```
;=============================================================================;
;  GoASM                                                                      ;
; =======                                                                     ;
;                                                                             ;
; This is a two-player game based on the board-game Go.                       ;
; One player is white - the other black. Alternately each player places its   ;
; stone on the points on the board.                                           ;
;                                                                             ;
; = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = ;
; Made by:      Manuel Bellersen                                              ;
;  Matrikel:    46958                                                         ;
;  E-Mail:      manuel.bellersen@stud.htwk-leipzig.de                         ;
;  License:     GPLv3 or later                                                ;
;=============================================================================;
    .model small
    .486
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;  Constants
; ~~~~~~~~~~~
    esc_code = 1Bh              ;=> Codes of ASCII ESC,
    b_code   = 62h              ;   B,
    w_code   = 77h              ;   W,
    o_code   = 6Fh              ;   O,
    p_code   = 70h              ;   P,
    wav_code = 7Eh              ;   ~,
    CR = 13                     ;   Carriage Return
    LF = 10                     ;   and Line Feed                           <=|
    video_seg  = 0A000h         ; Video-Segment
    board_size = 12             ; Size of the board (square) -> 12x12
    field_size = 16             ; Size of a graphical field
    win_width  = 320            ; Width of the dos-window
    win_height = 200            ; Height of the dos-window
    next_y = board_size         ; Number of fields till next y
    next_x = 1                  ;              and till next x
    BLACK   = 0                 ;=> Symbolic constants for BLACK,
    WHITE   = 1                 ;   WHITE,
    TRUE    = 0                 ;   TRUE,
    FALSE   = 1                 ;   FALSE,
    FREE    = 0                 ;   FREE,
    UNFREE  = 1                 ;   UNFREE
    ALLFOUR = 4                 ;   and ALLFOUR (meaning all 4 directions)  <=|
    SGF_MAX = 256               ; Size of buffer to store placed stones for SGF
    S1         = 1000           ; One second is 1000ms
    MS55       = 55             ; Timer is called every 55ms
    SOUNDMAX   = 2              ; Sound is played maximal 2*55ms
    SOUND_HIGH = 00Bh           ; 16bit note to play ( G on 4th octave, 392Hz)
    SOUND_LOW  = 0E3h           ;
    .data
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;  Variables
; ~~~~~~~~~~~
    COUNT       DW 0h           ; Counting number of captured stones of a ERASE
    PointsBlack DW 0h           ; Holding number of captured stones of BLACK
    PointsWhite DW 0h           ;                                     and WHITE
    OldISRcs    DW 0h           ; Holding CS and
    OldISRbx    DW 0h           ;         BX of old ISR
    TimeAdd     DW 0h           ; Counter for Timer
    TimeLast    DW 0h           ; Holding last timer-var
    TimeBlack   DW 0h           ; Counter of seconds for BLACK
    TimeWhite   DW 0h           ;                     and WHITE
    TimeCounterBlack DW 0h      ; Counter of ms for BLACK
    TimeCounterWhite DW 0h      ;               and WHITE
    TimeSound   DW 0h           ; Counter for how long to hear the *beep*
    SOUND_OFF   DB 0h           ; Status of sound
    KISAVE      DW 0h           ; Saving overhead of random number
    SGF_FID     DW 0h             ; Pointer to file to save sgf-data
```

```
 SGFPATH     DB "myGoASM.sgf",0 ; Path to store file
 SGFERRORMSG DB "Could not create myGoASM.sgf","$"    ; FAIL and
 SGFOKMSG    DB "Saved your game at myGoASM.sgf.","$" ; OK message
 SGFDATACOLOR DW SGF_MAX DUP (0) ;=> Place to save the color,
 SGFDATAX DB SGF_MAX DUP (0)     ;   the X
 SGFDATAY DB SGF_MAX DUP (0)     ;   and Y coordinates for the sgf-file
 SGFNUMBER   DW 0000h            ; Number of free field to save next stone
 SGFRUNNER   DW 0000h            ; Running up to number of last field
 SGFSIGNS DB "abcdefghijkl","$"  ;=> Characters to use in SGF-file for X,Y,
 SGFB     DB "B","$"             ;   player BLACK,
 SGFW     DB "W","$"             ;   player WHITE,
 SGFBRACKETOPEN  DB "(","$"      ;   start,
 SGFBRACKETCLOSE DB ")","$"      ;   end,
 SGFBOXOPEN      DB "[","$"      ;   coordinate-start,
 SGFBOXCLOSE     DB "]","$"      ;   coordinate-end,
 SGFSEMICOLON    DB ";","$"      ;   group-end and
 SGFEXTRA   DB ";AP[GoASM:1.0]GM[1]SZ[12]","$" ; some static information
 BOARD      DW 144 DUP (0000h) ; The board where to play on (current  12x12)
 CHECKBOARD DB 144 DUP (0000h) ; Extra virtual board to set checked stones
 PLACED     DB 00h             ; Was stone placed: 0 OK    -- else Not OK
 NextPlayer DB 00h             ; Who is next  : 0 BLACK -- else WHITE
 AROUND     DB 00h             ; Is around OK : 0 OK    -- else Not OK
 LOOK       DB 00h             ; Look, is free: 0 Free  -- else not free
 COVAR      DB 00h             ; Is direction free: 0 Free  -- else not free
 YVAR       DW 00h             ; For restoring the Y value
 TEN        DB 10              ; "Constant" to dived to get single values
 StartX     DB 0h              ;=> Start-points for X and
 StartY     DB 0h              ;   Y for displaying HUD
 MYCOLOR      DW 0000h       ; Color of stone next to where to placed to
 OldMouseInt  DW 0000h       ; Saving old mouse routine
 OldMouseMask DW 0000h       ;    and old mouse mask
 MOUSEX       DW 0000h       ; Saving mouse x and
 MOUSEY       DW 0000h       ;                 y coordinates
 ; Following are the BLACK and WHITE stones and the empty field
BufferBlack DB field_size DUP (185, 185, 185, 185, 185, 185, 000, 000, 000, 000, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185, 185)
            DB field_size DUP (185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185)
            DB field_size DUP (185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185)
            DB field_size DUP (185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185)
            DB field_size DUP (000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000)
            DB field_size DUP (000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000)
            DB field_size DUP (000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000)
