Arquitectura de Computadores y Ensambladores 1
Laboratorio Sección: A
Byron Gerardo Castillo Gómez
201700544

MANUAL TÉCNICO

BALL BREAKER

Utilizando MASM se implementó un juego basado en el popular Ball Braker, este consta de 3 niveles cuya dificultad va incrementando a lo largo del juego, esto se puede apreciar en la velocidad y número de bloques dentro del juego.

BALL BREAKER

Contenido

Requerimientos

DOSBOX

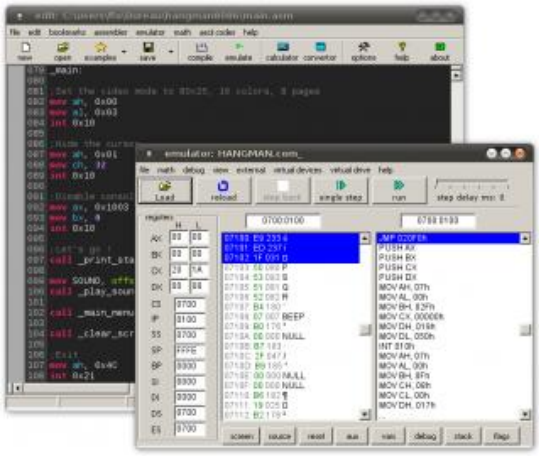
DosBox es un emulador que recrea un entorno familiar al sistema DOS con el objetivo de ejecutar programas y juegos originalmente escritos para MS-DOS. Su instalación requiere simplemente de usar ejecutable, normalmente ya trae todas sus configuraciones por defecto, como la ruta de instalación en el disco C u otras cosas. (el archivo está en enlaces DOSBOX).

MASM611

El macroensamblador de Microsoft (MASM)proporciona varias ventajas para el ensamblado en línea. Su instalación requiere de la carpeta ***masm611*** con todos sus componentes***.***

EMU8086 Ó VS CODE

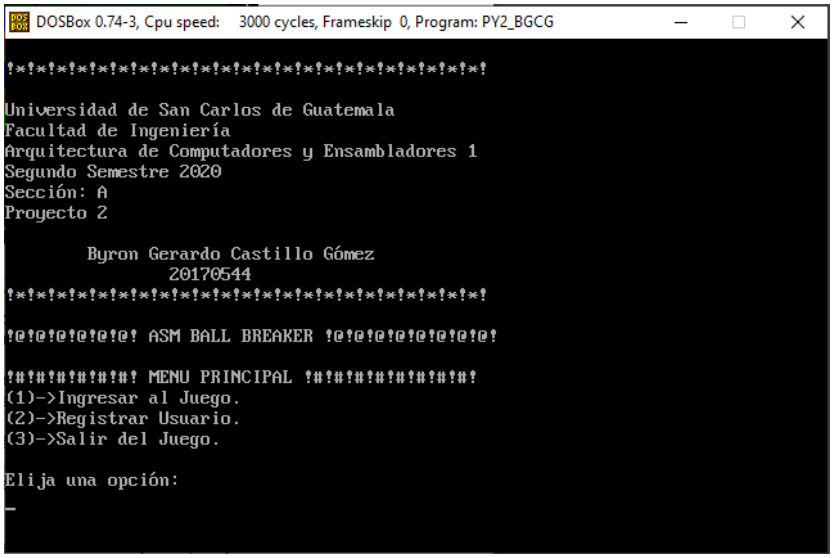
Editores de texto que pueden ser muy útiles para trabajar con ASM, proporcionando reconocimiento de sintaxis, funciones intellisense y detalle de registros entre otras funciones.

