MasterMind

1. Principle

Gameplay:

The game is based on logical thinking, guessing and luck. The player has to guess what the color code is. You get tips how many colors are placed correctly and incorrectly. A + means correct color but misplaced. A X means that a color has been placed correctly.

Game declaration:

1. Create a color code with the letters a to g (depends on settings)
2. Press <Enter> to confirm the code (if you want to delete something, press <Backspace>)
3. Read on the right side how may were correct or misplaced
4. Type in a new code

Settings:

1. Type in the number oft he setting, you want to adjust
2. Type in the new Value
3. Press <Enter> to confirm the input
4. Press <p> to start the game

Help:

1. Press ingame <?>
2. Read the help
3. Close the window

2. Code

1. Mastermind.h

```
#pragma once

// include guard which prevents double including
#ifndef MASTERMIND_H
#define MASTERMIND_H

// this defines the maximum and minimum
// values of the preferences of the game
#define MMPREFS_MIN_CODE_LENGTH 2
#define MMPREFS_MAX_CODE_LENGTH 8
#define MMPREFS_MIN_COLOR_COUNT 4
#define MMPREFS_MAX_COLOR_COUNT 8
#define MMPREFS_MIN_ATTEMPT_COUNT 3
#define MMPREFS_MAX_ATTEMPT_COUNT 12
#define MM_COLOR_BGFG 0xF0

// this defines the frame for the game in Windows
#if defined(_WIN32) || defined(_WIN64)
    #define __usingwindows__ 1
    #define MMKEY_BACKSPACE '\b'
    #define KEY_BACKSPACE '\b'
    #define MMKEY_ENTER '\r'
    #define KEY_ENTER '\n'
    #define MMFRAME_TOP_LEFT_CORNER 201
    #define MMFRAME_TOP_RIGHT_CORNER 187
    #define MMFRAME_BOTTOM_LEFT_CORNER 200
    #define MMFRAME_BOTTOM_RIGHT_CORNER 188
    #define MMFRAME_LINE_HORIZONTALLY 205
    #define MMFRAME_LINE_VERTICALLY 186
    #define MMFRAME_TITLE_LEFT 185
    #define MMFRAME_TITLE_RIGHT 204
    typedef enum { false, true } bool;
// this defines the frame for the game in Linux
#elif defined(__linux__) || defined(__linux) || defined(linux)
    #define __usinglinux__ 1
    #define MMKEY_BACKSPACE '\b'
    #define MMKEY_ENTER '\n'
    #define MMFRAME_TOP_LEFT_CORNER '+'
    #define MMFRAME_TOP_RIGHT_CORNER '+'
    #define MMFRAME_BOTTOM_LEFT_CORNER '+'
    #define MMFRAME_BOTTOM_RIGHT_CORNER '+'
    #define MMFRAME_LINE_HORIZONTALLY '-'
    #define MMFRAME_LINE_VERTICALLY '|'
    #define MMFRAME_TITLE_LEFT '|'
    #define MMFRAME_TITLE_RIGHT '|'
    #include <stdbool.h>
#endif
```

```
// these are the preferences, which you will change in the settings
typedef struct {
    unsigned int code_length; // 2 - 8
    unsigned int color_count; // 4 - 8
    unsigned int attempt_count; // 3 - 12
    bool multiple_colors; // false - true
    bool hints_position_based; // false - true
       bool __comsolve__;
} MASTERMIND_PREFERENCES;

/// <summary>Manages which action has to be done.</summary>
/// <param name='initprefs'>Initial preferences.</param>
/// <param name='start_game_direct'>Skip settings screen and go directly to
game.</param>
/// <returns>Gives back an integer value of 0.</returns>
int mastermind(MASTERMIND_PREFERENCES initprefs, bool start_game_direct);

// end of MASTERMIND_H include guard
#endif
```

2. Mastermind.c

```
#include "mastermind.h"

// Windows systems
#if defined(__usingwindows__)
// Visual Studio won't compile mastermind without it (because of strncpy)
    #define _CRT_SECURE_NO_WARNINGS
    #include <stdio.h>
    #include <Windows.h>
// Linux systems
#elif defined(__usinglinux__)
    // sudo apt-get install libncurses5-dev libncursesw5-dev
    // gcc ... -lcurses
    // Note: curses.h auto-includes stdio.h !!