            DB field_size DUP (000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000)
            DB field_size DUP (185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185)
            DB field_size DUP (185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185)
            DB field_size DUP (185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185)
            DB field_size DUP (185, 185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 000, 000, 000, 000, 000, 000, 000, 000, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 000, 000, 000, 000, 185, 185, 185, 185, 185, 185)
BufferWhite DB field_size DUP (185, 185, 185, 185, 185, 185, 015, 015, 015, 015, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185, 185)
            DB field_size DUP (185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185)
            DB field_size DUP (185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185)
            DB field_size DUP (185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185)
            DB field_size DUP (015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015)
            DB field_size DUP (015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015)
            DB field_size DUP (015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015)
            DB field_size DUP (015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015)
            DB field_size DUP (185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185)
            DB field_size DUP (185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185)
            DB field_size DUP (185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185)
            DB field_size DUP (185, 185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 015, 015, 015, 015, 015, 015, 015, 015, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 015, 015, 015, 015, 185, 185, 185, 185, 185, 185)
BufferField DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 000, 015, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 015, 000, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
            DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
```

```
          DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
          DB field_size DUP (185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185, 185)
Cursor    DW 1111111111111111b   ;=> This will be how the mouse cursor will
          DW 1111000000011111b   ;   look like
          DW 1110000000000111b   ;   (first layer)
          DW 1100000000000011b
          DW 1000000000000011b
          DW 1000000000000001b
          DW 1000000000000001b
          DW 1000000000000001b
          DW 1000000000000001b
          DW 1000000000000001b
          DW 1000000000000001b
          DW 1100000000000001b
          DW 1100000000000001b
          DW 1110000000000011b
          DW 1111100000000111b
          DW 1111111111111111b   ;<=|
          DW 0000000000000000b   ;=> (second layer)
          DW 0000111100000000b
          DW 0001110000000000b
          DW 0011100001000000b
          DW 0111100011100000b
          DW 0111100001000000b
          DW 0111110000000000b
          DW 0111111000000000b
          DW 0111111110000000b
          DW 0111111111100000b
          DW 0111110011110000b
          DW 0011100001110000b
          DW 0011110011110000b
          DW 0001111111100000b
          DW 0000011110000000b
          DW 0000000000000000b   ;<=|
BufferHUD DB CR, LF,"                                        "      ; This
           DB CR, LF,"                             GoASM     "      ; is
           DB CR, LF,"                                        "      ; the
           DB CR, LF,"                        Captured BLACK "      ; HUD
           DB CR, LF,"                                      0 "
           DB CR, LF,"                                        "
           DB CR, LF,"                        Captured WHITE "
           DB CR, LF,"                                      0 "
           DB CR, LF,"                                        "
           DB CR, LF,"                        Time            "
           DB CR, LF,"                         B:           0 "
           DB CR, LF,"                         W:           0 "
           DB CR, LF,"                        Pass       Exit", "$"
; The next one is the message at the end of the game
GoodBye DB CR, LF, "     _  _  _                         _          _  _  _  _    _           _   "
        DB CR, LF, "  _ (_)(_)(_) _                    _(_)_      _(_)(_)(_)(_)_ (_) _     _ (_)  "
        DB CR, LF, " (_)         (_)    _  _  _      _(_) (_)_   (_)          (_)(_)(_)   (_)(_)  "
        DB CR, LF, " (_)    _  _  _  _ (_)(_)(_) _ _(_)     (_)_ (_)_  _  _  _   (_) (_)_(_) (_)  "
        DB CR, LF, " (_)   (_)(_)(_)(_)         (_)(_) _  _  _ (_) (_)(_)(_)(_)_ (_)   (_)   (_)  "
        DB CR, LF, " (_)         (_)(_)         (_)(_)(_)(_)(_)(_) _          (_)(_)         (_)  "
        DB CR, LF, " (_) _  _  _ (_)(_) _  _  _ (_)(_)         (_)(_)_  _  _ _(_)(_)         (_)  "
        DB CR, LF, "    (_)(_)(_)(_)   (_)(_)(_)   (_)         (_) (_)(_)(_)(_)  (_)         (_)  "
        DB CR, LF, "                                                                              "
        DB CR, LF, " "
        DB CR, LF, "          _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ "
        DB CR, LF, "         (_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)"
        DB CR, LF, "         (_)                                                      (_)"
        DB CR, LF, "         (_)   Made by  Manuel Bellersen                          (_)"
        DB CR, LF, "         (_)   E-Mail   manuel.bellersen@stud.htwk-leipzig.de     (_)"
        DB CR, LF, "         (_)   Release  English version                           (_)"
        DB CR, LF, "         (_)   Date     10/01/2010                                (_)"
        DB CR, LF, "         (_)   Files    01/01                                     (_)"
        DB CR, LF, "         (_)   License  GPLv3 or later                            (_)"
        DB CR, LF, "         (_) _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _  _ (_)"
        DB CR, LF, "         (_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)(_)"
        DB CR, LF, " "
        DB CR, LF, "               ", "$"
.stack    8000h              ; Stack is set to 32K, hope this is enough
.code
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

```
;  Code
; ~~~~~~
GoASM  proc                    ;-------------------------
    xor ax, ax                 ; Let's start the program!
