****

Based on an article from the February 1983 issue of 'Your Computer'

"The program is copyright. You can produce a copy of the program from the listings for your own use, but you should not copy the listings or parts thereof and offer for sale"

****

****

it was modified to get more free space for the improvements - GZS

****

```
;#define QPawn                  ; choose this option for "Queen's Pawn Game"
#define ZX80upg                 ; choose this option for the upgraded ZX80

;;KEYBOARD      .equ $02bb       ;
DECODE          .equ $07bd       ;


        ; system variables


        ; ERR_NR        .equ $4000      ; 16384
        ; FLAGS         .equ $4001      ; 16385
        ; ERR_SP        .equ $4002      ; 16386
        ; RAMTOP        .equ $4004      ; 16388
        ; MODE          .equ $4006      ; 16390

;;origPos
        ; PPC           .equ $4007      ; 16391


        .org $4009              ; 16393         original values

        .db $00                 ; VERSN         ($00)
        .dw $0004               ; E_PPC         ($0004)
        .dw DFile               ; D_FILE        ($4332)
        .dw $0000               ; DF_CC         ($43A5)
        .dw Variables           ; VARS          ($43AF)
        .dw $0000               ; DEST          ($437F)
        .dw eof_Vars            ; E_LINE        ($43B6)
        .dw eof_Vars+4          ; CH_ADD        ($43C3)
        .dw $0000               ; X_PTR         ($0000)
        .dw $0000               ; STKBOT        ($43C4)
        .dw $0000               ; STKEND        ($43C4)
        .db $00                 ; BREG          ($48)
        .dw $0000               ; MEM           ($405D)
;;      .db $00                 ; UNUSED1       ($00)
;;      .db $00                 ; DF_SZ         ($02)
;;      .dw $0000               ; S_TOP         ($0000)
;
;       ------------------------ UNUSED1, DF_SZ, S_TOP (4 bytes)
TKP
        ld (hl),$0F             ; cursor ("?") character
        jr old_TKP              ; the original subroutine
;
;       ---------------------------------------------------------------
;
LAST_K  .dw $0000               ; LAST_K        ($FDBF)
        .db $00                 ; DEBOUN        ($FF)
        .db $00                 ; MARGIN        ($37)
;;      .dw Line2               ; NXTLIN        ($407D)  -  autostart
        .dw Line3               ; NXTLIN        ($407D)  -  autostart
        .dw $0000               ; OLDPPC        ($FFFE)
        .db $0000               ; FLAGX         ($02)
        .dw $0000               ; STRLEN        ($FA7D)
        .dw $0000               ; T_ADDR        ($0C87)
        .dw $0000               ; SEED          ($E595)

;;      .dw $0000               ; FRAMES        ($DDAD)
        .dw $8001               ; FRAMES        ($DDAD)

;;      .dw $0000               ; COORDS        ($0000)
;;      .db $00                 ; PR_CC         ($BC)
;;      .dw $0000               ; S_POSN        ($0A1C)
```

```
;
;       ------------------------- COORDS, PR_CC, S_POSN (5 bytes)
#ifdef ZX80upg
GameOver
        ld bc,$0181             ; an unavailable graphic symbol
        jr old_TKP              ;
#else
KYBD
        ld bc,$0826             ; possible character codes ("A".."H")
        jr KYBD0
#endif
;       ------------------------------------------------------------------
;
;;      .db $00                 ; CDFLAG        ($00)
        .db $40                 ; CDFLAG        starts in SLOW mode
;
;       ------------------------------------------------------------------
;
oldScore                        ; PRBUFF
        .db $00,$00,$00,$00,$00
newScore
        .db $00,$00,$00,$00,$00
newList
        .db $00,$00,$00,$00,$00,$00
        .db $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$76

                                ; MEMBOT

        .db $00,$00,$00,$00,$00,$00
oldList
        .db $00,$00,$00,$00,$00,$00,$00,$00,$00
        .db $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00

        .dw $0000               ; UNUNSED2      ($00,$00)

;;RAttack       .equ $+3

;
; ===============================================
; LINE 1 REM
; ===============================================
;
Line1
        .db $00,$01             ; Line 1
        .dw Line2-L1Text        ; Line 1 length
L1Text
        .db $EA                 ; REM
```

```
;
;       ---------------------------------------------------------------
;
; The subroutine TKP just scans the keyboard waiting for an appropriate key to
; be depressed. The alpha-numeric entry is then translated to a board address.