    #include <curses.h>
    // macro definitions for compatibility between stdio.h and curses.h
    #define putchar(c) addch(c)
    #define printf(f, ...) printw(f, ## __VA_ARGS__)
    #define getch() wgetch(stdscr)
#else
    #error "The OS-type could not be determined or is not supported..."
#endif

#include <stdlib.h>
#include <string.h>
#include <time.h>

// defines the variables for the width and height of the console
// also defines the variables for the initial x and y position of the cursor
struct mmwidgetinfo {
    int width,
        height,
        initc_x,
        initc_y;
};
// defines handler for mastermind() function
enum mmaction {
    MMACT_QUIT, MMACT_OPENSETTINGS, MMACT_STARTGAME
};
```

```
/*********************************/
/*** NON-EXPORT FUNCTIONS (NEX) ***/
/*********************************/

// gets actual console cursor position
void nex_getcursorpos(int *x, int *y) {
    #if defined(__usingwindows__)
    HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO info;
    GetConsoleScreenBufferInfo(hout, &info);
    *x = info.dwCursorPosition.X;
    *y = info.dwCursorPosition.Y;
    #elif defined(__usinglinux__)
    *x = 0;
    *y = 0;
    #endif
}
// sets console cursor to column (x value) and row (y value)
void nex_setcursorpos(unsigned int column, unsigned int row) {
    #if defined(__usingwindows__)
    HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD size = { .X = column,.Y = row };
    SetConsoleCursorPosition(hout, size);
    #elif defined(__usinglinux__)
    move(row, column);
    refresh();
    #endif
}
// sets the console text color
void nex_setcolor(int value) {
    #if defined(__usingwindows__)
    HANDLE hout = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hout, value);
    #elif defined(__usinglinux__)
    #endif
}
// draws the frame for the game
void nex_drawframe(unsigned int width, unsigned int height, char *title) {
    putchar('\r'); // go to beginning of line
    bool title_show = true; // controls if title is shown
    // if area is smaller than 3x3 then stop
    if (height < 3 || width < 3) return;
    // if width < 20 then do not show title
    if (width < 10 || strlen(title) == 0) title_show = false;
    int i, j; // counter variables
    for (i = 0; i < height; i++) {
        if (i == 0) {
            putchar(MMFRAME_TOP_LEFT_CORNER);
            if (title_show) {
                int len = strlen(title) > width - 6 ? width - 6 : strlen(title),
                    len_field = len + len % 2;
                for (j = 0; j < (width - len_field - 6) / 2; j++)
putchar(MMFRAME_LINE_HORIZONTALLY);
                putchar(MMFRAME_TITLE_LEFT);
                putchar(' ');
                for (j = 0; j < len; j++) putchar(title[j]);
                putchar(' ');
                putchar(MMFRAME_TITLE_RIGHT);
                for (j = 0; j < (width - len_field - 6) / 2 + len % 2; j++)
putchar(MMFRAME_LINE_HORIZONTALLY);
            }
```

```
            else {
                for (j = 0; j < width - 2; j++)
putchar(MMFRAME_LINE_HORIZONTALLY); // width - 2 because of the corner chars
            }
            putchar(MMFRAME_TOP_RIGHT_CORNER);
            putchar('\n');
        }
        else if (i == height - 1) {
            putchar(MMFRAME_BOTTOM_LEFT_CORNER);
            for (j = 0; j < width - 2; j++) putchar(MMFRAME_LINE_HORIZONTALLY);
            putchar(MMFRAME_BOTTOM_RIGHT_CORNER);
            putchar('\n');
