Home * Engines * CPW-Engine * **Board (0x88)**

```
#include "stdafx.h"
#include "0x88_math.h"
#include "transposition.h"

sboard b;


void clearBoard() {

    //reset b struct

    for (int i=0;i<128;i++) {
        b.pieces[i] = PIECE_EMPTY;
        b.color[i] = COLOR_EMPTY;
    }
    b.castle = 0;
    b.ep     = -1;
    b.ply    = 0;
    b.hash   = 0;
    b.phash  = 0;
    b.stm    = 0;
    b.rep_index=0;

    // reset perceived values

    b.PieceMaterial[WHITE] = 0;
    b.PieceMaterial[BLACK] = 0;
    b.PawnMaterial[WHITE]  = 0;
    b.PawnMaterial[BLACK]  = 0;
    b.PcsqMg[WHITE] = 0;
    b.PcsqMg[BLACK] = 0;
    b.PcsqEg[WHITE] = 0;
    b.PcsqEg[BLACK] = 0;


    // reset counters

    for (int i=0;i<6;i++) {
        b.PieceCount[ WHITE ] [ i ] = 0;
        b.PieceCount[ BLACK ] [ i ] = 0;
    }
```

```
}

/**************************************************************
* fillSq() and clearSq(), beside placing a piece on a given square  *
* or erasing it, must take care for all the incrementally  updated  *
* stuff: hash keys, piece counters, material and pcsq values, king  *
* location.                                                         *
**************************************************************/

void fillSq(U8 color, U8 piece, S8 sq) {

    // place a piece on the board
    b.pieces[sq] = piece;
    b.color[sq] = color;

    // update king location
    if (piece == KING)
        b.KingLoc[color] = sq;

    if ( piece == PAWN ) {
        // update pawn material
        b.PawnMaterial[color] += e.PIECE_VALUE[piece];


// update pawn hashkey - please note conversion to a 32-bit integer
        b.phash ^= zobrist.piecesquare[piece][color][sq];
    }
    else {
        // update piece material
        b.PieceMaterial[color] += e.PIECE_VALUE[piece];
    }

    // update piece counter
    b.PieceCount[color][piece]++;

    // update piece-square value
    b.PcsqMg[color] += e.mgPst[piece][color][sq];
    b.PcsqEg[color] += e.egPst[piece][color][sq];

    // update hash key
    b.hash ^= zobrist.piecesquare[piece][color][sq];
}


void clearSq(S8 sq) {
```

```
    // set intermediate variables, then do the same
    // as in fillSq(), only backwards

    U8 color = b.color[sq];
    U8 piece = b.pieces[sq];

    b.hash ^= zobrist.piecesquare[piece][color][sq];

    if ( piece == PAWN ) {
        b.PawnMaterial[color] -= e.PIECE_VALUE[piece];
        b.phash ^= zobrist.piecesquare[piece][color][sq];
    }
    else
        b.PieceMaterial[color] -= e.PIECE_VALUE[piece];

    b.PcsqMg[color] -= e.mgPst[piece][color][sq];
    b.PcsqEg[color] -= e.egPst[piece][color][sq];

    b.PieceCount[color][piece]--;

    b.pieces[sq] = PIECE_EMPTY;
    b.color[sq] = COLOR_EMPTY;
}


int board_loadFromFen(char * fen) {

    clearBoard();
    clearHistoryTable();

    char * f = fen;

    char col = 0;
    char row = 7;

    do {
        switch( f[0] ) {
        case 'K': fillSq(WHITE, KING, SET_SQ(row, col));   col++;
break;
        case 'Q': fillSq(WHITE, QUEEN, SET_SQ(row, col));  col++;
break;
        case 'R': fillSq(WHITE, ROOK, SET_SQ(row, col));   col++;
break;
        case 'B': fillSq(WHITE, BISHOP, SET_SQ(row, col)); col++;
break;
        case 'N': fillSq(WHITE, KNIGHT, SET_SQ(row, col)); col++;
```

```
break;
        case 'P': fillSq(WHITE, PAWN, SET_SQ(row, col));   col++;
break;
        case 'k': fillSq(BLACK, KING, SET_SQ(row, col));   col++;
break;
        case 'q': fillSq(BLACK, QUEEN, SET_SQ(row, col));  col++;
break;
        case 'r': fillSq(BLACK, ROOK, SET_SQ(row, col));   col++;
break;
        case 'b': fillSq(BLACK, BISHOP, SET_SQ(row, col)); col++;
break;
        case 'n': fillSq(BLACK, KNIGHT, SET_SQ(row, col)); col++;
break;
        case 'p': fillSq(BLACK, PAWN, SET_SQ(row, col));   col++;
break;
        case '/': row--; col=0; break;
        case '1': col+=1; break;
        case '2': col+=2; break;
        case '3': col+=3; break;
        case '4': col+=4; break;
        case '5': col+=5; break;
        case '6': col+=6; break;
        case '7': col+=7; break;
        case '8': col+=8; break;
        };

        f++;
    } while ( f[0] != ' ' );

    f++;

    if (f[0]=='w') {
        b.stm = WHITE;
    } else {
        b.stm = BLACK;
        b.hash ^= zobrist.color;
    }

    f+=2;

    do {
        switch( f[0] ) {
        case 'K': b.castle |= CASTLE_WK; break;
        case 'Q': b.castle |= CASTLE_WQ; break;
        case 'k': b.castle |= CASTLE_BK; break;
        case 'q': b.castle |= CASTLE_BQ; break;
```

```
        }

        f++;
    } while (f[0] != ' ' );

    b.hash ^= zobrist.castling[b.castle];

    f++;

    if (f[0] != '-') {
        b.ep = convert_a_0x88(f);
        b.hash ^= zobrist.ep[b.ep];
    }

    do { f++; } while (f[0] != ' ' );
    f++;

    sscanf(f, "%d", &b.ply);

    b.rep_index = 0;
    b.rep_stack[b.rep_index] = b.hash;

    return 0;
}


void board_display() {

    S8 sq;

    char parray[3][7] = { {'K','Q','R','B','N','P'},
                          {'k','q','r','b','n','p'},
                          { 0 , 0 , 0 , 0 , 0,  0, '.'}
                        };

    printf("   a b c d e f g h\n\n");

    for (S8 row=7; row>=0; row--) {

        printf("%d ", row+1);

        for (S8 col=0; col<8; col++) {
            sq = SET_SQ(row, col);
            printf(" %c",parray[b.color[sq]][b.pieces[sq]] );
        }
```

```
        printf("  %d\n", row+1);

    }

    printf("\n   a b c d e f g h\n\n");
}
```

What links here?

Page	Date Edited
0x88	Nov 28, 2016
CPW-Engine	Dec 31, 2014
CPW-Engine_board(0x88)	Dec 30, 2014
CPW-Engine_move(0x88)	Dec 30, 2014

Up one Level