;
;;TKP
old_TKP
        push hl                 ; save input line position (INPUT)
TKP0
        push bc                 ; save possible character codes
TKP1
;;      call KEYBOARD           ; ROM: $02bb

;;      ld b,h                  ; the status of the keyboard
;;      ld c,l                  ;

#ifdef ZX80upg

        bit 6,(iy+$3b)          ; is it an upgraded ZX80? (CDFLAG)
        call z,$0229            ; -> FAST mode displaying (PAUSE)
#endif
        ld bc,(LAST_K)          ; the status of the keyboard

;;      ld d,l                  ;
;;      inc d                   ; if non of the keys are pressed,

        inc c                   ; if non of the keys are pressed,
        jr z,TKP1               ; then read again

        dec c                   ; set back the keyboard status
        call DECODE             ; ROM: $07bd

        ld a,(hl)               ; A= the decoded character
        pop bc                  ; set back the possible character
        push bc                 ; codes for the test
TKP2
        cp c                    ; if it corresponds,
        jr z,TKP3               ; return

        inc c                   ; next value
        djnz TKP2               ; if it's not among the 8 possible

        pop bc                  ; value,
        jr TKP0                 ; then start from the beginning
TKP3
        pop bc                  ; possible character codes
        pop hl                  ; input position (INPUT)
        ld (hl),a               ; display the character
        ret
```

```
;
;       ---------------------------------------------------------------
;
; KYBD is a routine which sets up machine control of the keyboard such that only the eight
; key codes from code 29 and the eight key codes from code 38 are acceptable entries.
; Any other key depression is ignored.
;
#ifdef ZX80upg
KYBD
;;      ld bc,$081d             ; possible character codes ("1".."8")
        ld bc,$0826             ; possible character codes ("A".."H")
#endif
KYBD0
        call TKP                ; reading the keyboard

;;      dec hl                  ; next input position
;;      ld c,$26                ; possible character codes ("A".."H")
        inc hl                  ; next input position
        ld c,$1d                ; possible character codes ("1".."8")
        call TKP                ; reading the keyboard

;;      inc hl                  ; previous input position's (rows)
        ld a,(hl)               ; character ("1".."8")
        sub $1c                 ; number from character ("1" -> 1; "8" -> 8)

        ld b,a                  ; Y (rows)
;;      ld c,$0b                ; 11
        xor a                   ;
KYBD1
;;      add a,c                 ; 11*Y
        add a,$0b               ; 11*Y
        djnz KYBD1              ;

        add a,CountA1 & 255     ; ($61 originally)
        dec hl                  ; next input position (column)
        sub (hl)                ; A= the screen position's lower byte
```

```
;
;       ---------------------------------------------------------------
;
; STR: this routine takes the board address and determines whether the contents are:
; different from the current mover colour (0); empty (1); the board surround (2)
; or the same colour as the current mover (3).
;
STR
        ld c,a                  ; C= the screen position's lower byte
        ld l,c                  ; L= the screen position's lower byte
        ld h,BoardH1/256        ; HL= the whole screen address
;
;       ---------------------------------------------------------------
STR2
        ld a,l                  ; A= the screen position's lower byte
        ld b,$02                ; B=2 indicates the margin of the board

        cp BoardH1 & 255        ; if the address is smaller than the board's
        jr c,STR4               ; first square, see the next possible move

        cp BrdOut & 255         ; if the address is bigger than the board's
        jr nc,STR4              ; last square, then see the next possible move

        ld a,(hl)               ; the board position's content
;;      ld b,$01                ; B=1 indicates the empty square
        and $7f                 ; deleting colour bit
;;      cp $00                  ; (this is unnecessary)
;;      jr z,STR1               ; empty square?
        jr z,STR3               ; empty square?

;;      inc b                   ; B=2 indicates the margin of the board
        cp $76                  ; right margin (N/L)?
;;      jr z,STR1               ;
        jr z,STR4               ;

;;      cp whB                  ; left margin?
        cp $26                  ; left margin?
;;      jr c,STR1               ; (character code smaller than "B")
        jr c,STR4               ; (character code smaller than "A")

        ld a,(hl)               ; colour bit back again
        inc b                   ; B=3 indicates own piece (pawn, bishop...etc.)