        }
        else {
            putchar(MMFRAME_LINE_VERTICALLY);
            for (j = 0; j < width - 2; j++) putchar(' ');
            putchar(MMFRAME_LINE_VERTICALLY);
            putchar('\n');
        }
    }
}
// write interactive phrases
void nex_writephrase(MASTERMIND_PREFERENCES x, struct mmwidgetinfo w, char
*text) {
    nex_setcursorpos(2, w.initc_y + x.attempt_count + 5);
    printf("%s", text);
}
// same like pow() function in real mathematic
int nex_pow(int base, int exponent){
    int i, rt = 1; // rt means return value
    for (i = 0; i < exponent; i++)
        rt *= base;
    return rt;
}
// creates the help-window for the game
void nex_help(unsigned int width, unsigned int height) {
    #if defined(__usingwindows__)
    system("start \"MasterMind Help\" cmd /C \" echo off & \
    mode con: cols=80 lines=15 & \
    color F0 & \
    echo MasterMind v0.5.1.1246 (beta) & \
    echo (c) 2018 Patrick Goldinger and Matthias Gehwolf & \
    echo. & \
    echo CODE-LENGTH:           2- 8  ! can be from 2 to 8 characters & \
    echo COLOR-COUNT:           4- 8  ! can be from 4 to 8 colors in range of a-
h & \
    echo ATTEMPT-COUNT:         3-12  ! characters & \
    echo MULTIPLE-COLORS:       0- 1  ! if 1 colors can appear more than once &
\
    echo HINTS-POSITION-BASED:  0- 1  ! if 1 hints are for each position & \
    echo. & \
    pause\"");
    // Not available on Linux
    #elif defined(__usinglinux__)
    // coming soon
    #endif
}
```

```
// checks if the preferences are in the minimum and maximum range
bool nex_parametercheck(MASTERMIND_PREFERENCES prefs) {
    return
        !(prefs.code_length < MMPREFS_MIN_CODE_LENGTH || prefs.code_length >
MMPREFS_MAX_CODE_LENGTH ||
        prefs.color_count < MMPREFS_MIN_COLOR_COUNT || prefs.color_count >
MMPREFS_MAX_COLOR_COUNT ||
        prefs.attempt_count < MMPREFS_MIN_ATTEMPT_COUNT || prefs.attempt_count >
MMPREFS_MAX_ATTEMPT_COUNT ||
        (prefs.color_count < prefs.code_length && !prefs.multiple_colors));
}
// checks if a given character (to_search) exists in an array (*base_array)
bool nex_isinarray(char to_search, char *base_array, int base_array_length) {
    int i;
    for (i = 0; i < base_array_length; i++) {
        if (*(base_array + i) == to_search)
            return true;
    }
    return false;
}
// draws spaces over the game and resets cursor to initial position
enum mmaction nex_clearandexit(struct mmwidgetinfo w, enum mmaction r) {
    nex_setcursorpos(0, w.initc_y);
    nex_setcolor(0x0F);
    int i, j;
    for (i = 0; i < w.height; i++) {
        for (j = 0; j < w.width; j++) putchar(' ');
        putchar('\n');
    }
    nex_setcursorpos(0, w.initc_y);
    return r;
}
// set the message, which will be shown at the end of the game
// after that the game waits for the anwser of the player
enum mmaction nex_gamecomplete(MASTERMIND_PREFERENCES prefs, struct mmwidgetinfo
w, char *code_secret, bool victory) {
    char tmp;
    int n;
    nex_setcursorpos(2, w.initc_y + 2);
    for (n = 0; n < prefs.code_length; n++) { // show initial code
        nex_setcolor(0xF1 + (code_secret[n] - 'a'));
        putchar(code_secret[n]);
        nex_setcolor(MM_COLOR_BGFG);
        putchar(32);
    }
```

```
if (prefs.__comsolve__) {
              if (victory) // print different messages based on victory
                     nex_writephrase(prefs, w, "The COMSOLVER won the game.
