Table of Contents

Home * Engines * CPW-Engine * **com**

Description

This part of the engine deals with the communication with the GUI. The supported GUI formats are Xboard and UCI Protocol.

In the initialization process, if the engine is started through a program using one of these protocols, they identify themselves with "xboard" and "uci" respectively. In the com() function the variable mode is then set to store the protocol we communicate with.

com() is always called when the engine wants to check whether there is any new user input. gets(command) copies the input into command and returns true in case there has been a message.

The next step is that the commands get executed. This is now either done in com_xboard() or com_uci() depending on the GUI we are communicating with. Should we be running the engine without any GUI (e.g. through the command window for debugging), that means mode is set to 0, some basic commands get executed in the com() function.

Code

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

#include <windows.h>


enum etask {
    TASK_NOTHING,
```

```
    TASK_SEARCH,
    TASK_PONDER
} task = TASK_NOTHING;


enum eproto {
    PROTO_NOTHING,
    PROTO_XBOARD,
    PROTO_UCI
} mode = PROTO_NOTHING;

int debug = 0;

int pipe;
HANDLE hstdin;

int com_init() {

    unsigned long dw;
    hstdin = GetStdHandle(STD_INPUT_HANDLE);
    pipe = !GetConsoleMode(hstdin, &dw);

    if (!pipe) {
        SetConsoleMode(hstdin,dw&~(
ENABLE_MOUSE_INPUT|ENABLE_WINDOW_INPUT));
        FlushConsoleInputBuffer(hstdin);
    } else {
        setvbuf(stdin,NULL,_IONBF,0);
        setvbuf(stdout,NULL,_IONBF,0);
    }

    /* default search settings */
    chronos.movetime = 5000;
    chronos.flags = FMOVETIME;

    printWelcome();

    return 0;
}

int input() {

    unsigned long dw=0;