;;      ld l,MoverCol & 255     ; (was $37)
;;      add a,(hl)              ; the current mover's colour ($80 or $00)
;;      bit 7,a                 ; if it corresponds, the 7th bit
;;      jr z,STR1               ; is zero

MoverCol .equ $+1

        add a,$80               ; the current mover's colour ($80 or $00)
        rla                     ; if it corresponds, the 7th bit
        jr nc,STR4              ; is zero

;;      ld b,$00                ; otherwise B=0 indicates the opponent
        xor a                   ; set Z-flag
        ret                     ; A=0 indicates the opponent
STR3
        dec b                   ; B=1 indicates the empty square
STR4;;STR1
        ld a,b                  ; A=B (1..3) position's properties
;;      ld l,c                  ; HL= the whole screen address
        and a                   ; reset Z-flag
        ret                     ;
```

```
;
;         ---------------------------------- TABLES:
;
theKing
aRook
        .db $01,$0B,$FF,$F5
        ;     1, 11, -1,-11
blkPawn .equ $+1
aBishop
        .db $F6,$F4,$0C,$0A
        ;   -10,-12, 12, 10
akNight
        .db $0D,$F3,$15,$EB,$17,$E9,$F7,$09
        ;    13,-13, 21,-21, 23,-23, -9,  9
whtPawn .equ $+2
        .db $0B,$0A,$0C
        ;    11, 10, 12
;
;       ---------------------------------- PIECES:
;
whK     .equ $30                ; white King
whQ     .equ $36                ; white Queen
whR     .equ $37                ; white Rook
whB     .equ $27                ; white Bishop
whN     .equ $33                ; white kNight
whP     .equ $35                ; white Pawn
;
;       ---------------------------------
;
blK     .equ whK+$80            ; black King
blQ     .equ whQ+$80            ; black Queen
blR     .equ whR+$80            ; black Rook
blB     .equ whB+$80            ; black Bishop
blN     .equ whN+$80            ; black kNight
blP     .equ whP+$80            ; black Pawn
;
;       ---------------------------------
;
#ifdef QPawn    ; "Queen's Pawn Game"

QP1     .equ $80                ;
QP2     .equ whP                ;
KP1     .equ whP                ;
KP2     .equ $80                ;

#else           ; "King's Pawn Game"

QP1     .equ whP                ;
QP2     .equ $00                ;
KP1     .equ $00                ;
KP2     .equ whP                ;

#endif
;
;       ---------------------------------
;
whPieces
        ;   $36,$37,$27,$33,$35,$30
        ;    54, 55, 39, 51, 53, 48
        .db whQ,whR,whB,whN,whP,whK
```

```
;
;       ---------------------------------------------------------------
;
; PSC: gives a score to a chess piece - Q(5), R(4), B(3), N(2), P(1), K(0)
;
PSC
        and $7f                 ; deleting the colour of the piece
        ld hl,whPieces          ; the character codes of the pieces
;;      ld b,$05                ; the highest value
        ld bc,$0006             ; the highest value +1
;;PSC1
;;      cp (hl)                 ; if we found it
;;      ret z                   ; then back

;;      inc hl                  ; pointer to the next character code
;;      djnz PSC1               ; lowering the value
        cpir                    ; search

;;      ld a,b                  ; A=B=0 if there's no match
        ld a,c                  ; Z=0 if there's no match
        ret                     ; A=C
;
;       ---------------------------------------------------------------
;
; AddList: adds to the current legal move list another entry on the end.
;
ALIST
        ld hl,newList           ; the starting address of the list of moves
        inc (hl)                ; increase the counter
        ld a,(hl)               ; read the counter
        add a,l                 ; calculate the new address
        ld l,a                  ; set pointer
        ld (hl),c               ; and store the new item
        ret                     ;
;
;       ---------------------------------------------------------------
;
; TestList: tests to see if there are any moves in the move list.
;
TL
        ld hl,newList           ; starting address of the list of moves
        dec (hl)                ; lower the counter
        ld a,(hl)               ; read the counter
        inc a                   ; if the original value is
        ret z                   ; zero, then back

        add a,l                 ; otherwise set the pointer
        ld l,a                  ; to the last item
        ld a,(hl)               ; return with the value of the last item
        ret                     ;
```

```
;
;       ----------------------------------------------------------------
;
; PIECE: this sets up pointers to possible move tables and number of steps and
; directions.
;
PIECE
        ld e,l                  ; save the current position of the piece

        xor a                   ; clear the list
        ld (newList),a          ; of moves
        ;
;;      ld a,(hl)               ; clears the colour of the
;;      and $7f                 ; piece under consideration
;;      cp whP                  ; is it a "P"awn?
;;      jr z,PAWN               ; creating list of moves
        ld a,(hl)               ; content of the current position
        ld d,a                  ; store it

        call PSC                ; is a piece? Q(5),R(4),B(3),N(2),P(1),K(0)
        ret nz                  ; no, return


;;      ld c,$01                ; max. number of moves
;;      ld b,$08                ; and directions
        ld bc,$0801             ; max. nr. of directions and moves

;;      ld hl,akNight           ; pointer to the table of moves
;;      cp whN                  ; is it a k"N"ight?