Again? <r>");
              else
                     nex_writephrase(prefs, w, "The COMSOLVER lost the game.
Again? <r>");
       }
       else {
              if (victory) // print different messages based on victory
                     nex_writephrase(prefs, w, "Congrats, you won! Play again?
<r>");
              else
                     nex_writephrase(prefs, w, "You lost! Play again? <r>");
       }
    while (1) { // now listen on further actions of user
        tmp = getch();
        if (tmp == 'q')
            return nex_clearandexit(w, MMACT_QUIT);
        else if (tmp == 'r')
            return nex_clearandexit(w, MMACT_STARTGAME);
        else if (tmp == '?')
            nex_help(w.width, w.height);
        else if (tmp == 27)
            return nex_clearandexit(w, MMACT_OPENSETTINGS);
    }
}
```

```
/***************************/
/*** MASTERMIND FUNCTIONS ***/
/***************************/

/// <summary>Main game function which displays UI and handles the
behind.</summary>
/// <param name='prefs'>Preferences of game.</param>
/// <returns>An action to do after the game is done.</returns>
enum mmaction mastermind_game(MASTERMIND_PREFERENCES prefs) {

    // --< counter variables >--
    int n, m;

    // --< widget user interface setup >--
    // draws the frame of the game
    putchar('\r'); // reset x coordinate
    struct mmwidgetinfo widget = { 50, prefs.attempt_count + 9, 0, 0 }; //
width: 50; height: attemps + 9
    nex_getcursorpos(&widget.initc_x, &widget.initc_y); // get init cursor
position
    nex_setcolor(MM_COLOR_BGFG); // set color defined in mastermind.h
    nex_drawframe(widget.width, widget.height, "MasterMind"); // draw frame with
width and height attributes

    // --< parameter check >--
    // checks if the preferences are in the maximum and minimum range
    if (!nex_parametercheck(prefs)) {
        nex_setcursorpos(2, widget.initc_y + 2);
        printf("Got incorrect parameters.");
        nex_setcursorpos(2, widget.initc_y + 4);
        printf("Press any key to continue to settings...");
        getch();
        // return from mastermind_game() and trigger settings screen
        return nex_clearandexit(widget, MMACT_OPENSETTINGS);
    }

    // --< draw layout of points and commands available >--
    // draws the inside layout for the points and commands available
    // row 3 : question marks as placeholder for secret code and attempt count
    nex_setcursorpos(2, widget.initc_y + 2);
    for (n = 0; n < prefs.code_length; n++)
        printf("? ");
    printf("|"); // after this position the attempt counter will be written
    // row 4 : layout spacier (_____)
    nex_setcursorpos(1, widget.initc_y + 3);
    for (n = 0; n < widget.width - 2; n++)
        printf("_");
    // row 5+ : attempt input and output structure
    for (n = 0; n < prefs.attempt_count; n++) {
        nex_setcursorpos(2, widget.initc_y + n + 4);
        for (m = 0; m < prefs.code_length; m++)
            printf(". ");
        printf("| ");
        for (m = 0; m < prefs.code_length; m++)
            printf(". ");
    }
    // row -5 : layout spacier (_____)
    nex_setcursorpos(1, widget.initc_y + prefs.attempt_count + 4);
       for (n = 0; n < widget.width - 2; n++)
```

```
        printf("_");
    // row -2 : List commands available and their char trigger
    nex_setcursorpos(2, widget.initc_y + prefs.attempt_count + 7);
    printf("Quit <q>       Help <?>       Settings <ESC>");

    // --< setup game variables >--
    // sets the secret code
    srand(time(NULL)); // generate new seed for rand()
    char code_secret[MMPREFS_MAX_CODE_LENGTH], // holds the random generated
secret code
        code_in[MMPREFS_MAX_CODE_LENGTH], // holds the code input
             code_comsol[MMPREFS_MAX_CODE_LENGTH], // holds the computer solved
code