    mov ax, @DATA              ;-------------------------
    mov ds, ax                 ;=> Loading ds with DATA
    mov ax, video_seg          ;
    mov es, ax                 ;       and es with VIDEO
    mov ax, 13h                ;=> Setting video-mode 13h
    int 10h                    ;   (640 x 480 x 16)
    lea dx, BufferHUD          ;=> Showing text-hud on screen
    mov ah, 09h                ;
    int 21h                    ;
    call DRAWBOARD             ; Drawing the board
    call SETMOUSE              ; Setting up the mouse
    call SETTIMER              ; Setting up the timer
KeyCode: mov ax, 0100h         ;=> Check for keystroke
         int 16h               ;
         jz RefreshHUD         ;
         xor ah, ah            ;=> Query key-code
         int 16h               ;   <ESC> to quit
         cmp al, esc_code      ;   <B>   to set a BLACK stone
         je GoASM_END          ;   <W>   to set a WHITE stone
         cmp al, b_code        ;   <O>   to set a stone of current player
         je KeyBlack           ;   <P>   to pass to next player
         cmp al, w_code        ;   <~>   to call MOUSEONHUD
         je KeyWhite
         cmp al, o_code
         je KeyKIMove
         cmp al, p_code
         je KeyPass
         cmp al, wav_code
         je KeyMouse
         jmp RefreshHUD
KeyBlack: cmp NextPlayer, WHITE ;=> Only if BLACK has pressed <B> the stone
          je RefreshHUD        ;   will appear
          jmp KeyKIMove
KeyWhite: cmp NextPlayer, BLACK ;=> Only if WHITE has pressed <W> the stone
          je RefreshHUD        ;   will appear
KeyKIMove: call KIMOVE         ;=> Try to set a stone at a random place
           jmp RefreshHUD
KeyPass: mov cx, 420           ;=> Set position of PASS and change player
         mov dx, 187
         call MOUSEONHUD
         jmp RefreshHUD
KeyMouse: mov cx, MOUSEX       ;=> Set position of mouse-xy and make a move
          mov dx, MOUSEY
          call MOUSEONHUD
          jmp RefreshHUD
RefreshHUD: call HUDTIMER      ;=> Refresh the HUD
            jmp KeyCode
GoASM_END: call UNSETMOUSE     ; Reset the mouse
           call UNSETTIMER     ; Reset the interrupt
           mov ax, 3           ;=> Setting standard video-mode 3h
           int 10h             ;   (80 x 25)
           lea dx, GoodBye     ;=> Showing end-message on screen
           mov ah, 09h         ;
           int 21h             ;
           call SAVESGF        ; Save the sgf-file
           mov ah, 4Ch         ;=> Going back to DOS
           int 21h             ;
           ret
GoASM endp
SETTIMER proc             ;-------------------------------------------------
    cli                   ; Setting up the timer interrupt to use TIMER
    mov ah, 35h           ;< OldISRcs, OldISRbx - cs, bx of  old ISR
```

```
    mov al, 1Ch           ;-----------------------------------------------------
    int 21h                     ; Get the current interrupt handler
    push es
    pop ax
    mov OldISRcs, ax            ; Save code-segment and
    mov OldISRbx, bx            ; function of used interrupt handler
    push cs                     ;=> Set own TIMER as interrupt handler
    pop ds
    mov dx, OFFSET TIMER
    mov ah, 25h
    mov al, 1Ch
    int 21h                     ;<=|
    sti
    push @DATA                  ;=> Restore
    pop ds                      ;   data
    push video_seg              ;   and
    pop es                      ;   extended segment                        <=|
    ret
SETTIMER endp
UNSETTIMER proc           ;-----------------------------------------------------
    push ds               ; Setting back to the previous used interrupt handler
    cli                   ;-----------------------------------------------------
    mov dx, OldISRbx            ;=> Set back to the old interrupt handler
    push OldISRcs
    pop ds
    mov ah, 25h
    mov al, 1Ch
    int 21h                     ;<=|
    sti
    pop ds
    ret
UNSETTIMER endp
SAVESGF proc                ;-------------------------------------------------
    push @DATA              ; Create the SGF-file and insert data
    pop ds                  ;-------------------------------------------------
    mov dx, OFFSET SGFPATH      ;=> Try to delete the file
    xor cx, cx
    mov ax, 4100h
    int 21h                     ;<=|
    mov dx, OFFSET SGFPATH      ;=> Try to create a new file
    mov ax, 3C00h
    xor cx, cx
    int 21h                     ;<=|
    jc SGF_ERROR                ; Check if an error occurred
    mov SGF_FID, ax             ; Save the FileID
    mov dx, OFFSET SGFBRACKETOPEN   ;=> Write starting-bracket
    call WRITECHAR                  ;
    mov dx, OFFSET SGFEXTRA         ;=> Write some extra information
    mov ax, 4000h
    mov bx, SGF_FID
    mov cx, 25
    int 21h                         ;<=|
    call WRITESGF                   ; Write data of placed stones
    mov dx, OFFSET SGFBRACKETCLOSE  ;=> Write the closing-bracket
    call WRITECHAR                  ;
    jmp SGF_OK
SGF_ERROR: mov dx, OFFSET SGFERRORMSG   ; Load FAIL-message
           jmp SGF_SHOWMSG
SGF_OK: mov bx, SGF_FID                 ;=> Close the open file
        mov ax, 3E00h
        int 21h                         ;<=|
        mov dx, OFFSET SGFOKMSG         ; Load OK-Message
SGF_SHOWMSG: mov ax, 900h               ;=> Show the message
             int 21h                    ;
SGF_END: ret
SAVESGF endp
```

```
WRITESGF proc                           ;---------------------------------------
    pusha                               ; Write the SGF-data into the file
WRITERETRY: mov dx, OFFSET SGFSEMICOLON ;---------------------------------------
            call WRITECHAR              ; Write GroupEnd
            mov di, OFFSET SGFDATACOLOR ;=> See if there is another data-block
            shl SGFRUNNER, 1
            add di, SGFRUNNER
            shr SGFRUNNER, 1
            cmp WORD PTR [di], 0
            je WRITEEND                 ;<=|
            cmp WORD PTR [di], OFFSET BufferBlack   ; See who is current player
            jne WRITEWHITE
            mov dx, OFFSET SGFB         ; Set BLACK player
            jmp WRITECOLOR
WRITEWHITE: mov dx, OFFSET SGFW         ; Set WHITE player
WRITECOLOR: call WRITECHAR              ; Write current player
            mov dx, OFFSET SGFBOXOPEN   ; Open the BOX
            call WRITECHAR
            mov dx, OFFSET SGFSIGNS     ;=> Write the x-coordinate
            xor ax, ax
            mov di, OFFSET SGFDATAX
            add di, SGFRUNNER
            add al, BYTE PTR [di]
            add dx, ax
            call WRITECHAR              ;<=|
            mov dx, OFFSET SGFSIGNS     ;=> Write the y-coordinate
            xor ax, ax
            mov di, OFFSET SGFDATAY
            add di, SGFRUNNER
            add al, BYTE PTR [di]
            add dx, ax
            call WRITECHAR              ;<=|
            mov dx, OFFSET SGFBOXCLOSE  ; Close the BOX
            call WRITECHAR
            inc SGFRUNNER               ; Next pair, please
            cmp SGFRUNNER, SGF_MAX-1    ; Check if this was all
            jae WRITEEND
            mov di, OFFSET SGFDATACOLOR ;=> Check if we have more pairs
            shl SGFRUNNER, 1
            add di, SGFRUNNER
            shr SGFRUNNER, 1
            cmp WORD PTR [di], 0
            jne WRITERETRY              ;<=|
WRITEEND: popa
          ret
WRITESGF endp
WRITECHAR proc          ;-----------------------------------------------------
    mov ax, 4000h       ; Writing a character to a file.