;;      jr z,MOVE               ; creating list of moves
;;      ld l,theKing & 255      ; pointer to the table of moves
;;      cp whK                  ; the "K"ing?
        ld hl,theKing           ; pointer to the table of moves
        and a                   ; the "K"ing?
        jr z,MOVE               ; creating list of moves

;;      ld c,b                  ; max. 8 moves / 8 directions
;;      cp whQ                  ; the "Q"ueen?
;;      jr z,MOVE               ; creating list of moves
;;      ld b,$04                ; max. 8 moves / 4 directions
;;      cp whR                  ; is it a "R"ook?
;;      jr z,MOVE               ; creating list of moves
;;      ld l,aBishop & 255      ; pointer to the table of moves
;;      cp whB                  ; is it a "B"ishop?
;;      ret nz                  ; if not, then return

        ld l,blkPawn & 255      ; pointer to the table of moves
        dec a                   ; is it a "P"awn?
        jr z,PAWN               ; creating list of moves

        ld l,akNight & 255      ; pointer to the table of moves
        dec a                   ; is it a k"N"ight?
        jr z,MOVE               ; creating list of moves

        ld bc,$0408             ; max. nr. of directions and moves
        ld l,aBishop & 255      ; pointer to the table of moves
        dec a                   ; is it a "B"ishop?
        jr z,MOVE               ; creating list of moves

        ld l,aRook & 255        ; pointer to the table of moves
        dec a                   ; is it a "R"ook?
        jr z,MOVE               ; creating list of moves

        ld b,c                  ; 8 directions and 8 moves (the "Q"ueen)
```

```
;
;       ---------------------------------------------------------------
;
; MOVE: produces a list of all legal moves available to the piece under
; consideration.
;
MOVE
        ld a,e                  ; the current position of the piece
MOVE1
        add a,(hl)              ; new position based on the table of moves
        push af                 ; save position
        push hl                 ; save table of movements' pointer
        push bc                 ; save max. number of moves and directions

;;      cp BoardH1 & 255        ; if the address is smaller than the board's
;;      jr c,MOVE2              ; first square, see the next possible move
;;      cp BrdOut & 255         ; if the address is bigger than the board's
;;      jr nc,MOVE2             ; last square, then see the next possible move

        call STR                ; analyse content

        cp $02                  ; if it's not empty (1) or not an opponent (0)
        jr nc,MOVE2             ; then see the next possible move

        push af                 ; otherwise save position property
        call ALIST              ; and add to the list of moves

        pop af                  ; set back the position property
;;      cp $00                  ; if it's an opponent, then we can't go in that direction
        and a                   ; if it's an opponent, then we can't go in that direction
        jr z,MOVE2              ; let's see the next possible move

        pop bc                  ; max. number of moves and the direction
        pop hl                  ; the pointer of table of moves
        ld a,c                  ; if this piece (in this direction)
;;      cp $01                  ; can only move once (one square)
        dec a                   ; can only move once (one square)
        jr z,MOVE3              ; let's see the next possible direction

        pop af                  ; otherwise set back the address of the position
        jr MOVE1                ; then calculate the next move
MOVE2
        pop bc                  ; max. number of moves and the direction
        pop hl                  ; the pointer of table of moves
MOVE3
        pop af                  ; set back the address of the position
        inc hl                  ; the next item of the table of moves
        djnz MOVE               ; if it wasn't the last direction,
                                ; then repeat from the beginning
        ret                     ;
```

```
;
;       ---------------------------------------------------------------------
;
; PAWN: produces a list of all possible legal moves including initial double moves.
;
PAWN
;;      ld a,(hl)               ; the pawn's
;;      and $80                 ; colour
;;      ld hl,blkPawn           ; (black pawn's table of moves)
        bit 7,d                 ; the pawn's colour
        jr nz,PAWN1             ; black?