             solution_out[MMPREFS_MAX_CODE_LENGTH]; //
    int tmp; // temporary character storage
    for (n = 0; n < prefs.code_length; n++) {
        do code_secret[n] = 'a' + rand() % prefs.color_count;
        while (nex_isinarray(code_secret[n], code_secret, n) &&
!prefs.multiple_colors);
    }

    // --< main game loop >--
    // this is the main game
    int cattp; // current attempt position
    for (cattp = 0; cattp < prefs.attempt_count; cattp++) {
        int marker = 0; // marker for char input count
        nex_setcursorpos(2 * prefs.code_length + 4, widget.initc_y + 2); // set
cursor to position beside ? ? ? ? |
        printf("Attempt %d/%d", cattp + 1, prefs.attempt_count); // write
attempt count
             // COMSOLVER begin
             if (prefs.__comsolve__) {
                    int cc;
                    char code_next[MMPREFS_MAX_CODE_LENGTH];
                    if (cattp < 6) {
                           for (cc = 0; cc < prefs.code_length; cc++)
                                  code_next[cc] = 'a' + cattp;
                    }
                    if (cattp > 0) {
                           for (cc = 0; cc < prefs.code_length; cc++) {
                                  if (solution_out[cc] == 'X')
                                         code_comsol[cc] = code_in[cc];
                           }
                    }
                    if (cattp == prefs.attempt_count - 1) { // when all try
attempts have been done
                           for (cc = 0; cc < prefs.code_length; cc++)
                                  code_next[cc] = code_comsol[cc];
                    }
                    strncpy(code_in, code_next, prefs.code_length);
             }
             // COMSOLVER end
        while (1) { // inner attempt loop
            if (marker == prefs.code_length) // input is complete and ready to
be confirmed by user
                nex_writephrase(prefs, widget, "Input ok? <ENTER>");
            else // input is not ready to be confirmed - delete input ok phrase
                nex_setcursorpos(2 * marker + 2, widget.initc_y +
prefs.attempt_count - cattp + 3);
                    tmp = prefs.__comsolve__ ? code_in[marker] : getch(); //
wait for user input
            if (tmp == 'q') // q -> quit the game directly
```

```
                return nex_clearandexit(widget, MMACT_QUIT);
            else if (tmp >= 'a' && tmp <= ('a' + prefs.color_count - 1) &&
marker < prefs.code_length) { // a - h -> char input
                code_in[marker++] = tmp;
                nex_setcolor(0xF1 + (tmp - 'a'));
                putchar(tmp);
                nex_setcolor(MM_COLOR_BGFG);
            }
            else if ((tmp == MMKEY_BACKSPACE || tmp == '*' || tmp ==
KEY_BACKSPACE) && marker > 0) { // \b -> backspace
                code_in[--marker] = tmp;
                nex_setcursorpos(2 * marker + 2, widget.initc_y +
prefs.attempt_count - cattp + 3);
                putchar('.');
                nex_writephrase(prefs, widget, "                    ");
            }
            else if ((tmp == MMKEY_ENTER || tmp == '\'' || tmp == KEY_ENTER ||
prefs.__comsolve__) && marker == prefs.code_length) { // \n \r ... -> input
complete
                char points_out[MMPREFS_MAX_CODE_LENGTH] = { '.', '.', '.', '.',
'.', '.', '.', '.' }, // needed for algorithm, not for output!!
                    code_secret_tmp[MMPREFS_MAX_CODE_LENGTH], // holds clone of
real secret code
                    code_in_tmp[MMPREFS_MAX_CODE_LENGTH]; // holds clone of real
secret code
                int points_out_idx = 0, // counts the written chars for correct
filling of points_out[]
                    victory_count = 0; // counts the number of 'X'ses
                strncpy(code_secret_tmp, code_secret, MMPREFS_MAX_CODE_LENGTH);
// safe clone (cannot use code_secret_tmp = code_secret)!!