    mov bx, SGF_FID     ;> dx - Character to write
    mov cx, 1           ;> SGF-FID - File to write into
    int 21h             ;-----------------------------------------------------
    ret
WRITECHAR endp
TIMER proc far          ;-----------------------------------------------------
    push ds             ; Add 55ms to TimeAdd each interruption
    push @DATA          ;< TimeAdd - adding 55ms
    pop ds              ;-----------------------------------------------------
    add TimeAdd, MS55
    pop ds
    iret
TIMER endp
HUDTIMER proc           ;-----------------------------------------------------
    push ax             ; React on time-changes, toggle sound off, refresh HUD
    push ds             ;< TimeAdd, TimeLast - 0 if equal
    push @DATA          ;< TimeSound, TimeCounter*, TimeBlack, TimeWhite
    pop ds              ;< StartX, StartY
```

```
    mov ax, TimeLast    ;-------------------------------------------------------
    cmp TimeAdd, ax             ; Check if time change since last time
    je HUDTIMER_NOTIME
    mov TimeAdd, 0
    mov TimeLast, 0             ;<=|
    cmp SOUND_OFF, TRUE         ;=> Check if sound needs to be turned off
    je HUDTIMER_COLOR
    inc TimeSound
    cmp TimeSound, SOUNDMAX
    jb HUDTIMER_COLOR
    call STOPSOUND              ;<=|
HUDTIMER_COLOR: cmp NextPlayer, WHITE       ; See who is current player
                je HUDTIMER_WHITE
                add TimeCounterBlack, MS55  ;=> Check for time-changes for
                cmp TimeCounterBlack, S1    ;   BLACK
                jb HUDTIMER_END
                sub TimeCounterBlack, S1
                inc TimeBlack               ;<=|
                jmp HUDTIMER_END
HUDTIMER_WHITE: add TimeCounterWhite, MS55  ;=> Check for time-changes for
                cmp TimeCounterWhite, S1    ;   WHITE
                jb HUDTIMER_END
                sub TimeCounterWhite, S1
                inc TimeWhite               ;<=|
HUDTIMER_END: mov StartX, 38    ;=> Show time of BLACK
              mov StartY, 19
              mov ax, TimeBlack
              call INSERTNUMBER ;<=|
              mov StartX, 38    ;=> Show time of WHITE
              mov StartY, 21
              mov ax, TimeWhite
              call INSERTNUMBER ;<=|
HUDTIMER_NOTIME: pop ds
                 pop ax
                 ret
HUDTIMER endp
SHOWMOUSE proc      ;-------------------------------------------------------------
    push ax         ; Shows the mouse cursor
    mov ax, 1       ;-------------------------------------------------------------
    int 33h
    pop ax
    ret
SHOWMOUSE endp
HIDEMOUSE proc      ;-------------------------------------------------------------
    push ax         ; Hides the mouse cursor
    mov ax, 2       ;-------------------------------------------------------------
    int 33h
    pop ax
    ret
HIDEMOUSE endp
SETMOUSE proc       ;-------------------------------------------------------------
    push es         ; Sets the mouse
    push @DATA      ;< OldMouse* - mask and interrupt handler of old mouse
    pop es          ;-------------------------------------------------------------
    xor ax, ax                  ; Reset mouse
    int 33h
    call SHOWMOUSE              ; Shows the mouse cursor
    mov dx, OFFSET Cursor       ;=> Setting cursor icon
    mov bx, 7h
    mov cx, 7h
    mov ax, 9
    int 33h                     ;<=|
    push @CODE
    pop es
    mov ax, 0014h               ;=> Setting routine for mouse-interaction
    mov cx, 1010100b            ;   Mouse-buttons 1,2,3 release
```

```
    mov dx, OFFSET MOUSEROUTINE
    int 33h                     ;   Exchanging Interrupts                    <=|
    mov OldMouseMask, cx        ; Saving old mask
    mov OldMouseInt, dx         ;        and interrupt
    pop es
    ret
SETMOUSE endp
UNSETMOUSE proc      ;-------------------------------------------------------
    push es          ; Resets the mouse.