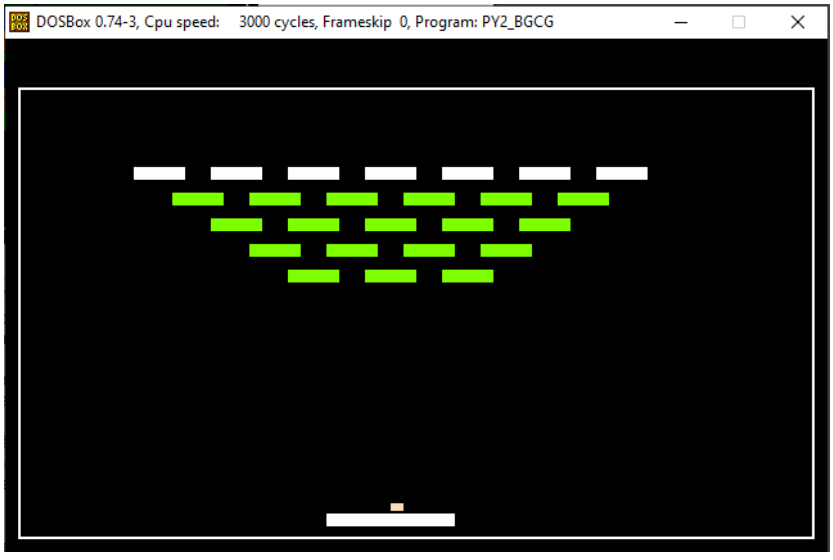


BALL BREAKER

Descripción

Ball Breaker es un juego que permite distintos niveles de dificultad con el fin de entretener al usuario. Los distintos usuarios pueden registrarse, ingresar al juego y en el caso de ser un usuario administrador ver distintos tops que se establecen con las distintas jugadas de los usuarios, generando gráficos de barras que pueden demostrar el tiempo de demora de distintos ordenamientos en base a los tops de los usuarios.

