Home * Engines * CPW-Engine * **Chronos**

```
#include "stdafx.h"

enum etask {
    TASK_NOTHING,
    TASK_SEARCH,
    TASK_PONDER
} extern task;

stime chronos;

extern sSearchDriver sd;

extern bool time_over;

#define TIMEBUFFER 500
#define MOVESTOGO 24


#if defined(_MSC_VER) || defined(_WINDOWS_)
    #include <Windows.h>

    unsigned int gettime() {
        FILETIME ft;
        GetSystemTimeAsFileTime(&ft);
        return (unsigned int) ((((U64)ft.dwHighDateTime << 32) |
 ft.dwLowDateTime) / 10000);
    }
#else
    #include <sys/time.h>

    unsigned int gettime() {
        timeval t;
        gettimeofday(&t, 0);
        return t.tv_usec;
    }
#endif


void time_uci_go(char * command) {
```

```
    time_over = false;
    task = TASK_SEARCH;

    chronos.flags = 0;

    if (strstr(command, "infinite")) {chronos.flags |= FINFINITE;}
    if (strstr(command, "ponder")) {chronos.flags |=
 FINFINITE; task = TASK_PONDER;}

    //not implemented (do an infinite search instead)
    if (strstr(command, "searchmoves")) {chronos.flags |= FINFINITE;}

    if (strstr(command, "wtime")) {chronos.flags |= FTIME;
sscanf(strstr(command, "wtime"), "wtime %d", &chronos.time[WHITE]);}
    if (strstr(command, "btime")) {chronos.flags |= FTIME;
sscanf(strstr(command, "btime"), "%*s %d", &chronos.time[BLACK]);}
    if (strstr(command, "winc")) {chronos.flags |= FINC; sscanf(
strstr(command, "winc"), "%*s %d", &chronos.inc[WHITE]);}
    if (strstr(command, "binc")) {chronos.flags |= FINC; sscanf(
strstr(command, "binc"), "%*s %d", &chronos.inc[BLACK]);}
    if (strstr(command, "movestogo")) {chronos.flags |=
 FMOVESTOGO; sscanf(strstr(command, "movestogo"), "%*s %d", &
chronos.movestogo);}
    if (strstr(command, "depth")) {chronos.flags |= FDEPTH;
sscanf(strstr(command, "depth"), "%*s %d", &chronos.depth);}
    if (strstr(command, "nodes")) {chronos.flags |= FNODES;
sscanf(strstr(command, "nodes"), "%*s %d", &chronos.nodes);}

    //not implemented (do an infinite search instead)
    if (strstr(command, "mate")) {chronos.flags |= FMATE;
 chronos.flags |= FINFINITE; sscanf(strstr(command, "mate"),
"%*s %d", &chronos.mate);}

    if (strstr(command, "movetime")) {chronos.flags |=
 FMOVETIME; sscanf(strstr(command, "movetime"), "%*s %d", &
chronos.movetime);}

    if (chronos.flags == 0) chronos.flags |= FINFINITE;
}

int time_uci_ponderhit() {
    if (task != TASK_PONDER) return 0;

    /* switch from pondering mode to normal search */
    chronos.flags &= ~FINFINITE;
    sd.starttime = gettime();
```

```
    task = TASK_SEARCH;

    return 0;
}

void time_xboard_go() {

    time_over = false;
    task = TASK_SEARCH;

    if (!chronos.flags) chronos.flags = FINFINITE;
}

void time_nothing_go() {

    time_over = false;
    task = TASK_SEARCH;
}

void time_calc_movetime() {

    /* no movetime to be calculated in these search types */

    if (chronos.flags & (FINFINITE | FDEPTH | FNODES)) return;

    /* if the movetime is given */

    if (chronos.flags & FMOVETIME) {
        if (chronos.movetime > TIMEBUFFER) {
            sd.movetime = chronos.movetime - TIMEBUFFER;
        } else {
            sd.movetime = -1;
        }
        return;
    }

    /* in any other case we are given an ordinary timecontrol
       (time + opt. movestogo + opt. incremental)

        we take the total time left (chronos.time[sd.myside]), divide
it
        by the number of moves that are still to play (chronos.movesto
go)
        and we have the allowed time per move.
        If we are given an incremental time control, we consider this
        here as well.
```

```
        In case we are not told how many moves we are supposed to play
        with the time left, we assume a constant. This results in a
        slower play in the beginning and more rapid towards the end.

        as a little buffer we always assume that there are more moves
        to be played than we actually have to. This should avoid loosi
ng on time.
    */

    sd.movetime = 0;
    int movestogo = MOVESTOGO;
    if (chronos.flags & FMOVESTOGO) movestogo = chronos.movestogo + 2;

    if ( chronos.time[sd.myside] < 0 ) chronos.time[sd.myside] = 0;
    if ( chronos.inc[sd.myside]  < 0 ) chronos.inc[sd.myside]  = 0;

    if ( chronos.flags & FTIME) sd.movetime += chronos.time[sd.
myside] / movestogo;