    push @CODE       ;-------------------------------------------------------
    pop es
    mov ax, 0014h               ;=> Restoring old mouse-routine
    mov cx, OldMouseMask
    mov dx, OldMouseInt
    int 33h                     ;<=|
    pop es
    call HIDEMOUSE              ; Hiding the mouse cursor
    xor ax, ax                  ; Resetting the mouse
    int 33h
    ret
UNSETMOUSE endp
MOUSEROUTINE proc far   ;-------------------------------------------------------
    push ds             ; Saving mouse-xy and pushing the wave
    push @DATA          ; (mouse-interrupt-hanlder)
    pop ds              ;< MOUSEX, MOUSEY - mouse-coordinates
    push cx             ;-------------------------------------------------------
    pop MOUSEX
    push dx
    pop MOUSEY
    mov cx, wav_code            ;=> Push wave-keystroke
    mov ax, 500h
    int 16h                     ;<=|
    pop ds
    ret
MOUSEROUTINE endp
MOUSEONHUD proc         ;-------------------------------------------------------
    pusha               ; Handles mouse-button pressing and releasing.
    call HIDEMOUSE      ;> CX - cursor column
    push @DATA          ;> DX - cursor row
    pop ds              ;< NextPlayer, PLACED
    push video_seg      ;-------------------------------------------------------
    pop es
    mov ax, cx                  ;=> Using ax and bx for X/2 and Y
    shr ax, 1
    mov bx, dx                  ;<=|
    cmp NextPlayer, BLACK       ; Checking who is current player
    jnz NEXT_WHITE
    mov dx, OFFSET BufferBlack  ;=> Loading BLACK
    mov NextPlayer, WHITE       ;   Next player : WHITE
    jmp NEXT_END
NEXT_WHITE: mov dx, OFFSET BufferWhite  ;=> Loading WHITE
            mov NextPlayer, BLACK       ;   Next player : BLACK
NEXT_END:   call MAKEBOARDPOSITION      ; Changing X & Y to board-positions
            cmp bx, 176                 ; If Y > 176
            jnbe BACK                   ;   then OutOfEverything
            cmp ax, 176                 ; If X > 176
            jnbe MENU                   ;   then check menu
            call PLACEONBOARD           ; Try placing on board
            cmp PLACED, TRUE            ;=> Check if placing was OK
            jne BACK                    ;
            call STARTSOUND             ; Start the KLACK-sound
            call PLACESTONE             ; Graphical placing the stone
            jmp ENDIT
MENU:   cmp bx, 176             ; If Y < 176
        jb BACK                 ;   then OutOfMenu
        cmp ax, 192             ; If X < 192
```

```
        jb BACK                 ;   then OutOfMenu
        cmp ax, 208             ; If X > 208
        ja TRYEXIT              ;   then maybe Exit pushed
        jmp BACK_END            ; Don't set color-change back because PASS
TRYEXIT:    cmp ax, 288         ; If X < 288
            jb BACK             ;   then not Exit pushed
            mov cx, esc_code    ;=> Push ESC-keystroke
            mov ax, 500h        ;
            int 16h             ;<=|
BACK:   cmp NextPlayer, BLACK           ;=> No stone was placed so same player
        jne BACK_WHITE                  ;   is still on the move. Need to take
        mov dx, OFFSET BufferBlack      ;   back the changes.
        mov NextPlayer, WHITE
        jmp BACK_END
BACK_WHITE: mov dx, OFFSET BufferWhite
            mov NextPlayer, BLACK
BACK_END:   mov PLACED, TRUE            ;<=|
            jmp ENDIT
ENDIT:  call SHOWPOINTS         ; Updating capture-points
        call SHOWMOUSE          ; Unhides mouse cursor
        popa
        ret
MOUSEONHUD endp
STARTSOUND proc         ;-------------------------------------------------------
    push ax             ; Starting the KLACK-sound
    in al, 61h          ;< SOUND_OFF - false , TimeSound - 0
    or al, 3            ;-------------------------------------------------------
    out 61h, al                 ; Turn speaker on
    mov al, 0B6h
    out 43h, al
    mov al, SOUND_LOW           ;=> Push note
    out 42h, al
    mov al, SOUND_HIGH
    out 42h, al                 ;<=|
    mov SOUND_OFF, FALSE        ; Sound is on
    mov TimeSound, FREE         ; Time for sound is 0
    pop ax
    ret
STARTSOUND endp
STOPSOUND proc          ;-------------------------------------------------------
    push ax             ; Stopping the KLACK-sound
    in al, 61h          ;< SOUND_OFF - true
    and al, 0FCh        ;-------------------------------------------------------
    out 61h, al                 ; Turn speaker off
    mov SOUND_OFF, TRUE         ; Sound is off
    pop ax
    ret
STOPSOUND endp
SHOWPOINTS proc         ;-------------------------------------------------------
    push ax             ; Showing points of BLACK and WHITE
    mov StartX, 38      ;< StartX, StartY
    mov StartY, 7       ;-------------------------------------------------------
    mov ax, PointsBlack         ; Showing points of black
    call INSERTNUMBER           ;<=|
    mov StartX, 38              ;=> Showing points of white
    mov StartY, 13
    mov ax, PointsWhite
    call INSERTNUMBER           ;<=|
    pop ax
    ret
SHOWPOINTS endp
INSERTNUMBER proc       ;-------------------------------------------------------
    push ax             ; Showing a number on HUD
    push cx             ;> ax - number to show
    push dx             ;-------------------------------------------------------
IN_INSERT:  call SETSTART       ; Setting start-point
```

```
            div TEN             ; Get last value
            add ah, 30h         ; Transform into ASCII
            mov cx, ax          ;=> Show ASCII-character on screen
            mov dl, ah
            mov ax, 200h
            int 21h             ;<=|
            mov ax, cx
            xor ah,ah
            dec StartX          ; Set next place
            cmp ax, 0           ; See if there is more
            jne IN_INSERT
            pop dx
            pop cx
            pop ax
            ret
INSERTNUMBER endp
SETSTART proc           ;-------------------------------------------------------