        ld l,whtPawn & 255      ; no - white pawn's table of moves
PAWN1
        ld d,$03                ; can move to 3 different directions
PAWN2
        ld a,e                  ; the pawn's position (lower byte)
PAWN3
        add a,(hl)              ; new position based on the table of moves
        push hl                 ; save pointer of the table of moves
        push af                 ; save position

;;      cp BoardH1 & 255        ; if the address is smaller than the board's
;;      jr c,PAWN4              ; first square, then see the next possible move
;;      cp BrdOut & 255         ; if the address is bigger than the board's
;;      jr nc,PAWN4             ; last square, then see the next possible move

        call STR                ; analyse content

;;      cp $00                  ; if it's an opponent,
        jr z,PAWN5              ; then add to the list of moves

;;      cp $01                  ; if it's not empty,
        dec a                   ; if it's not empty,
        jr nz,PAWN4             ; then see the next direction

        ld a,d                  ; if it's not the
;;      cp $01                  ; last direction (forward)
        dec a                   ; last direction (forward)
        jr nz,PAWN4             ; see the next direction

        call ALIST              ; otherwise add to the list of moves

        ld a,e                  ; the pawn's position (lower byte)

        cp wPwnMax & 255        ; 2nd row? (white pawn's first move)
        jr c,PAWN6              ; if yes, then let's see the next move

        cp bPwnMax & 255        ; 7th row? (black pawn's first move)
        jr nc,PAWN6             ; if yes, then let's see the next move
PAWN4
        pop af                  ; set back the address of the position
        pop hl                  ; the pointer of the table of moves
        dec hl                  ; set to the next direction
        dec d                   ; check the direction
        jr nz,PAWN2             ; if it wasn't the last, then again

        ret
```

```
;
;       ----------------------------------------------------------------
PAWN5
        ld a,d                  ; if the direction is
;;      cp $01                  ; not the last (forward)
        dec a                   ; not the last (forward)
        call nz,ALIST           ; then add to the list of moves

        jr PAWN4                ; test the next direction
;
;       ----------------------------------------------------------------
PAWN6
        pop af                  ; set back the address of the position
        pop hl                  ; the pointer of the table of moves
        ld e,a                  ; keep the positon's address
        jr PAWN3                ; test the new position




;
;       ----------------------------------------------------------------
origPos .equ $+1
PMOVE
;;      ld hl,(origPos)         ; original position
        ld hl,BoardH1           ; original position
        ld a,(de)               ; the content of the new position
        ld c,a                  ; to the register "C"
        ld a,(hl)               ; the content of the original position
        ld (hl),$00             ; (deleting temporarily)
        ld (de),a               ; to the new position
        ld b,a                  ; and to register "B"
;;      ret                     ;
        exx                     ; save the main register set
        and a                   ; CY=0
        call SHIFT              ; save the list of moves




;
;       ----------------------------------------------------------------
;
; CHK locates current mover's Kings and stores the position in the attack
; register.
;
CHK
        ld a,(MoverCol)         ; current mover's colour: $80(black), $00(white)
        add a,whK               ; code for the "K"ing
        ld hl,BoardH1           ; the board's starting address
        ld b,a                  ; setting the counter (bc) for search
        cpir                    ; search
;;      dec hl                  ;
;;      ld (RAttack),hl         ; save the result
        ld a,l                  ;
        dec a                   ; the result
```

```
;
;       ---------------------------------------------------------------
;
; SQuare ATtack: determines whether the opposition can attack the square in the
; attack register.
;
SQ_AT
        ld (RAttack),a          ; save the result
        ld b,BrdSize & 255      ; check upon the number of screenposition (max. 86)
        ld hl,BoardH1-1         ; the board's starting address (-1)
SQ_AT1
        inc hl                  ; the next position
        push hl                 ; save it
        push bc                 ; save the counter
;;      ld e,l                  ; store the lower byte of the address
        call STR2               ; analyse the content

;;      cp $00                  ; opponent?
        jr nz,SQ_AT3            ; if not, then next

        ex de,hl                ; (save HL)
        call CHGMV              ; changing the colour of the mover for the check up
        ex de,hl                ; (restore HL)

;;      ld l,e                  ; the lower byte of the address
        call PIECE              ; creating the list of moves

        call CHGMV              ; set back the colour of the mover
SQ_AT2
        call TL                 ; test the list
        jr z,SQ_AT3             ; if empty, see the next position

;;      ld hl,(RAttack)         ; is the address in the attack register
;;      cp l                    ; on the list?

RAttack .equ $+1
                                ; is the address in the attack register
        cp BoardH1 & 255        ; on the list?
        jr nz,SQ_AT2            ; if not, see the next component on the list

        pop bc                  ; set back the counter
        pop hl                  ; in HL the address of the attacker
        scf                     ; CY=1 indicates that there's an attack
        ret                     ;
SQ_AT3
        pop bc                  ; set back the counter
        pop hl                  ; the screen position which was checked upon
        djnz SQ_AT1             ; if the last position is safe as well

        and a                   ; CY=0 indicates that there's no attack
        ret                     ;
```

```
;
;       ------------------------------------------------------------------
;
; INC determines whether a square is being attacked.
;
INC_
        ld a,l                  ; the lower byte of the address

        exx                     ; save the main register set

;;      ld (RAttack),a          ; store the address of the board
        call SQ_AT              ; is this position under attack?