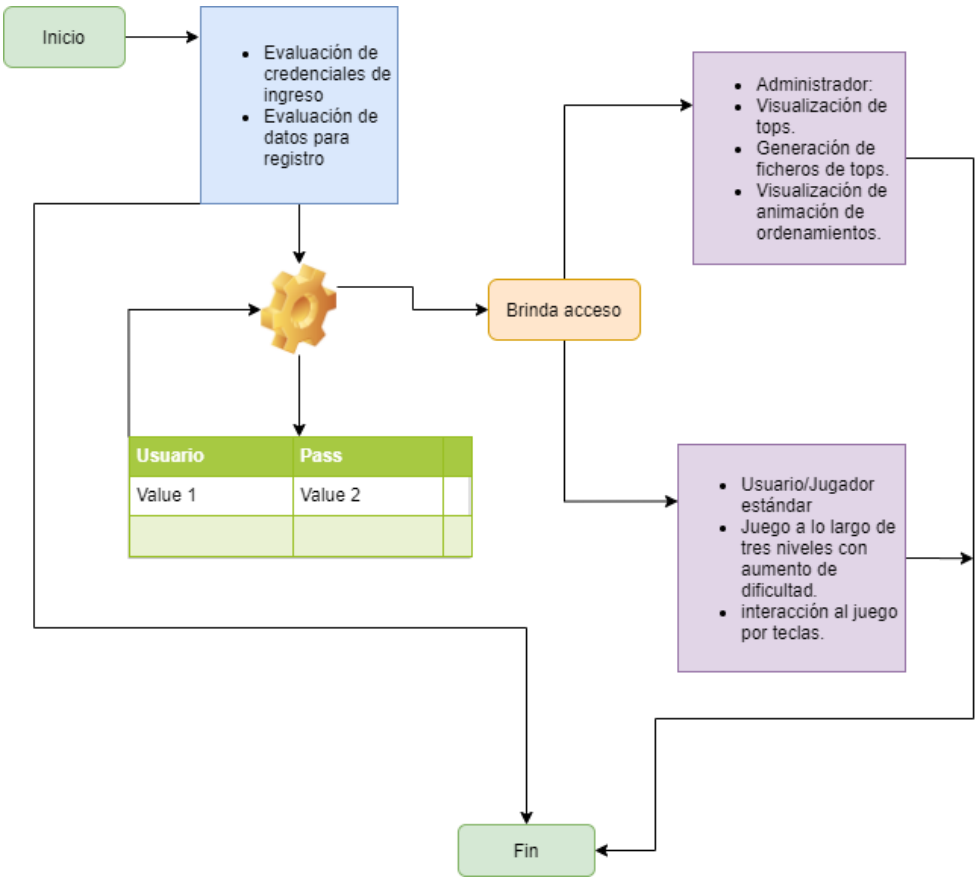




Flujo de la aplicación

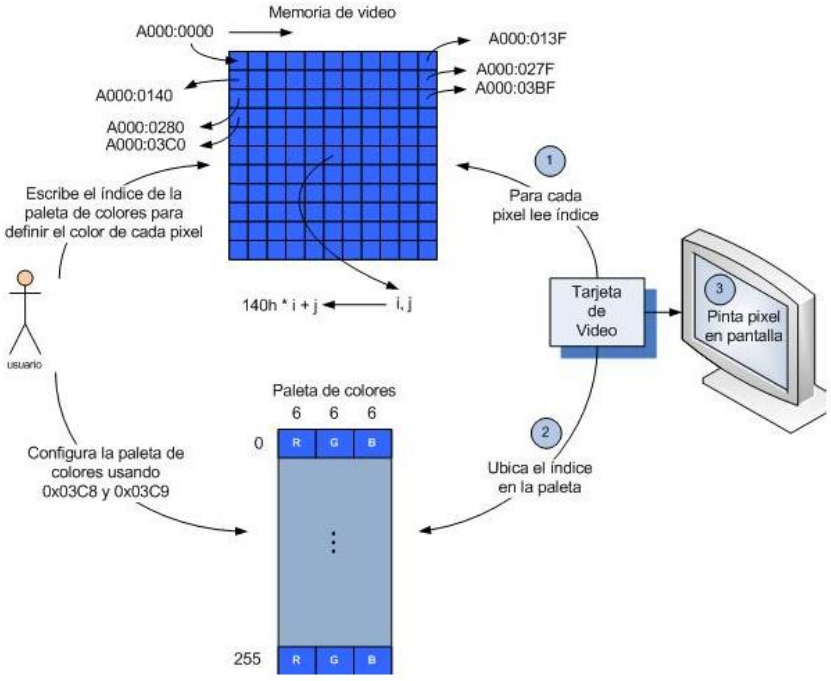
El flujo de la aplicación consiste en la interacción de usuarios administradores como de usuarios estándar o jugadores, según su rol en el software pueden acceder a las distintas funciones del juego.

- El usuario administrador podrá simplemente acceder a los reportes con sus distintas animaciones de ordenamientos y la generación de ficheros.
- El jugador estándar podrá acceder a una partida de juego, tratando así de vencer los 3 niveles del juego con un aumento de la dificultad a lo largo del mismo.
- Ambos roles pueden emplear un registro de un nuevo usuario, siempre y cuando su ***username*** no se repita con el resto caracterizado por ser case sensitive.



BALL BREAKER

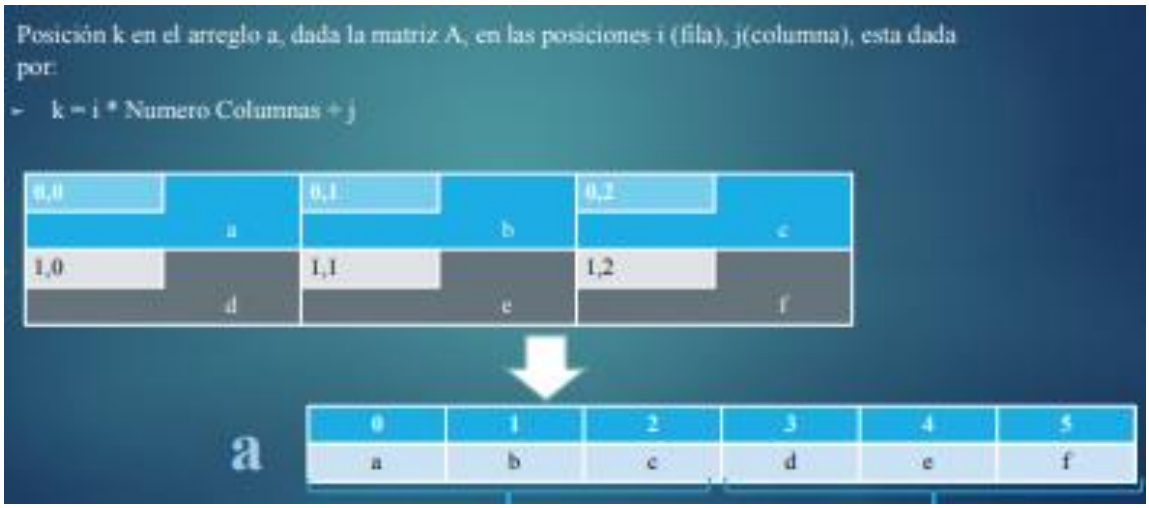
Descripción del Mapeo para el modo Video