                strncpy(code_in_tmp, code_in, MMPREFS_MAX_CODE_LENGTH); // safe
clone (cannot use code_in_tmp = code_in)!!
                nex_setcursorpos(2 * prefs.code_length + 4, widget.initc_y +
prefs.attempt_count - cattp + 3);
                // X detection and output
                for (n = 0; n < prefs.code_length; n++) {
                    if (code_secret_tmp[n] == code_in_tmp[n] &&
code_secret_tmp[n] != 0) {
                        if (prefs.hints_position_based) {
                            points_out[n] = 'X'; // color and position right
                            nex_setcursorpos(2 * prefs.code_length + 4 + 2 * n,
widget.initc_y + prefs.attempt_count - cattp + 3);
                            printf("X ");
                        }
                        else {
                            points_out[points_out_idx++] = 'X'; // color and
position right
                            printf("X ");
                        }
                        code_secret_tmp[n] = 0;
                        code_in_tmp[n] = 0;
                        victory_count++;
                    }
                }
                // + detection and output
                for (n = 0; n < prefs.code_length; n++) {
                    for (m = 0; m < prefs.code_length; m++) {
                        if (code_in_tmp[n] == code_secret_tmp[m] && n != m &&
code_secret_tmp[m] != 0) {
                            if (prefs.hints_position_based) {
                                points_out[n] = '+'; // color only right
```

```
                                nex_setcursorpos(2 * prefs.code_length + 4 + 2 *
n, widget.initc_y + prefs.attempt_count - cattp + 3);
                                printf("+ ");
                            }
                            else {
                                points_out[points_out_idx++] = '+'; // color
only right
                                printf("+ ");
                            }
                            code_secret_tmp[m] = 0;
                            code_in_tmp[n] = 0;
                        }
                    }
                }

if (victory_count == prefs.code_length) // if victory_count equals code length
it must be a victory
                    return nex_gamecomplete(prefs, widget, code_secret, true);
                nex_writephrase(prefs, widget, "                    "); // clean
up input ok phrase
                            strncpy(solution_out, points_out,
prefs.code_length);
                break; // next attempt input or finish
            }
            else if (tmp == '?') // ? -> help
                nex_help(widget.width, widget.height); // open help window
                 else if (tmp == 27) { // \033 -> escape (settings screen)
                            nex_writephrase(prefs, widget, "Are you sure? (game
will be resetted) <y/n>");
                            bool input_ok = false;
                            while (!input_ok){
                                    tmp = getch(); // wait for input
                    if (tmp == 'n') { // n -> go on with game
                        input_ok = true;
                                     nex_writephrase(prefs, widget, "
");
                    }
                    else if (tmp == 'y') // y -> exit game and open settings
screen
                        return nex_clearandexit(widget, MMACT_OPENSETTINGS);
                            }
                 }
        } // <-- end of while (attempt)
    } // <-- end of main game loop's for (n = 0; ...)