    if ( chronos.flags & FINC ) sd.movetime += chronos.inc[sd.myside];

    if (sd.movetime > TIMEBUFFER)
        sd.movetime -= TIMEBUFFER;
    else
        sd.movetime = -1;

    return;

}

bool time_stop_root() {

    if (time_over) return 1;


/* in the root function we check at the beginning of every iteration,
       whether we should start calculating another cycle.
    */

    /* first check for any new commands (e.g. the stop command,
        which would set task to TASK_NOTHING)
    */
    com();
    if (task == TASK_NOTHING) return 1;
```

```
/* in case we are supposed to search to a certain depth, nodes count,
        fixed time or infinite, check whether we have finished the tas
k.
    */

    if ( chronos.flags & FINFINITE ) return 0;
    if ( chronos.flags & FDEPTH    ) return (sd.depth >
 chronos.depth);
    if ( chronos.flags & FNODES    ) return (sd.nodes >
 chronos.nodes);
    if ( chronos.flags & FMOVETIME ) return ((int)(gettime() - sd.
starttime) > sd.movetime);

    /* none of the other types of search-
duration controls were triggered.
        This means we are given an ordinary timecontrol.

        Now we ask the question, whether it is worth to search another

        ply, risk running out of time and having to abort  the search
        or if we should just stop the search now and save the time for
        another move.

        based on some tests in average positions (where the hash isn't
 full yet,
        there are not too many captures - resulting in a longer quiesc
ence search)
        the time for searching a ply is about the duration of the sear
ch
        of the previous plies.

        So:
        timeused = (gettime() - sd.starttime);
        predictedtime = timeused;
        timeleft = movetime - timeused;

        if (predictedtime > timeleft) we stop the search now


        Simplifying the equations:
        (predictedtime > timeleft)
        ( timeused > (movetime - timeused) )
        ( (timeused * 2) > movetime )

    */
```

```
    return (((int)(gettime() - sd.starttime) * 2) > sd.movetime);

}

bool time_stop() {

    /* time_stop() function is similar to time_stop_root(), only that
        this function is not called every time we reach another ply,
        but every couple thousand nodes. This function is very time
        consuming, so calling it too often will slow down the search.
        Calling it to seldom results in a slower response to input
        through the console and it might overlook the running out of
        time in faster games.
    */

    if (sd.depth <= 1) return 0;

    /* the first few lines are the same as in time_stop_root().
        for more comments see that function
    */

    com();
    if (task == TASK_NOTHING) return 1;

    if (chronos.flags & FINFINITE) return 0;
    if (chronos.flags & FDEPTH   ) return (sd.depth > chronos.depth);
    if (chronos.flags & FNODES   ) return (sd.nodes > chronos.nodes);
    if (chronos.flags & FMOVETIME) return ((int)(gettime() - sd.
starttime) > chronos.movetime );


/* again we are not doing one of the easier to manage time controls

        other than the function before we are now already in the middl
e of
        a search. Actually if our prediction was right, we wouldn't ha
ve
        entered this ply if we weren't able to finish it.

        so ( (gettime() - sd.starttime) < movetime ) should always be
given.

        Anyway there are some situations where our prediction method f
ails.
        The problem is that should we return true here and stop the se
arch
```

```
        a lot of information gathered gets useless, as the different l
ines
        searched are not verfied.

        if possible give in such cases a little overdraft and hope tha
t
        the misprediction was only small
    */

    if ((int)(gettime() - sd.starttime) > sd.movetime) {

        int movestogo = MOVESTOGO;
        if ( chronos.flags & FMOVESTOGO ) movestogo =
 chronos.movestogo;

        if ( ( movestogo > 5 ) &&
             ( (int)(gettime() - sd.starttime) < (sd.movetime * 2) )
&&
             ( sd.movetime > 5000) ) {
            return 0;
        } else {
            return 1;
        }

    }

    return 0;

}
```

What links here?

Page	Date Edited
CPW-Engine	Dec 31, 2014
CPW-Engine_chronos	Dec 30, 2014
Time Management	Mar 20, 2018

Up one Level