    push ax             ; Setting start to show a character on the HUD
    push bx             ;> StartX - value to start for X
    push dx             ;> StartY - value to start for Y
    mov dh, StartY      ;-------------------------------------------------------
    mov dl, StartX
    xor bx, bx
    mov ax, 200h
    int 10h
    pop dx
    pop bx
    pop ax
    ret
SETSTART endp
MAKEBOARDPOSITION proc  ;-------------------------------------------------------
    sub ax, 4           ; Making xy-coordinates to exact start-position on the
    shr ax, 4           ; board
    shl ax, 4           ;<> ax - X
    sub bx, 4           ;<> bx - Y
    shr bx, 4           ;-------------------------------------------------------
    shl bx, 4
    ret
MAKEBOARDPOSITION endp
PLACEONBOARD proc       ;-------------------------------------------------------
    push ax             ; Try to place the stone and add that to SGF
    push bx             ;> ax - X
    push @DATA          ;> bx - Y
    pop ds              ;> dx - Offset <Color>
    push video_seg      ;-------------------------------------------------------
    pop es
    shr ax, 4                   ;=> Making board 12x12
    shr bx, 4                   ;
    call CHECKANDPLAY           ; Checking and placing on board
    cmp PLACED, FALSE           ; If failed
    je CHECKFAILED              ;   jump to end
    call ADDTOSGF               ; Adding the stone to SGF
CHECKFAILED:    pop bx
                pop ax
                ret
PLACEONBOARD endp
ADDTOSGF proc                   ;-----------------------------------------------
    cmp SGFNUMBER, SGF_MAX-1    ; Adding the currently placed stone to SGF-Data
    jae ADD_END                 ;> al - X
    push ax                     ;> bl - Y
    push di                     ;> dx - Offset <Color>
    mov di, OFFSET SGFDATACOLOR ;< SGFNUMBER - +1, if possible
    shl SGFNUMBER, 1            ;-----------------------------------------------
    add di, SGFNUMBER           ; Saving color
    shr SGFNUMBER, 1
    mov [di], dx                ;<=|
```

```
    mov di, OFFSET SGFDATAX     ;=> Saving X
    add di, SGFNUMBER
    mov BYTE PTR [di], al       ;<=|
    mov di, OFFSET SGFDATAY     ;=> Saving Y
    add di, SGFNUMBER
    mov BYTE PTR [di], bl       ;<=|
    inc SGFNUMBER               ; Setting next free place
    pop di
    pop ax
ADD_END: ret
ADDTOSGF endp
CHECKANDPLAY proc           ;-------------------------------------------------
    push ax                 ; Checks if the place where to play on is OK and
    push bx                 ; cleans up to remove captured stones
    mov PLACED, TRUE        ;> ax - X
    imul bx, board_size     ;> bx - Y
    add ax, bx              ;> dx - Offset <Color>
    shl ax, 1               ;< PLACED - TRUE if stone was placed
    mov bx, OFFSET BOARD    ;-------------------------------------------------
    add bx, ax                  ; Checking if current position is free
    shr ax, 1
    cmp WORD PTR [bx], FREE
    je ITSOK
    mov PLACED, FALSE           ;<=|
    jmp CAPEND
ITSOK:  call VISIT              ; Visit current place
        mov WORD PTR [bx], dx   ; Setting stone on board
        call LOOKAROUND         ; Check if placing is OK
        call UNVISIT            ; Unvisit current place
        cmp AROUND, FREE        ;=> See if around was free
        je CAPEND               ;
        mov WORD PTR [bx], FREE ;
        mov PLACED, FALSE       ;<=|
CAPEND: pop bx
        pop ax
        ret
CHECKANDPLAY endp
LOOKAROUND proc         ;-------------------------------------------------------
    push ax             ; Checking the fields around a field.
    push cx             ;> ax - BYTE-XY-coordinate
    mov AROUND, ALLFOUR ;> dx - Offset <Color>
    mov LOOK, FREE      ;< AROUND - changed according CHECKONE, 0-OK, 1-fail
    xor cx, cx          ;-------------------------------------------------------
    inc cl                      ;/////////// Field above /////////
    cmp ax, board_size          ; If XY < board_size
    jb RIGHTFIELD               ;   then no field above
    dec cl                      ;
    sub ax, next_y              ; Get field above
    call CHECKONE               ; Check it
    add ax, next_y              ; Get back
    add cl, LOOK                ; See if flag was set
RIGHTFIELD: inc cl              ;/////////// Field right /////////
            push ax
            inc ax              ; Make XY -> 1-144
            call MODSIZE        ; See if we are right
            cmp ax, 0           ; If right == 0
            pop ax              ;
            je DOWNFIELD        ;   then no field on right
            dec cl              ;
            inc ax              ; Set right field
            call CHECKONE       ; Check it
            dec ax              ; Set back
            add cl, LOOK        ; See if flag was set
DOWNFIELD:  inc cl              ;/////////// Field below /////////
            cmp ax, 131         ; If XY > 11*12-1
            ja LEFTFIELD        ;   then no field below
```

```
            dec cl              ;
            add ax, next_y      ; Get field below
            call CHECKONE       ; Check it
            sub ax, next_y      ; Get back
            add cl, LOOK        ; See if flag was set
LEFTFIELD:  inc cl              ;/////////// Field left /////////
            push ax
            call MODSIZE        ; See if we are left
            cmp ax, 0           ; If left == 0
            pop ax              ;
            je ENDFIELD         ;   then no field on left
            dec cl              ;
            dec ax              ; Set left field
            call CHECKONE       ; Check it
            inc ax              ; Set back
            add cl, LOOK        ; See if flag was set
ENDFIELD:   and AROUND, cl      ; See if all 4 directions are unfree
            pop cx              ; if yes then AROUND is UNFREE, else FREE
            pop ax
            ret
LOOKAROUND endp
MODSIZE proc                    ;-----------------------------------------------