Dado que se manejó la memoria de video directamente para una mayor fluidez en los cambios de colores se asignó a DS la posición 0000h para luego poder hacer por una referencia a cada pixel de la matriz. Ya que la matriz se linealiza en memoria entonces se accedió a sus posiciones correctas equivalentes mediante un mapeo por filas:

$$\text{PosArregloMapeado} = (\text{posFila}) * \text{tamañoFila} + \text{posColumna}$$

$$A[199,319] = (199) * 320 + 319 = 63{,}999$$



Funciones Principales

APERTURA, LECTURA Y ESCRITURA DE FICHEROS

Son las principales funciones para el manejo de entrada y escritura de archivos, capaz de guardar los resultados de las distintas partidas, así como los usuarios jugadores con sus respectivas credenciales. La lectura y escritura dependen de la apertura del archivo.

```
reiniciarLectorFicheros macro
    LOCAL borrar
    PUSH SI
    MOV si,0;Bits de SI en 0
    MOV CX, dimensionLectorFicheros
    borrar:
        MOV lectorEntradaFicheros[si], finCadena
        inc si
        loop borrar
    POP SI
endm

abrirArchivo macro rutaArchivo,controlador
    mov ah,subFuncionAbrirFichero
    mov al, permisoLecturaEscritura  ;modo de acceso:
    mov dx, offset rutaArchivo
    int funcionesDOS
    jc errorAperturaArchivo
    mov controlador,ax
endm

leerArchivo macro numBytes,arregloLector,controlador
    mov ah,subFuncionLeerFichero
    mov bx,controlador
    mov cx,numBytes       ;cuantos bytes se leer�n
    lea dx,arregloLector
    int funcionesDOS
    jc errorLecturaArchivo ;Se produce acarreo = 1 en una lectura fallida
endm

cerrarArchivo macro controlador
    MOV ah, subFuncionCerrarFichero
    MOV bx, controlador
    int funcionesDOS
endm

adjuntarContenidoArchivo macro numBytes,arregloEscritor,controlador
    mov bx, controlador
    MOV ah, subFuncionAdjuntarInfoFichero
    MOV cx, numBytes
    MOV dx, offset arregloEscritor
    int funcionesDOS
endm
```

ADMINISTRADOR DE RUTAS

Para el control del juego se manejan distintos archivos, por ende, se necesita un administrador de las rutas que se emplearan para la lectura y escritura en los mismos.

```
;=== VECTORES ===
lectorEntradaTeclado db 20 dup(finCadena)
; llenamos el vector de $ y agregamos un final de cadena
lectorEntradaFicheros db 200 dup(finCadena)
;Rutas ejecutando desde Emu8086:
nombreArchivoJugadores db 'C:\B\Gamers.txt',finRutaFichero
;En dosbox es 'B\Gamers.txt',finRutaFichero
nombreArchivoPartidas db 'C:\B\Rounds.txt',finRutaFichero
;En dosbox es 'B\Rounds.txt',finRutaFichero
nombreReportePuntajes db 'C:\B\Puntos.rep',finRutaFichero
;En dosbox es 'B\Puntos.txt',finRutaFichero
nombreReporteTiempos db 'C:\B\Tiempo.rep',finRutaFichero
;En dosbox es 'B\Tiempo.txt',finRutaFichero
controladorFicheros dw ?
```

```
leerArchivoEnrutado macro indicador
    LOCAL
 asignarRutaJugadores, asignarRutaPartidas, asignarRutaPuntos,asignarRutaTiempos, lectura
    reiniciarLectorFicheros
    cmp indicador,0
    je asignarRutaJugadores
    cmp indicador,1
    je asignarRutaPartidas
    cmp indicador,2
    je asignarRutaPuntos
    cmp indicador,3
    je asignarRutaTiempos
    asignarRutaJugadores:
        abrirArchivo nombreArchivoJugadores,controladorFicheros
        jmp lectura
    asignarRutaPartidas:
        abrirArchivo nombreArchivoPartidas,controladorFicheros
        jmp lectura
    asignarRutaPuntos:
        abrirArchivo nombreReportePuntajes,controladorFicheros
        jmp lectura
    asignarRutaTiempos:
        abrirArchivo nombreReporteTiempos,controladorFicheros
    lectura:
        leerArchivo dimensionLectorFicheros,lectorEntradaFicheros,controladorFicheros
        imprimirEnConsola  lectorEntradaFicheros
endm
```

