Manual tecnico

String:

compareString:

Comapara dos strings, se ingresa en el si el inicio de un string y en el di el inicio del otro string y en el cx se ingresa el tamano de los strings, solo funciona para string con el mismo tamanno.

Retorna el resultado en la bandera zero.

```
compareString proc
    push bp
    mov bp, sp

    LOOP_START:
    mov al, [si]
    mov ah, [di]
    inc si ;los incs se hacen antes de la comparacion
    inc di
    cmp al, ah
    jne RETURN
    loop LOOP_START

    RETURN:
    pop bp
    ret
compareString endp
```

copyString:

Copia un string de un lugar en memoria a otro. En el registro si recibe el inicio del string origen y el di el inicio del string destino, en cx el tamano de el string origen

```
fillString proc

    push bp
    mov bp, sp

    LOOP_START:
    mov [si], al
    inc si
    loop LOOP_START

    pop bp
    ret

fillString endp
```

fillString:

Llena un string de tamanno cx e inicio si con el caracter almacenado en al

```
fillString proc

    push bp
    mov bp, sp

    LOOP_START:
    mov [si], al
    inc si
    loop LOOP_START

    pop bp
    ret

fillString endp
```

Metodos de lectura y escritura:

File:

openFile

Abre el archivo con direccion path

```
;Devuelbe en ax el handler, si ocurrio error flag carry esta en 1
openFile macro path
    mov ah, 3ch
    xor cx, cx
    lea dx, path
    int 21h
endm
```

writeContent

Escribe un string con tamano _size en el archivo con el handler handler

```
;returns: cx numero de bytes escritos, carry flag verdadera si ocurrio error
writeContent macro handler, _size, content
    mov bx, handler
    mov ah, 40h
    mov cx, _size
    lea dx, content
    int 21h
endm
```

closeFile

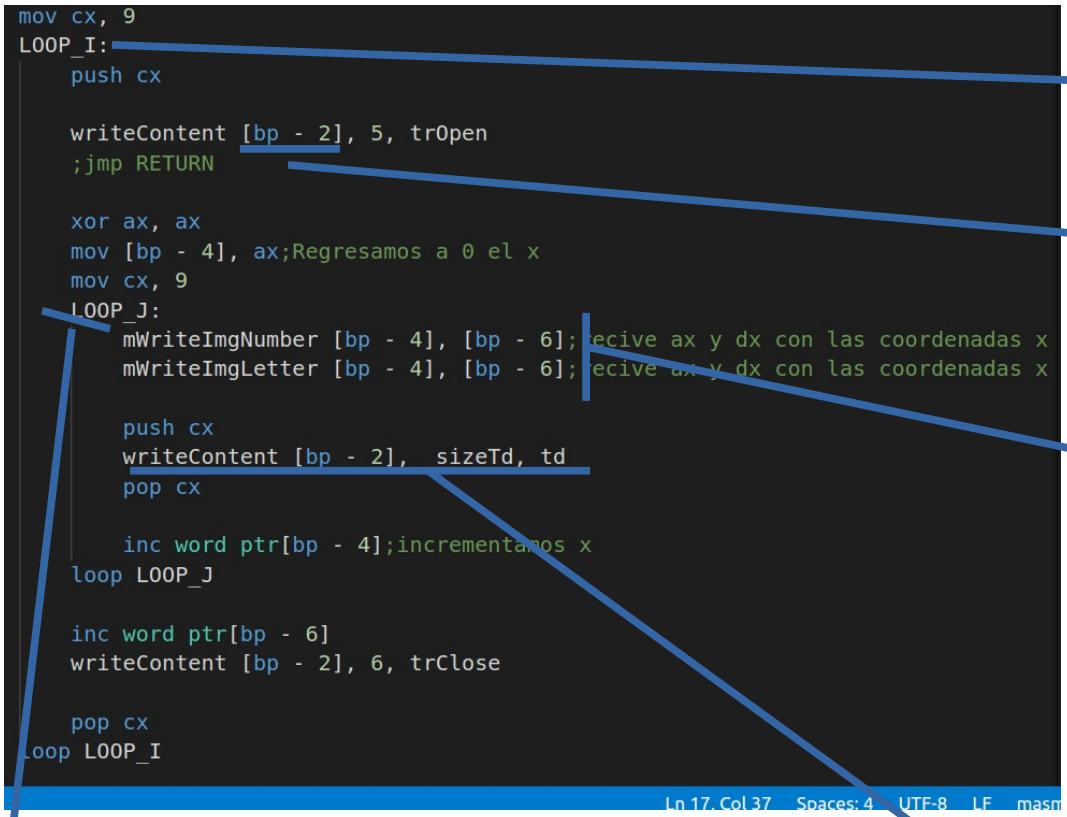
```
;carry flag verdadera si ocurrio error
closeFile macro handler
    mov ah, 3eh
    mov bx, handler ;bx todavia contiene el handler
    int 21h
endm
```

newFile

```
;retorna en ax el handler del nuevo archivo, carry flag si ocu
newFile macro path
LOCAL End
    ;makeFile
    mov ah, 3ch
    xor cx, cx
    lea dx, path
    int 21h
endm
```