                                ; (CY=1, if yes)
        exx                     ; set back the main register set

;;      ld a,c                  ;
        ld a,d                  ;
        ret                     ;
```

```
;
;       ---------------------------------------------------------------
;
; SCORE provides a move score based on the following:
; first, the "To" position results in taking of a piece.
; Second, the "From" position is attacked.
; Third, the "To" position is attacked.
; Fourth, "To" enables the computer to obtain a check
; and finally the "From" position is defended.
;
; The current move score is then compared with the previous best and if this is
; superior, the move is saved as the best so far.
;
SCORE
        push hl                 ; save the original position's address
        push bc                 ; save the content of the positions (original/new)
        push de                 ; save the new position's address

        push hl                 ; save the original position's address
        push bc                 ; save the content of the positions (original/new)

;;      ld d,l                  ; original position lower byte
;;      ld hl,oldScore+4        ; 16448 ($4040 in Printer buffer)
;;      call $0724              ; ROM:
                ;; LD (HL),B    ; PRBUFF+4 = content of the original position
                ;; DEC HL       ;
                ;; LD (HL),C    ; PRBUFF+3 = content of the new position
                ;; DEC HL       ;
                ;; LD (HL),E    ; PRBUFF+2 = address of the new position (lower byte)
                ;; DEC HL       ;
                ;; LD (HL),D    ; PRBUFF+1 = address of the original position (lower byte)
                ;; RET          ;

        ld h,b                  ; store the content of the original position and the
        ld (oldScore+3),hl      ; address of the original position (lower byte), and
        ld (oldScore+1),de      ; store the new position's address, as score parameters

        call PSC                ; get value of the piece: Q(5), R(4), B(3), N(2), P(1)

;;      ld a,b                  ; value of the piece: Q(5), R(4), B(3), N(2), P(1)
        add a,h                 ; + $40 (64; whPieces upper byte)
;;      ld c,a                  ; saved in C
        ld d,a                  ; saved in D

        pop af                  ; A= piece in the original position

        call PSC                ; value of the piece: Q(5), R(4), B(3), N(2), P(1)

        pop hl                  ; set back the original position
        call INC_               ; is this position under attack? (CY=1, if it is)

        jr nc,SCORE1            ;

;;      add a,b                 ; if it is, then change the value of the move
        add a,c                 ; if it is, then change the value of the move
SCORE1
;;      ld c,a                  ; and save it to a temporary repertory

        pop hl                  ; reading the address of the new position and
        pop de                  ; the original(D)/new(E) content
;;      ld e,(hl)               ; content of the new position
        ld (hl),d               ; changed to the content of the original position
        push hl                 ; save the address of the new position
        push de                 ; save the content of the positions (original/new)
```

```
        ld d,a                  ; and save it to a temporary repertory

        call INC_               ; is the new position under attack? (CY=1, if it is)

        jr nc,SCORE2            ;

;;      sub b                   ; if yes, then set back the previous value of the move
        sub c                   ; if yes, then set back the previous value of the move
SCORE2
        push af                 ; save the calculated value of the move

        call CHGMV              ; changing the colour of the mover

        call CHK                ; check?

        pop bc                  ; (calculated value to register "B")
        jr nc,SCORE3            ;

        inc b                   ; if yes, then increase the value of the move
        inc b                   ; by 2
SCORE3
        pop de                  ; reading the original(D)/new(E) content
        pop hl                  ; set back the content of
        ld (hl),e               ; the new position

        pop hl                  ; change the colour of the
        call CHG                ; original position

        call INC_               ; is this position under attack? (CY=1, if it is)

        jr nc,SCORE4            ;
        dec b                   ; if it is, then lower the value of the move
SCORE4
        call CHG                ; set back the colour of the position

        call CHGMV              ; set back the colour of the mover

        ld a,b                  ; save the calculated value of the
;;      ld hl,oldScore          ;
;;      ld (hl),a               ; move
;;      ex de,hl                ;
        ld de,oldScore          ;
        ld (de),a               ; move
        ld hl,newScore          ; if the previously calculated
        cp (hl)                 ; value is smaller,
        ret c                   ;

        ld bc,$0005             ; then copy the data of the new position
        jr SHIFT1               ; as a replacement for the old data
```

```
;
;       ---------------------------------------------------------------
;
; Shift moves the current move list to a safe position whilst Check is being
; evaluated, and then recovers the move list on completion. It is also used to
; shift the best move so far up into the move list.