    // --< after game cleanup (you lost the game when you reach this point) >--
    return nex_gamecomplete(prefs, widget, code_secret, false);
}

/// <summary>Manages the settings screen so users can edit the prefs
object.</summary>
/// <param name='initprefs'>Initial preferences before editing.</param>
/// <param name='newprefs'>Pointer to memory address where new prefs should be
written.</param>
/// <returns>An action to do after the settings screen is done.</returns>
enum mmaction mastermind_settings(MASTERMIND_PREFERENCES initprefs,
MASTERMIND_PREFERENCES *newprefs) {

    // --< variables >--
    // declares variables
    int n, m, tmp_int;
```

```
    int tmp; // used as char, but on unix chars may be 2 bytes so int is the
solution
    MASTERMIND_PREFERENCES tmpprefs = initprefs;
       tmpprefs.__comsolve__ = false;

    // --< widget UI interface setup >--
    // sets UI for the setting-screen
    putchar('\r'); // reset x coordinate
    struct mmwidgetinfo widget = { 50, 14 + 6, 0, 0 }; // +6 = note on COMSOLVER
    nex_getcursorpos(&widget.initc_x, &widget.initc_y);
    nex_setcolor(MM_COLOR_BGFG);
    nex_drawframe(widget.width, widget.height, "MasterMind Settings");

    // --< main and settings loop >--
    // sets setting screen and main settings loop
    while (1) {
        // Help
        nex_setcursorpos(2, widget.initc_y + 2);
        printf("To edit entries, type the number.");
        // check and output settings for the code length
        nex_setcursorpos(2, widget.initc_y + 4);
        if (tmpprefs.code_length < MMPREFS_MIN_CODE_LENGTH ||
            tmpprefs.code_length > MMPREFS_MAX_CODE_LENGTH)
            tmpprefs.code_length = 4;
        printf("#1 code length          [ 2; 8] = %d  ", tmpprefs.code_length);
        // check and output settings for the color count
        nex_setcursorpos(2, widget.initc_y + 5);
        if (tmpprefs.color_count < MMPREFS_MIN_COLOR_COUNT ||
            tmpprefs.color_count > MMPREFS_MAX_COLOR_COUNT)
            tmpprefs.color_count = 6;
        printf("#2 color count          [ 4; 8] = %d  ", tmpprefs.color_count);
        // check and output settings for the attempt count
        nex_setcursorpos(2, widget.initc_y + 6);
        if (tmpprefs.attempt_count < MMPREFS_MIN_ATTEMPT_COUNT ||
            tmpprefs.attempt_count > MMPREFS_MAX_ATTEMPT_COUNT)
            tmpprefs.attempt_count = 7;
        printf("#3 attempt count        [ 3;12] = %d  ",
tmpprefs.attempt_count);
        // settings for multiple colors
        nex_setcursorpos(2, widget.initc_y + 7);
        printf("#4 multiple colors      [ 0; 1] = %d  ",
tmpprefs.multiple_colors);
        // settings for position based hints
        nex_setcursorpos(2, widget.initc_y + 8);
        printf("#5 hints position based [ 0; 1] = %d  ",
tmpprefs.hints_position_based);
        // draws a line
        nex_setcursorpos(1, widget.initc_y + 9);
        for (n = 0; n < widget.width - 2; n++)
            putchar('_');
        // writes a quit, computer-solve and play text
        nex_setcursorpos(2, widget.initc_y + 12);
        printf("Quit <q>        COMSOLVER <c>        Play <p>");
             // Write note for COMSOLVER
        nex_setcursorpos(2, widget.initc_y + 14);
        printf("Note: COMSOLVER is currently only availabe if:");
        nex_setcursorpos(2, widget.initc_y + 15);
             printf("   #1 == 4 && #2 == 6 && #3 == 7 &&");
        nex_setcursorpos(2, widget.initc_y + 16);
             printf("   #4 == 0 && #5 == 1");
        nex_setcursorpos(2, widget.initc_y + 17);
             printf("   This algorithm is in alpha state, therefore");
```

```
            nex_setcursorpos(2, widget.initc_y + 18);
                  printf("   it has this restrictions.");
            // input of the player
            nex_setcursorpos(2, widget.initc_y + 10);
            printf("edit setting:             \b\b\b\b\b\b\b\b\b\b\b");
            tmp = getch();
            tmp_int = tmp - '1' + 1;
            if (tmp == 'q') // quit
                return nex_clearandexit(widget, MMACT_QUIT);
            if (tmp == 'p') { // play with new settings
                        tmpprefs.__comsolve__ = false;
                *newprefs = tmpprefs;
                return nex_clearandexit(widget, MMACT_STARTGAME);
            }
