[Home](#) * [Engines](#) * [CPW-Engine](#) * **Quiescence**

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

extern bool time_over;

int Quiesce( int alpha, int beta )  {

    if ( !time_over && !(sd.nodes & 0x3FF)  )
        time_over = time_stop();

    if (time_over) return 0;

    sd.nodes++;
    sd.q_nodes++;

    /* get a "stand pat" score */
    int val = eval( alpha, beta, 1);
    int stand_pat = val;

    /* check if stand-pat score causes a beta cutoff */
    if( val >= beta )
        return beta;

    /* check if stand-pat score may become a new alpha */
    if( alpha < val )
        alpha = val;


/*************************************************************
    *  We have taken into account rhe stand pat score, and it didn't l
et *
    *  us  come to a definite conclusion about the position. So we  mu
st *
    *  do a real search.
   *
    *************************************************************
***/

    smove movelist[256];
    U8 mcount = movegen_qs(movelist);
```

```
    for (U8 i = 0; i < mcount; i++) {

        movegen_sort( mcount,movelist, i );

        if ( movelist[i].piece_cap == KING ) return INF;


/*************************************************************
        *  Delta cutoff - a move guarentees the score well below alpha
,   *
        *  so  there's no point in searching it. This heuristic is  no
t   *
        *  used  in the endgame, because of the  insufficient  materia
l   *
        *  issues and special endgame evaluation heuristics.
    *
        *************************************************************
***/

        if ( ( stand_pat + e.PIECE_VALUE[ movelist[i].
piece_cap ] + 200 < alpha ) &&
             ( b.PieceMaterial[!b.stm] - e.PIECE_VALUE[
movelist[i].piece_cap] > e.ENDGAME_MAT ) &&
             ( !move_isprom(movelist[i]) ) )
            continue;


/*************************************************************
        *  badCapture() replaces a cutoff based on the  Static Exchang
e   *
        *  Evaluation,  marking  the place where it ought to  be  code
d.  *
        *  Nevertheless, it saves quite a few nodes.
    *
        *************************************************************
***/

        if ( badCapture( movelist[i] )
        &&    !move_canSimplify( movelist[i] )
        &&  !move_isprom( movelist[i] )
           )
           continue;


/*************************************************************
        *  Cutoffs  misfired, so the move in question can turn out wel
```

```
l. *
        *  Let us try it, then.
   *
        ************************************************************
***/

        move_make( movelist[i] );

        val = -Quiesce( -beta, -alpha );

        move_unmake( movelist[i] );

        if (time_over) return 0;

        if ( val > alpha ) {
            if( val >= beta )
                return beta;

            alpha = val;
        }
    }
    return alpha;
}

int badCapture(smove move) {

    /* captures by pawn do not lose material */
    if (move.piece_from == PAWN ) return 0;


/* Captures "lower takes higher" (as well as BxN) are good by definiti
on. */
    if ( e.PIECE_VALUE[move.piece_cap] >= e.PIECE_VALUE[move.
piece_from] - 50 )
        return 0;


/*****************************************************************
*****
    *   When the enemy piece is defended by a pawn, in the quiescence
search  *
    *   we  will  accept rook takes minor, but not minor takes pawn. (
 More   *
    *   exact  version  should accept B/N x P if (a) the pawn  is  the
  sole  *
    *   defender and (b) there is more than one attacker.
```

```
         *
    ************************************************************************
********/

    if ( pawnRecapture(b.color[move.from], move.to) &&
         e.PIECE_VALUE[move.piece_cap] + 200 - e.
PIECE_VALUE[move.piece_from] < 0 )
          return 1;

    /* if a capture is not processed, it cannot be considered bad */
    return 0;
}

int pawnRecapture( U8 capturers_color, char sq) {

    if (capturers_color == WHITE) {
       if ( ( IS_SQ(sq+NW) && isPiece(BLACK, PAWN, sq+NW) ) ||
            ( IS_SQ(sq+NE) && isPiece(BLACK, PAWN, sq+NE) ) )
         return 1;
    } else {
        if ( ( IS_SQ(sq+SW) && isPiece(WHITE, PAWN, sq+SW) ) ||
                ( IS_SQ(sq+SE) && isPiece(WHITE, PAWN, sq+SE) )  )
         return 1;
    }
    return 0;
}
```

What links here?

Page	Date Edited
CPW-Engine	Dec 31, 2014
CPW-Engine_quiescence	Dec 30, 2014
Quiescence Search	Aug 19, 2017

Up one Level