;
SHIFT
        ld hl,oldList           ; address of the saved list
        ld de,newList           ; address of the list of moves
        ld bc,$001c             ; max. 28 bytes
        jr c,SHIFT2             ; if CY=1, then set back
SHIFT1
        ex de,hl                ; if CY=0, then save
SHIFT2
        ldir                    ; moving the data
        ret                     ;
```

```
;
;       ---------------------------------------------------------------
;
; MPSCAN: scans the board for computer pieces and, using move and score,
; determines all legal moves and saves the best.
;
MPSCAN
        xor a                   ; initialise (to zero) the
        ld (newScore),a         ; calculated value of the move

        ld b,BrdSize & 255      ; we look through max. 86 positions
        ld hl,BoardH1-1         ; starting with the board's first component
MPSCAN1
        inc hl                  ; pointer to the next position on the board
        push hl                 ; save the position on the board
        push bc                 ; save the counter
;;      ld e,l                  ; position to a temporary storage
        call STR2               ; analyse content

        cp $03                  ; is it our own piece?
        jr nz,MPSCAN5           ; if not, then check the next position

;;      ld l,e                  ; set back the position on the board
        ld (origPos),hl         ; and save it
        call PIECE              ; creating list of moves
MPSCAN2
        call TL                 ; a component of the list
        jr z,MPSCAN5            ; if empty, then check the next position

        ld e,a                  ; a possible new position for the piece
        ld d,BoardH1/256        ;
        call PMOVE              ; move the piece

;;      exx                     ; save the main register set

;;      and a                   ; CY=0
;;      call SHIFT              ; save the list of moves
;;      call CHK                ; check? (is the king under attack?)
;;MPSCAN3
        exx                     ; set back the main register set

        ld (hl),b               ; set back the content of the original position
        ld a,c                  ; and
        ld (de),a               ; the content of the new position
;;      jr c,MPSCAN4            ; if the king is under attack, then delete move
;;      call SCORE              ; otherwise choose the best possible move
        call nc,SCORE           ; choose the best possible move
;;MPSCAN4
        scf                     ; CY=1
        call SHIFT              ; set back of the list of moves
        jr MPSCAN2              ; the next component of the list of moves
MPSCAN5
        pop bc                  ; the current value of the counter and
        pop hl                  ; the position on the board
        djnz MPSCAN1            ; next position

        ld a,(newScore)         ; this was the last position
;;      cp $00                  ; checking the result
        and a                   ; checking the result
```

```
#ifdef ZX80upg

        jp z,GameOver           ; if the value of the move is zero, then the game ends
#else
MPSCAN6
        jr z,MPSCAN6            ; if the value of the move is zero, then the game ends
#endif

;;      ld hl,newScore+4        ; otherwise
;;      ld a,(hl)               ; read the original content
;;      dec hl                  ;
;;      dec hl                  ;
;;      ld e,(hl)               ; the address of the new position
;;      ld d,Board/256  ;
;;      ld (de),a               ; original content to the new position (move)
;;      dec hl                  ;
;;      ld l,(hl)               ; the address of the
;;      ld h,d                  ; original position

        ld de,(newScore+3)      ; original content(D) / position(E)
        ld hl,(newScore+1)      ; the address of the new position
        ld (hl),d               ; original content to the new position (move)
        ld l,e                  ; the address of the original position
;
;       ------------------------------------------------------------------
CHGSQ
        bit 0,l                 ; the even-numbered

#ifdef ZX80upg

        ld (hl),$80             ; squares are black
#else
        ld (hl),$00             ; squares are white
#endif

;;      jr z,CHGSQ1             ; the odd ones are
;;      ld (hl),$00             ; white

        call nz,CHG             ; the odd ones are inverse
;
;;      ---------------------------------- (two unnecessary rows)
;;CHGSQ1
;;      call CHGMV              ; changing the colour of the mover
;;      ret                     ;
;
;       ------------------------------------------------------------------
CHGMV
        ld hl,MoverCol          ; the colour of the mover
;
;       ------------------------------------------------------------------
CHG
        ld a,(hl)               ; changing the
        add a,$80               ; colour of a piece
        ld (hl),a               ; or position
        ret                     ;
```

```
;
;       ---------------------------------------------------------------
;
; DRIVER:
; Main control logic, uses all the other subroutines to provide program control.