            else if (tmp == 'c') { // let computer solve with new settings
                        if (tmpprefs.code_length != 4
                               || tmpprefs.color_count != 6
                               || tmpprefs.attempt_count != 7
                               || tmpprefs.multiple_colors
                               || !tmpprefs.hints_position_based) {
                              nex_setcursorpos(2, widget.initc_y + 10);
                              printf("Invalid settings, see note below! <any>");
                              getch();
                              nex_setcursorpos(2, widget.initc_y + 10);
                              printf("                                          ");
                        }
                        else {
                              tmpprefs.__comsolve__ = true;
                              *newprefs = tmpprefs;
                              return nex_clearandexit(widget, MMACT_STARTGAME);
                        }
            }
            else if (tmp_int >= 1 && tmp_int <= 5) {
                char buf[2] = { 0, 0 };
                int marker = 0;
                nex_setcursorpos(2, widget.initc_y + 10);
                printf("enter new value of (%d): ", tmp_int);
                while (1) {
                    tmp = getch();
                    if (tmp == 'q')
                        return nex_clearandexit(widget, MMACT_QUIT);
                    else if (tmp == 27)
                        break;
                    else if ((tmp - '0') >= 0 && (tmp - '0') <= 9 && marker < 2) {
                        buf[marker++] = tmp;
                        putchar(tmp);
                    }
                    else if ((tmp == MMKEY_BACKSPACE || tmp == '*' || tmp ==
KEY_BACKSPACE) && marker > 0) {
                        printf("\b \b");
                        marker--;
                    }
                    else if ((tmp == MMKEY_ENTER || tmp == '\'' || tmp == KEY_ENTER)
&& marker > 0) {
                        int new_val = 0;
                        for (n = 0; n < marker; n++)
                            new_val += (buf[n] - '0') * nex_pow(10, marker - n - 1);
                        if (tmp_int == 1)
                            tmpprefs.code_length = new_val;
                        else if (tmp_int == 2)
                            tmpprefs.color_count = new_val;
                        else if (tmp_int == 3)
```

```
                        tmpprefs.attempt_count = new_val;
                    else if (tmp_int == 4 && new_val >= 0 && new_val <= 1)
                        tmpprefs.multiple_colors = new_val;
                    else if (tmp_int == 5 && new_val >= 0 && new_val <= 1)
                        tmpprefs.hints_position_based = new_val;
                    break;
                }
            }
        }
    }

    return nex_clearandexit(widget, MMACT_STARTGAME);
}

/// <summary>Manages which action has to be done.</summary>
/// <param name='initprefs'>Initial preferences.</param>
/// <param name='start_game_direct'>Skip settings screen and go directly to
game.</param>
/// <returns>Gives back an integer value of 0.</returns>
int mastermind(MASTERMIND_PREFERENCES initprefs, bool start_game_direct) {
    #if defined(__usinglinux__)
    // initialize screen for curses library
    initscr();
    noecho();
    keypad(stdscr, TRUE);
    #endif
    enum mmaction retstate = MMACT_OPENSETTINGS;
    MASTERMIND_PREFERENCES newprefs = initprefs;
    while (1) {
        if (start_game_direct) {
            start_game_direct = false;
            retstate = mastermind_game(initprefs);
        }
        else if (retstate == MMACT_QUIT)
            break;
        else if (retstate == MMACT_OPENSETTINGS)
            retstate = mastermind_settings(newprefs, &newprefs);
        else if (retstate == MMACT_STARTGAME)
            retstate = mastermind_game(newprefs);
    }
    #if defined(__usinglinux__)
    // clean up screen for further use of the 'normal' shell
    endwin();
    #endif
    return 0;
}
```

3. Main.c

```c
#include "mastermind.h"


//start code
int main(void) {
    MASTERMIND_PREFERENCES myprefs = {
        .code_length = 4,
        .color_count = 6,
        .attempt_count = 7,
        .multiple_colors = false,
        .hints_position_based = false
    };
    mastermind(myprefs, false);

    return 0;
}
```