INTERRUPCIONES Y COLORES

Se manejaron constantes equivalentes para no emplear directamente el hexadecimal y facilitar la lectura de código, asimismo se identificaron sus respectivas sub-funciones utilizadas para el desarrollo del juego. De la misma manera se empleó esta metodología para los colores en el modo de video, para facilitar su implementación.

```
;=== COLORES EQUIVALENTES ===
colorBlancoGrafico EQU 0fh
colorRojoGrafico EQU 28h
colorAmarilloGrafico EQU 0eh
colorVerdeGrafico EQU 2eh
colorCelesteGrafico EQU 4eh
colorPielGrafico EQU 5ah
colorAzulGrafico EQU 21h
colorNegroGrafico EQU 00h

;=== INTERRUPCIONES EQUIVALENTES ===
subFuncionLeerCaracter EQU 01h ;01h -> 1 -> Entrada de caracter con salida
subFuncionVerCadena EQU 09h ;09h -> 9 -> Visualización de una cadena de caracteres
subFuncionCrearFichero EQU 3ch ;3ch -> 60 -> Crear Fichero
subFuncionAbrirFichero EQU 3dh ;3dh -> 61 -> Abrir Fichero
subFuncionCerrarFichero EQU 3eh; 3eh-> 62 -> Cerrar Archivo
subFuncionLeerFichero EQU 3fh ;3fh -> 63 -> Lectura de Fichero o dispositivo
subFuncionAdjuntarInfoFichero EQU 40h ;40h -> 64 -> Escritura(Adjuntada) en Fichero o dispositivo.
subFuncionFinPrograma EQU 4ch ;4ch -> 76 -> Terminación de Programa con Código de Retorno
funcionesDOS EQU 21h ;21h -> 33 -> petición de función al DOS

subFuncionModoVideo EQU 00h ;00h->0->Establecer modo de video
funcionesDespligueVideo EQU 10h ;10h -> 16 -> funciones de modo gráfico o video

subFuncionEstadoTeclado EQU 10h
;10h -> 16 -> verifica el estado del teclado para ver si se ingresaron teclas o no mediante el valor de CF(carry flag)
funcionesTeclado EQU 16h ;16h -> 22 ->  Servicios de Entrada y Salida de teclado
```

ADMINISTRACIÓN DE VIDEO Y GRÁFICOS

Se manejó directamente la memoria de video, por lo que se implementaron procedimientos para cambiar constantemente el valor segmento de datos DS, para alternarnos entre la información de las variables y la referencia a las posiciones de pixel del modo video.

```
establecerSegmentoDatos proc
    asignarDIreccionDatos:
        MOV dx,@data ; Dirección del segmento de datos para poder acceder a las variables
        MOV ds,dx
    ret
establecerSegmentoDatos endp
establecerModoVideo proc
    asignarDIreccionDatos:
        MOV ax, modoVideoGrafico
        int funcionesDespligueVideo
        MOV ax, direccionBaseMemoriaGrafica
        MOV ds,ax
    ret
establecerModoVideo endp

establecerModoTexto proc
    asignarDIreccionDatos:
        MOV ax, 0003h
        int funcionesDespligueVideo
        MOV dx, @data
        MOV ds,dx
    ret
establecerModoTexto endp
```