;
DRIVER4
        ld (hl),b               ; move the piece to the original position
        ld a,c                  ; set back the original value
        ld (de),a               ; of the new position
;
; ***************************************************************************
;       MAIN PROGRAM ENTRY (RAND USR X)
; -------------------------------------
;
DRIVER
;;      ld b,$05                ; the INPUT line consists of 5
;;      ld a,$08                ; "chessboard" character
;;      ld hl,INPUT-1           ; starting address (-1)
        ld bc,$0416             ; the INPUT line consists of 4 "-" character
        ld hl,INPUT+4           ; end of the INPUT LINE
DRIVER1
;;      inc hl                  ;
;;      ld (hl),a               ; fill the INPUT line
        ld (hl),c               ; fill the INPUT line
        dec hl                  ;
        djnz DRIVER1            ;

        call KYBD               ; get and check the "from" position

        cp $03                  ; is it our own piece?
        jr nz,DRIVER            ; if not, then from the beginning

        ld (origPos),hl         ; save the "from" position
;;      ld e,l                  ; lower byte to E
        call PIECE              ; creating the list of moves

;;      ld hl,INPUT+1           ;
        ld hl,INPUT+3           ;
        call KYBD               ; get and check the target position

        cp $02                  ; if not empty, or
        ex de,hl                ; not one of the pieces of the opponent,
        jr nc,DRIVER            ; then from the beginning
DRIVER2
        call TL                 ; testing the list
        jr z,DRIVER             ; if empty, then from the beginning

        cp c                    ; is it on the list?
        jr nz,DRIVER2           ; if not, see the next item of the list

        call PMOVE              ; move piece to the new position

;;      exx                     ; save the main register set
;;      call CHK                ; check?
        exx                     ; set back the main register set

        jr c,DRIVER4            ; cancel the move, if the king is under attack

        call CHGSQ              ; setting of the colours of the old position and
                                ; changing the colour of the mover
        call MPSCAN             ; ZEDDY moves
;;DRIVER3
        jr DRIVER               ; USER moves
```

```
;;DRIVER4
;;      ld (hl),b               ; move the piece to the original position
;;      ld a,c                  ; set back the original value of the new position
;;      ld (de),a               ;
;;      jr DRIVER3              ; get the new move
;
;       ------------------------------------------------------------------
;
        .db $76                 ; line 1 end (halt)
;
; ================================================
; LINE 2 SLOW   - eliminated, using CDFLAG=$40
; ================================================
;
Line2
;;      .db $00,$02             ; line number
;;      .dw Line3-L2Text        ; line length
;;L2Text
;;      .db $E4                 ; SLOW
;;      .db $76                 ; line 2 end (halt)
;
; ================================================
; LINE 3 RAND USR X
; ================================================
;
Line3
        .db $00,$03             ; line number
        .dw DFile-L3Text        ; line length
L3Text
        .db $F9,$D4,$3D         ; RAND USR X
        .db $76                 ; line 3 end (halt)
```

```
;
; ================================================
; D-FILE ($4332)
; ================================================
;
DFile
        .db $76,$76,$76,$76,$76,$76                             ; rows 1-5
        .db $76,$76,$76,$76,$76,$76                             ; rows 6-11
;;MoverCol
;;      .db $80,$08
        .db $A9,$B7,$AD,$76,$76                                 ; row 12-13

;;Board .equ $+1
CountA1 .equ $+9-11+$26
BoardH1 .equ $+2

        .db $1D,$08,whR,whN,whB,whK,whQ,whB,whN,whR,$76,        ; row 14
        .db $1E,$08,whP,whP,whP,KP1,QP1,whP,whP,whP,$76,        ; row 15

wPwnMax .equ $-1

        .db $1F,$08,$00,$80,$00,KP2,QP2,$80,$00,$80,$76,        ; row 16
        .db $20,$08,$80,$00,$80,$00,$80,$00,$80,$00,$76,        ; row 17
        .db $21,$08,$00,$80,$00,$80,$00,$80,$00,$80,$76,        ; row 18
        .db $22,$08,$80,$00,$80,$00,$80,$00,$80,$00,$76,        ; row 19

bPwnMax .equ $-1

        .db $23,$08,blP,blP,blP,blP,blP,blP,blP,blP,$76,        ; row 20
        .db $24,$08,blR,blN,blB,blK,blQ,blB,blN,blR,$76,        ; row 21

BrdSize .equ $-BoardH1
BrdOut  .equ $-1

        .db $08,$08,$2D,$2C,$2B,$2A,$29,$28,$27,$26,$76,        ; row 22
        .db $76                                                 ; row 23
INPUT
        .db $16,$16,$16,$16,$16,$76                             ; row 24
;;      .db $76,$76,$76,$76,$76,$76,$76,$76,$76
;
; ================================================
; VARIABLES
; ================================================
;
dblDRV  .equ (2*DRIVER)& $7FFF
Variables
        .db $7D,$8F             ; ($7D,$8F,$04,$7E,$00,$00)
        .db dblDRV/256          ;
        .db dblDRV&255          ;
        .db $00,$00             ; X=DRIVER

        .db $80
eof_Vars
.end
```