    mov YVAR, 0                 ; Divides al by board_size
MODSTART:   cmp ax, board_size  ;<> ax - remainder
            jb MODEND           ;< YVAR - quotient
            sub ax, board_size  ;-----------------------------------------------
            inc YVAR
            jmp MODSTART
MODEND: ret
MODSIZE endp
CHECKONE proc           ;-------------------------------------------------------
    push ax             ; Checks a field and its surroundings
    push bx             ;> ax - BYTE-XY-coordinate
    push cx             ;> dx - Offset <Color>
    push dx             ;< LOOK - FREE - OK, UNFREE - not free
    mov bx, OFFSET BOARD;-------------------------------------------------------
    shl ax, 1                   ; Check if this field is free
    add bx, ax
    shr ax, 1
    cmp WORD PTR [bx], FREE
    jne CONOTOK
    mov LOOK, FREE              ;<=|
    jmp COEND
CONOTOK: mov cx, [bx]
         mov MYCOLOR, cx        ; Save current stone-color
         call CHECKAROUND       ; Check fields around
         call UNVISIT           ; Unvisit all stones
         cmp cx, dx             ;=> Check if color of this stone is same or
         jne CODIFF             ;   different then which is placed
         mov bl, COVAR
         mov LOOK, bl           ;<=|
         jmp COEND
CODIFF: mov LOOK, UNFREE        ; For different stones: look = !covar
        cmp COVAR, FREE         ;  (other had free place)
        je COEND                ;
        mov LOOK, FREE          ; else it will be free
        mov COUNT, 0            ;////////// Clearing captured /////////
        call ERASE              ; Remove captured stones
        call UNVISIT            ; Unvisit all stones
        cmp dx, OFFSET BufferBlack  ;////////// Adding captured /////////
        jne ADDWHITE
        mov cx, PointsBlack         ;=> to BLACK
        add cx, COUNT
        mov PointsBlack, cx         ;<=|
        jmp COEND
ADDWHITE:   mov cx, PointsWhite     ;=> to WHITE
```

```
            add cx, COUNT
            mov PointsWhite, cx     ;<=|
COEND:  pop dx
        pop cx
        pop bx
        pop ax
        ret
CHECKONE endp
VISIT proc                      ;-----------------------------------------------
    push bx                     ; Sets a place on CHECKBOARD to UNFREE
    mov bx, OFFSET CHECKBOARD   ;> ax - place to unfree
    add bx, ax                  ;-----------------------------------------------
    mov BYTE PTR [bx], UNFREE
    pop bx
    ret
VISIT endp
UNVISIT proc                    ;-------------------------------------
    push bx                     ; Sets hole CHECKBOARD to FREE
    push cx                     ;-------------------------------------
    mov cx, 72
    mov bx, OFFSET CHECKBOARD
UN72:   mov WORD PTR [bx], FREE
        add bx, 2
        loop UN72
        pop cx
        pop bx
        ret
UNVISIT endp
CHECKAROUND proc        ;-------------------------------------------------------
    push ax             ; Recursive check of field and surroundings
    push bx             ;> ax - BYTE-XY-coordinate
    push cx             ;> dx - Offset <Color>
    mov COVAR, UNFREE   ;< COVAR - FREE / UNFREE
    mov bx, OFFSET BOARD;-------------------------------------------------------
    shl ax, 1                   ; Is this field free ?
    add bx, ax
    shr ax, 1
    cmp WORD PTR [bx], FREE
    je CA_FREEEND               ;   (Yes)
    mov cx, [bx]                ;   (No)                                    <=|
    cmp cx, MYCOLOR             ; If color != mycolor
    jne CA_END                  ;   then end
    mov bx, OFFSET CHECKBOARD   ;=> Check the CheckBoard
    add bx, ax
    cmp BYTE PTR [bx], UNFREE
    je CA_END                   ;   (visited)
    mov BYTE PTR [bx], UNFREE   ;   (not visited)                           <=|
    cmp ax, board_size          ;////////// Check around this field /////////
    jb CA_RIGHT                 ; See if there is a field above
    sub ax, next_y              ; Set to above
    call CHECKAROUND            ; Check that
    add ax, next_y              ; Set back
    cmp COVAR, FREE             ; See what happened
    je CA_END                   ; End if free
CA_RIGHT:   push ax             ;////////// RIGHT /////////
            inc ax              ; Make 1-144
            call MODSIZE
            cmp ax, 0           ; See if ax was multiple of 12
            pop ax
            je CA_BELOW         ;   then no right there
            inc ax              ; Set right field
            call CHECKAROUND    ; Check it
            dec ax              ; Set back
            cmp COVAR, FREE     ; See what happened
            je CA_END           ; End if free
CA_BELOW:   cmp ax, 131         ;////////// BELOW /////////
```

```
            ja CA_LEFT          ; See if there is a field below
            add ax, next_y      ; Set to below
            call CHECKAROUND    ; Check it
            sub ax, next_y      ; Set back
            cmp COVAR, FREE     ; See what happened
            je CA_END           ; End if free
CA_LEFT:    push ax             ;/////////// LEFT /////////
            call MODSIZE
            cmp ax, 0           ; See if mod is 0
            pop ax
            je CA_END           ; If 0 then no left field then end
            dec ax              ; Set left
            call CHECKAROUND    ; Check it
            inc ax              ; Set back
            jmp CA_END
CA_FREEEND: mov COVAR, FREE     ; Free to play
CA_END:     pop cx
            pop bx
            pop ax
            ret
CHECKAROUND endp
ERASE proc              ;-------------------------------------------------------
    push ax             ; Erase all bordering stones of same color
    push bx             ;> ax - BYTE-XY-coordinate
    push cx             ;> MYCOLOR - color of stone to erase
    mov COVAR, UNFREE   ;< COVAR - FREE / UNFREE
    mov bx, OFFSET BOARD;< COUNT - number of erased stones
    shl ax, 1           ;-------------------------------------------------------
    add bx, ax                  ; Is this field free