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

```
    if (stdin->_cnt > 0) return 1;

    if (pipe) {
        if (!PeekNamedPipe(hstdin, 0, 0, 0, &dw, 0)) return 1;
        return dw;
    } else {
        GetNumberOfConsoleInputEvents(hstdin, &dw);
        if (dw > 1) task = TASK_NOTHING;
    }

    return 0;
}


int com() {

    char command[65536];

    if (!input()) return 0;

    /* unwind the search-stack first */
    if (task == TASK_SEARCH) {
        task = TASK_NOTHING;
        return 0;
    }

    gets(command);

    switch (mode) {
    case PROTO_XBOARD :    com_xboard(command);  break;
    case PROTO_UCI    :    com_uci(command);     break;
    case PROTO_NOTHING: com_nothing(command); break;
    }

    return 0;
}

int com_nothing(char * command) {

    if (!strcmp(command, "xboard"))
                com_xboard(command);
        else if (!strcmp(command, "uci"))        com_uci(command);
        else if (!strncmp(command,"perft", 5))
    perft_start(command);
        else if (!strncmp(command,"bench", 5))    util_bench(command);
        else if (!strcmp(command, "eval"))      printEval();
```

```
        else if (!strcmp(command, "stat"))      printStats();
        else if (!strcmp(command, "d"))           board_display();
        else if (!strcmp(command, "new"))
        board_loadFromFen(STARTFEN);
        else if (!strncmp(command, "pos", 3))
    board_loadFromFen(command+4);
        else if (!strcmp(command, "go"))        time_nothing_go();
        else if (!strcmp(command, "quit"))        exit(0);
        else if (!strcmp(command, "help"))        printHelp();
        else if (com_ismove(command) ) {
                if ( algebraic_moves(command) )
                   time_nothing_go();
                else
                    printf("Sorry, this is not a legal move\n");
        }
                else if (!strncmp(command, "st", 2)) {
        sscanf(command, "st %d", &chronos.movetime);
        chronos.movetime *= 1000;
        chronos.flags = FMOVETIME;
        }
        else if (!strncmp(command, "sd", 2)) {
        sscanf(command, "sd %d", &chronos.depth);
        chronos.flags = FDEPTH;
        }
        else if (command[0] == '\n')            {}
        else {
            strcat(command,
" - UNKNOWN COMMAND (type 'help' for a list of commands)");
            com_send(command);
        }

    return 0;
}

int com_xboard(char * command) {

    if (!strcmp(command, "xboard"))
        mode = PROTO_XBOARD;

    if (!strcmp(command, "new"))
        board_loadFromFen(STARTFEN);

    else if (!strcmp(command, "force"))
        task = TASK_NOTHING;

    else if (!strcmp(command, "white"))
```

```
        sd.myside = WHITE;

    else if (!strcmp(command, "black"))
        sd.myside = BLACK;

    else if (!strncmp(command, "st", 2)) {
        sscanf(command, "st %d", &chronos.movetime);
        chronos.movetime *= 1000;
        chronos.flags = FMOVETIME;
    }

    else if (!strncmp(command, "sd", 2)) {
        sscanf(command, "sd %d", &chronos.depth);
        chronos.flags = FDEPTH;
    }

    else if (!strncmp(command, "time", 4)) {
        sscanf(command, "time %d", &chronos.time[sd.myside]);
        chronos.flags = FTIME;
    }

    else if (!strncmp(command, "otim", 4)) {
        sscanf(command, "otim %d", &chronos.time[!sd.myside]);
        chronos.flags = FTIME;
    }

    else if (!strcmp(command, "go"))
        time_xboard_go();

    else if (!strcmp(command, "hint")) {
        // hint
    }

    else if (!strcmp(command, "undo")) {
        // undo
    }

    else if (!strcmp(command, "remove")) {
        // remove
    }

    else if (!strcmp(command, "post")) {
        // post
    }

    else if (!strcmp(command, "nopost")) {
```

```
        // nopost
    }

    else if (!strcmp(command, "quit")) {
        exit(0);
    }

    else if (com_ismove(command)) {
        algebraic_moves(command);
        time_xboard_go();
    }

    return 0;
}


int com_uci(char * command) {

    if (!strcmp(command, "uci")) {
        mode = PROTO_UCI;

        com_send("id name CPW-Engine 1.1");
        com_send("id author Computer Chess Wiki");

        printf(
"option name Hash type spin default 64 min 1 max 1024\n");
        // send options

        com_send("uciok");
    }

    if (!strcmp(command, "isready"))
        com_send("readyok");

    if (!strncmp(command, "setoption", 9)) {
        char name[256];
        char value[256];

        sscanf(command, "setoption name %s value %s", &name, &value);

        if (!strcmp(name, "Hash")) {
            int val;
            sscanf(value, "%d", &val);
            tt_setsize(val<<20);
            ttpawn_setsize(val<<18);
        }
```

```
    }

    if (!strcmp(command, "ucinewgame")) {}

    if (!strncmp(command, "position", 8)) {
        //position [fen | startpos] [moves ...]

        if (!strncmp(command,"position fen",12)) {
            board_loadFromFen(command + 13);
        } else {
            board_loadFromFen(
"rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1");
        }

        char * moves = strstr(command, "moves");
        if (moves) algebraic_moves(moves+6);
    }

    if (!strncmp(command, "go", 2))
        time_uci_go(command);

    if (!strncmp(command, "debug", 5))
        debug = strcmp(command,"debug off");

    if (!strcmp(command, "ponderhit"))
        time_uci_ponderhit();

    if (!strcmp(command, "stop"))
        task = TASK_NOTHING;

    if (!strcmp(command, "quit"))
        exit(0);

    return 0;
}


int com_send(char * command) {
    printf("%s\n",command);
    return 0;
}

int com_sendmove(smove m) {

    int promotion = 0;
    char parray[5] = {0,'q','r','b','n'};
```

```
    char command[20];
    char move[6];

    switch (mode) {
    case PROTO_XBOARD: strcpy(command,"move "); break;
    case PROTO_UCI:    strcpy(command,"bestmove "); break;
    default: strcpy(command,"CPW: ");
    }

    convert_0x88_a(m.from, move);
    convert_0x88_a(m.to, move+2);

    //Promotion piece
    if (m.piece_to != m.piece_from) {
        promotion = m.piece_to;
    }
    move[4] = parray[promotion];
    move[5] = 0;

    strcat(command, move);

    com_send(command);

    /* in xboard and nothing actually do the move on the board */
    if (mode == PROTO_XBOARD || mode == PROTO_NOTHING)
        move_make(m);

    return 0;
}

int com_ismove(char * command) {
    return (command[0] >= 'a' && command[0] <= 'h' &&
            command[1] >= '1' && command[1] <= '8' &&
            command[2] >= 'a' && command[2] <= 'h' &&
            command[3] >= '1' && command[3] <= '8' &&
            ( command[4] == ' ' || command[4] == '\n' ||
 command[4] == 0 ||
              command[4] == '-' ||
              command[4] == 'q' || command[4] == 'r' ||
 command[4] == 'b' || command[4] == 'n'));

 /************************************************************
 *  command[4] might be:                                     *
 *                                                           *
 *  (a) any kind of a blank space                            *
```

```
 *    (b) '-' or any other mark used in opening book processing   *
 *    (c) first letter of a name of a promoted piece              *
 *****************************************************************/
}
```

What links here?

Page	Date Edited
Chess Engine Communication Protocol	Mar 4, 2018
CLI	Jan 20, 2018
CPW-Engine	Dec 31, 2014
CPW-Engine_com	Dec 30, 2014
CPW-Engine_main	Jun 13, 2011
GUI	Mar 16, 2018
UCI	Mar 4, 2018
WinBoard	Mar 4, 2018
XBoard	Jan 21, 2018

Up one Level