Ini Codigo--

        mov bx,[bp+4]           ;bx = col
        mov ax,[bp+6]           ;ax = fila
        mov cx,320
        mul cx
        add ax,bx

    ;Fin Codigo--

    ;ini Subrutina epilogo
        FIN:
            pop cx
            pop dx
            pop bx
            mov sp,bp            ;esp vuelve apuntar al inicio y elimina las variables locales
            pop bp               ;restaura el valor del puntro base listo para el ret
            ret 4
```

Para el botón de pausa se utilizó un proc que controla el uso de este. Se usa una variable declarada en la sección de datos llamada “pausa” que al moverle un 0 decimal se quita la pausa de lo contrario continua.

Al estar en pausa el programa se coloca en modo de escucha, es decir, que espera a que el usuario teclee “esc”, se compara con su valor en hexadecimal, que es 01bh, y si coincide el bucle del juego continua. Ya que toda esta acción se engloba en un procedimiento “Game” y la etiqueta “Bucle” que controla el flujo del juego.

```
Game proc
    BUCLE:
        GameState:
            cmp pausa,1d
            je ControlPausa
            cmp terminoJuego, 1d
            je GAMEOVER
            jmp INSTRUCCION

        Paus:
            call listening
            cmp al, 01bh
            je Despausar
            jmp BUCLE

        QuitPause:
            mov pausa,0d
            jmp BUCLE

        INSTRUCCION:
            mInstrucciones

    jmp BUCLE
```

Al terminar el juego, cuando termina el tiempo o el jugador pierde, se hace un llamado a un procedimiento llamada “GuardarPartida”, que primeramente compara los jugadores que ya están guardados en el juego y empieza a capturar la información de puntos, tiempo y la información del usuario. Finalmente, se reinician todas las variables.