    shr ax, 1
    cmp WORD PTR [bx], FREE
    je ERASE_FREEEND            ;   (Yes)
    mov cx, [bx]                ;   (No)
    cmp cx, MYCOLOR             ; If color != mycolor
    jne ERASE_END               ;   then end
    mov bx, OFFSET CHECKBOARD   ;=> Check the CheckBoard
    add bx, ax
    cmp BYTE PTR [bx], UNFREE
    je ERASE_END                ;   (visited)
    mov BYTE PTR [bx], UNFREE   ;   (not visited)                              <=|
    xor cx, cx                  ;/////////// Check around this field /////////
    cmp ax, board_size          ; See if there is a field above
    jb ERASE_RIGHT
    sub ax, next_y              ; Set to above
    call ERASE                  ; Check that
    add ax, next_y              ; Set back
    add cl, COVAR               ; Add 1/0 to COVAR
ERASE_RIGHT:    push ax         ;/////////// RIGHT /////////
                inc ax          ; Make 1-144
                call MODSIZE
                cmp ax, 0       ; See if ax was multiple of 12
                pop ax
                je ERASE_BELOW  ;   then no right there
                inc ax          ; Set right field
                call ERASE      ; Check it
                dec ax          ; Set back
                add cl, COVAR   ; Add 1/0 to COVAR
ERASE_BELOW:    cmp ax, 131     ;/////////// BELOW /////////
                ja ERASE_LEFT   ; See if there is a field below
                add ax, next_y  ; Set to below
                call ERASE      ; Check it
                sub ax, next_y  ; Set back
                add cl, COVAR   ; Add 1/0 to COVAR
ERASE_LEFT: push ax             ;/////////// LEFT /////////
            call MODSIZE
            cmp ax, 0           ; See if mod is 0
```

```
            pop ax
            je ERASE_UNSETEND   ; If 0 then no left field then end
            dec ax              ; Set left
            call ERASE          ; Check it
            inc ax              ; Set back
            add cl, COVAR       ; Add 1/0 to COVAR
            jmp ERASE_UNSETEND
ERASE_FREEEND:  mov COVAR, FREE ; Free to play
                jmp ERASE_END
ERASE_UNSETEND: shl cl, 2       ; cl/4; if cl == 4 then cl = 1, else cl = 0
                mov COVAR, cl   ; 1-> unfree, 0-> free
                call ERASEME    ; Remove stone
                inc COUNT       ; Adding one stone to erased
ERASE_END:  pop cx
            pop bx
            pop ax
            ret
ERASE endp
ERASEME proc        ;-------------------------------------------------------
    push ax         ; Erase on stone from board and graphical board
    push bx         ;> ax - BYTE-XY-coordinate
    push dx         ;-------------------------------------------------------
    shl ax, 1                   ;=> Removing stone from board
    mov bx, OFFSET BOARD
    add bx, ax
    mov WORD PTR [bx], FREE
    shr ax, 1                   ;<=|
    mov dx, OFFSET BufferField  ;=> Removing stone from GUI
    call MODSIZE
    mov bx, YVAR
    shl ax, 4
    shl bx, 4
    call PLACESTONE             ;<=|
    pop dx
    pop bx
    pop ax
    ret
ERASEME endp
KIMOVE proc         ;-------------------------------------------------------
    push ax         ; Get some number and try to place a stone there
    push cx         ;-------------------------------------------------------
    push dx
    add ax, KISAVE
    add ax, TimeCounterBlack
    add ax, TimeCounterWhite
    add ax, TimeBlack
    add ax, TimeWhite
KIDIV:  cmp ax, 144             ;=> Make the number lower then 144
        jb KINEXT
        sub ax, 144
        inc KISAVE
        jmp KIDIV               ;<=|
KINEXT: xor dx, dx
KIX:    cmp ax, board_size      ;=> Create X and Y values out of that number
        jb KIY
        sub ax, board_size
        inc dx
        jmp KIX                 ;<=|
KIY:    shl ax, 5               ;=> Make that values fit a mouse click
        add ax, 32
        mov cx, ax
        shl dx, 4
        add dx, 16              ;<=|
        call MOUSEONHUD         ; Try to set that stone
        pop dx
        pop cx
```

```
        pop ax
        ret
KIMOVE endp
PLACESTONE proc         ;-----------------------------------------------------
    call HIDEMOUSE      ; Placing a stone on the board
    push bx             ;> ax X-coordinate
    push cx             ;> bx Y-coordinate
    push di             ;> dx Offset <Color>
    push si             ;-----------------------------------------------------
    push @DATA
    pop ds
    push video_seg
    pop es
    add bx, 4                   ;=> Setting the start-position on screen
    imul bx, win_width          ;   ( y * window_width + x)
    mov di, ax
    add di, bx
    add di, 4                   ;<=|
    mov si, dx                  ; Setting stone to place
    mov bx, field_size          ; Setting height to field_size
    cli
ROW:    mov cx, field_size      ; Setting width to field_size
COL:    movsb                           ; Moving ds:si to es:di
        loop COL                        ;   for complete field_size
        add di, win_width - field_size  ; Switching to next line
        add si, 256                     ; Adding 16*16 = 256 to si for next line
        dec bx                          ; one less row to go
        jnz ROW
        sti
        pop si
        pop di
        pop cx
        pop bx
        call SHOWMOUSE          ; Showing mouse cursor
        ret
PLACESTONE endp
DRAWBOARD proc                  ;-----------------------------------------------
      xor bx, bx                ; Draws the board on the screen. The size will
      mov dx, OFFSET BufferField; be board_size*board_size.
      mov ch, board_size        ;-----------------------------------------------
ROW1: mov cl, board_size
      xor ax, ax
COL1: call PLACESTONE             ; Place a field-stone
      add ax, field_size          ; Move to next X stone
      dec cl
      cmp cl, 0
      jne COL1
      add bx, field_size          ; Move to next Y stone
      dec ch
      cmp ch, 0
      jne ROW1
      ret
DRAWBOARD endp

;== END ==
end GoASM
```