MENÚS Y EQUIVALENTES

Se manejaron constantes equivalentes para evitar recordar distintas representaciones hexadecimales para la llamada de sub-funciones en interrupciones, así mismo se emplearon para caracteres especiales adjuntadas a las distintas cadenas o arreglos.

```
;=== MENUS DE SELECCI◆N ===
menuPrincipal db saltoLn,retornoCR,
'!#!#!#!#!#!#! MENU PRINCIPAL !#!#!#!#!#!#!#!#!'
,saltoLn,retornoCR,'(1)->Ingresar al Juego.',saltoLn,retornoCR,
'(2)->Registrar Usuario.',saltoLn,retornoCR,
'(3)->Salir del Juego.',saltoLn,retornoCR,saltoLn,
'Elija una opci',OtildadaMinus,'n:',saltoLn,retornoCR,finCadena
menuIngreso db saltoLn,retornoCR,
'!#!#!#!#!#!#! INGRESO: !#!#!#!#!#!#!#!#!',
saltoLn,retornoCR,finCadena
menuRegistro db saltoLn,retornoCR,
'!#!#!#!#!#!#! REGISTRO: !#!#!#!#!#!#!#!#!',
saltoLn,retornoCR,finCadena
menuTops db saltoLn,retornoCR,
'!#!#!#!#!#!#! MENU TOPS !#!#!#!#!#!#!#!#!',saltoLn,retornoCR,
'(1)->Ver Top 10 Puntajes.',saltoLn,retornoCR,'
(2)->Ver Top 10 Tiempos.',saltoLn,retornoCR,'
(3)->Regresar.',saltoLn,retornoCR,saltoLn,
'Elija una opci',OtildadaMinus,'n:',saltoLn,retornoCR,finCadena
menuOrdenamientos db saltoLn,retornoCR,
'!#!#!#!#!#!#! ORDENAMIENTOSL !#!#!#!#!#!#!#!#!',saltoLn,retornoCR,
'(1)->Bubble Sort.',saltoLn,retornoCR,
'(2)->Quick Sort.',saltoLn,retornoCR,
'(3)->Shell Sort.',saltoLn,retornoCR,saltoLn,
'Elija una opci',OtildadaMinus,'n:',saltoLn,retornoCR,finCadena
menuOrden db saltoLn,retornoCR,
'!#!#!#!#!#!#! ORDEN !#!#!#!#!#!#!#!#!',
saltoLn,retornoCR,'(1)->Ascendente.',saltoLn,retornoCR,
'(2)->Descendente.',saltoLn,retornoCR,
'Elija una opci',OtildadaMinus,'n:',saltoLn,retornoCR,finCadena
```

Los menús se emplearon a base de cadenas con su respectiva distribución de caracteres, haciendo uso de la interrupción 21 para ser mostrados en consola. La implementación de equivalentes ayudo a la legibilidad en el uso de saltos de línea y de retornos de carro en cada una de ellas.

VERIFICACIÓN DE CREDENCIALES

Es la principal función que valida los usuarios existentes, tanto para los jugadores estándar como para los administradores, aperturando y haciendo una lectura previa del usuario, si este no coincide se desplaza al siguiente sucesivamente hasta encontrar una coincidencia y validar si este no existe. Ase manejaron distintos archivos para los administradores y para los jugadores estándar para evitar código redundante.

```
verificarIngreso macro
    LOCAL
 LecturaIngresoUsuario, ErrorEntradaUsuario, reVerificarUsuario, verificarUsuario,denegarCar
    PUSH SI
    PUSH DI
    MOV dl, 0 ;indica que se use la ruta del archivo de indice 0 (Gamers.txt)
    MOV bl, 1 ;indicador de apertura y lectura archivo
    accionarArchivoEnrutado dl, bl ;se abre y lee el contenido de se archivo
    cerrarArchivo controladorFicheros ;se cierra el archivo para evitar problemas posterior
    xor si, si ;inicializamos nuestros controles de indice
    xor di, di
    LecturaIngresoUsuario:
       imprimirEnConsola  solicitudUsuario
       obtenerLecturaTeclado lectorEntradaUsuario
       cmp cl, longMaxUsuario ;si es <= 7 se evalua, si no lo vuelve a solicitar
       jle verificarUsuario
       ErrorEntradaUsuario:
           imprimirEnConsola usuarioMaxError
           MOV dl, 1
           reiniciarLectorTeclado dl; reinicia el lector de usuario
           jmp LecturaIngresoUsuario
    reVerificarUsuario:
       xor si,si ;Se reinicia SI para evaluar la cadena ingresada nuevamente
       add di, 3
       ;dado que actualmente esta ubicado en un ";"(separador)
       ;necesitamos el siguiente caracter para evaluar otro usuario,
       ;pero sumamos 3 dado que es ";", "\n", y "\r" lo que separa a cada usuario
    verificarUsuario:
       cmp  lectorEntradaFicheros[di], finCadena
       je usuarioInexistente
       ;si coincide con fin de cadena significa que ya evaluo todo
       ;y no hay usuario registrado con ese nombre
       MOV bl, lectorEntradaFicheros[di]
       cmp  lectorEntradaUsuario[si], bl
       je aceptarCaracter
       denegarCaracter:
           ;dado que no coincide el usuario:
           ;se desplaza hasta encontrar ; (nuestro separador de credenciales)
           cmp lectorEntradaFicheros[di], puntoComa
           je  reVerificarUsuario
           inc di
           jmp denegarCaracter
       aceptarCaracter:
           inc si
           inc di
           cmp lectorEntradaUsuario[si], finCadena
           je  verificarCoincidenciaUsuario
           jmp verificarUsuario
    verificarCoincidenciaUsuario:
           cmp  lectorEntradaFicheros[di], coma ; si equivale a coma es que el usuario si c
           je  LecturaIngresoPass
           jmp denegarCaracter
```