Html:

Guardar todo el contenido de la tabla html en una o mas variables hubiera ocupado demasiado espacio. Por lo que solamente se guardo la el codigo que va antes de la tabla, el codigo que va despues de hacer la tabla y para llenar la tabla de celdas se usa el mismo string, solo se va modificando coordenada por coordenada dependiendo que pieza contiene cada una.



Recorre todas la coordenadas y

[bp – 2] contiene el handler del archivo que abrimos previamente

Modifica la cadena td para que tenga la imagen correcta, dependiendo de la pieza y coordenada en el tablero

Recorre todas las coordenadas x

Concatena la cadena td al archivo .html

Guardando estado actual:

Solo guardamos todo el tablero, las puntuaciones y el jugador en turno en un archivo.

```
SAVE:
debPrintln <"Ingrese nombre para guardar: ">
cmdGetString usrInput, 0
mMakeAndWriteFile usrInput, board0, 877
```

```
CARGAR_JUEGO:
    debPrintln <"Ingrese nombre del archivo: ">
    cmdGetString usrInput, 0
    mGetFileContent usrInput, board0, 877, ax
```

877 es el numero de bytes que ocupa el tablero, las puntuaciones y el jugador en turno actual, todas esas variables estan asignadas de manera continua, lo cual facilita su manejo

Metodos de juego

Lo mas importante del juego es ser capaces de obtener un cadena go dada una ficha en el tablero Se utilizo un metodo recursivo que revisaba todas las posiciones adyacentes a a la ficha y dependiendo que habia en ella se comportaba diferente.

- Si la coordenada esta marcada o es una ficha de diferente color no hacemos nada y pasamos al siguiente offset
- Si es una ficha del mismo color: Indicamos a que esta marcada para no visitarla dos veces y la visitamos
- Si la coordenada no tiene ninguna pieza Agregamos una libertad a la cadena actual

```
getGoString proc
    push bp
    mov bp, sp
    sub sp, 4

    mov cx, 4 ;numero de posiciones adyancentes
    lea bx, adyacentOffsets
    LOOP_I:
        mov ax, [bp + 4]
        mov dx, [bp + 6]
        add al, byte ptr[bx]
        add dl, byte ptr[bx + 1]
        mov [bp - 2], ax
        mov [bp - 4], dx

        ;revisamos que no este fuera de rango
        cmp ax, 8
        ja CONTINUE
        cmp dx, 8
        ja CONTINUE

        mIsMarked  [bp - 2], [bp - 4]
        cmp ax, 1
        je CONTINUE
```

```
    mGetPieceAt [bp - 2], [bp - 4]
    lea si, goObj
    cmp al, byte ptr[si] ;goObj.color
    je SAME_COLOR
    cmp al, 0
    je _EMPTY
    ;else. si no esta vacio, ni es del mismo color
    jmp CONTINUE
    _EMPTY:
        mGoStringAddLib [bp - 2], [bp - 4]
        jmp CONTINUE
    SAME_COLOR:
        mMark [bp - 2], [bp - 4]
        mGoStringAddCoordenate [bp - 2], [bp - 4]
        ;los registros cx y bx deben mantener sin
        ;, no se si estoy usando bien esa palabra
        push cx
        push bx
        mov dx, [bp - 4]
        push dx
        mov ax, [bp - 2]
        push ax
        call getGoString
```

```
    lea si, goObj
    cmp al, byte ptr[si] ;goObj.color
    je SAME_COLOR
    cmp al, 0
    je _EMPTY
    ;else. si no esta vacio, ni es del mismo color vamo
    jmp CONTINUE
    _EMPTY:
        mGoStringAddLib [bp - 2], [bp - 4]
        jmp CONTINUE
    SAME_COLOR:
        mMark [bp - 2], [bp - 4]
        mGoStringAddCoordenate [bp - 2], [bp - 4]
        ;los registros cx y bx deben mantener sin cambi
        ;, no se si estoy usando bien esa palabra
        push cx
        push bx
        mov dx, [bp - 4]
        push dx
        mov ax, [bp - 2]
        push ax
        call getGoString
```

```
    CONTINUE:
        add bx, 2
        dec cx ;Estas dos lineas reemplazan lo
        jne LOOP_I;puede hacer saltos pequenno


RETURN:
    mov sp, bp
    pop bp
    ret 4
getGoString endp
```