DIBUJO DE ELEMENTOS GRAFICOS DEL JUEGO

Se emplearon distintas macros para facilitar el dibujo de los elementos del juego, elementos como la pelota, la barra o plataforma, los bloques y como todos ellos en conjunto formaron un nivel complejo y elaborado.

```
dibujarBloque macro
    ;En base a las posciones que se delimiten en los registros se dibuja el elemento
    LOCAL dibujar, final
    PUSH CX
    MOV cl, 0
    dibujar:
        PUSH AX;durante el trazado del dibujo estos registros se emplean,
        PUSH BX ;por lo que guardamos su valor original
        dibujarLineaHorizontal
        POP BX ;ahora podemos obtener la fila y columna originales orevias
        POP AX
        inc ax ;se incrementa la fila para que avance verticalmente
        inc cl
        cmp cl, alturaBloque;Es el número de lineas que formasn un bloque
        je final
        jmp dibujar
    final:
        POP CX
endm

dibujarContornosVerticales macro
    ;Nos posicionamos en la fila ax u en la columna bx,
    ;y con di indicamos cuantos pixeles vamos a pintar
    ;verticalmente a partir de ese punto
    MOV dl, colorBlancoGrafico
    MOV ax, 19;0f; 159
    MOV bx, 5 ;63h; 99
    MOV di, 175
    dibujarLineaVertical
    MOV dl, colorBlancoGrafico
    MOV ax, 19;0f; 159
    MOV bx, 314 ;63h; 99
    MOV di, 175
    dibujarLineaVertical
endm

dibujarContornosHorizontales macro
    ;Nos posicionamos en la fila ax u en la columna bx,
    ;y con di indicamos cuantos pixeles vamos a pintar
    ;horizontalmente a partir de ese punto
    MOV dl, colorBlancoGrafico
    MOV ax, 19;0f; 159
    MOV bx, 5 ;63h; 99
    MOV di, 310
    dibujarLineaHorizontal
    MOV dl, colorBlancoGrafico
    MOV ax, 194;0f; 159
    MOV bx, 5 ;63h; 99
    MOV di, 310
    dibujarLineaHorizontal
endm
```

CRONÓMETORS Y PUNTAJES

Se emplearon 2 arreglos equivalentes a el tiempo y los puntos acumulados en una partida, ambos son representados por sus centenas, decenas y unidades, esto facilita su lectura para los ordenamientos y así mismo facilita la forma de imprimirlo sobre la pantalla de modo video para que estos sean dinámicos.

```
incrementarPuntos proc
        ;inc DS:[Configurador.puntajeActual]
        ;0->centenas
        ;1->decenas
        ;2->unidades
        analizarCentenas:
            cmp visorPuntos[2], 9
            jl analizarInferiores
            cmp visorPuntos[1], 9
            jl analizarInferiores
            jmp aumentarCentenas
        analizarInferiores: ;decenas y centenas
            cmp visorPuntos[2], 9
            jl aumentarUnidades
            je aumentarDecenas
        aumentarUnidades:
            inc visorPuntos[2]
            jmp finDeSuma
        aumentarDecenas:
            MOV visorPuntos[2], 0
            inc visorPuntos[1]
            jmp finDeSuma
        aumentarCentenas:
            MOV visorPuntos[2], 0
            MOV visorPuntos[1], 0
            inc visorPuntos[0]
        finDeSuma:
            call imprimirPuntajeAcumulado
            ret
    incrementarPuntos endp

    incrementarTiempo proc
        ;inc DS:[Configurador.puntajeActual]
        ;0->centenas
        ;1->decenas
        ;2->unidades
        analizarCentenas:
            cmp tiempoEstable[2], 9
            jl analizarInferiores
            cmp tiempoEstable[1], 9
            jl analizarInferiores
            jmp aumentarCentenas
        analizarInferiores: ;decenas y centenas
            cmp tiempoEstable[2], 9
            jl aumentarUnidades
            je aumentarDecenas
        aumentarUnidades:
            inc tiempoEstable[2]
            jmp finDeSuma
        aumentarDecenas:
            MOV tiempoEstable[2], 0
            inc tiempoEstable[1]
            jmp finDeSuma
        aumentarCentenas:
            MOV tiempoEstable[2], 0
            MOV tiempoEstable[1], 0
            inc tiempoEstable[0]
        finDeSuma:
            ret
    incrementarTiempo endp
```

PELOTA Y DESPLAZAMIENTOS DE PLATAFORMA

Se emplearon distintos procedimientos con el fin de facilita los mismos, estos borran su posición actual y se desplazan a su nueva posición siendo nuevamente dibujados con sus respectivas validaciones en ejecución.

```
dibujarPlataforma proc
        call establecerSegmentoDatos
        MOV di, plataformaMovible.pixelesAncho
; las lineas horizontales de la plataforma tendran 40 pixeles de longitud
        MOV dl, plataformaMovible.colorActual
        MOV ax, plataformaMovible.filaActual
        MOV bx, plataformaMovible.columnaActual
        call establecerDireccionVideo
        dibujarBloque
        call establecerSegmentoDatos
        ret
    dibujarPlataforma endp

    borrarPlataformaActual proc
        call establecerSegmentoDatos
        MOV di, plataformaMovible.pixelesAncho
; las lineas horizontales de la plataforma tendran 40 pixeles de longitud
        MOV ax, plataformaMovible.filaActual
        MOV bx, plataformaMovible.columnaActual
        call establecerDireccionVideo
        borrarBloque
        call establecerSegmentoDatos
        ret
    borrarPlataformaActual endp

    dibujarPelotaEstandar proc
        call establecerSegmentoDatos
        MOV di, pelota.pixelesAncho ;
        MOV dl, pelota.colorActual
        MOV ax, pelota.filaActual
        MOV bx, pelota.columnaActual
        call establecerDireccionVideo
        dibujarPelota
        call establecerSegmentoDatos
        ret
    dibujarPelotaEstandar endp

    borrarPelotaActual proc
        call establecerSegmentoDatos
        MOV di, pelota.pixelesAncho ;
        MOV ax, pelota.filaActual
        MOV bx, pelota.columnaActual
        call establecerDireccionVideo
        borrarBloque
        call establecerSegmentoDatos
        ret
    borrarPelotaActual endp
```

```
;====================== MOVIMIENTOS ==========================

moverLadoIzquierdoPlataforma proc
    PUSH CX
    moverIzquierda:
        call borrarPlataformaActual
        call establecerSegmentoDatos
        dec  plataformaMovible.columnaActual
        call establecerDireccionVideo
        call dibujarPlataforma
        call establecerSegmentoDatos
    fin:
        POP CX
        ret
moverLadoIzquierdoPlataforma endp

moverLadoDerechoPlataforma proc
    PUSH CX
    moverDerecha:
        call borrarPlataformaActual
        call establecerSegmentoDatos
        inc  plataformaMovible.columnaActual
        call establecerDireccionVideo
        call dibujarPlataforma
        call establecerSegmentoDatos
    fin:
        POP CX
        ret
moverLadoDerechoPlataforma endp
```