```c
/**
* @file colorsBack.c
* Command handling
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <ctype.h>
#include "error.h"
#include "utils.h"
#include "defines.h"
#include "playGame.h"
#include "colorsBack.h"

bool movePiece ( game_t * game, int argc, char ** argv, char * msg );
bool save      ( game_t * game, int argc, char ** argv, char * msg );
bool undo      ( game_t * game, int argc, char ** argv, char * msg );
bool quit      ( game_t * game, int argc, char ** argv, char * msg );
bool help      ( game_t * game, int argc, char ** argv, char * msg );
bool roflcopter( game_t * game, int argc, char ** argv, char * msg );

typedef struct{
    char com[ MAX_COM_LEN ];
    bool (* func )( game_t *, int, char **, char * );
} command_t;

const command_t commands[] = {
    {"[", movePiece},
    {"save", save},
    {"undo", undo},
    {"quit", quit},
    {"help", help},
    {"ROFLcopter",roflcopter}
};


/**
* Parses the command and calls the corresponding function.
*
* @param game       contains all information about current game
* @param s          containins the command line about to be processed
* @param[out] msg   its an output containing the type of error
*
* @return false if there is an error, otherwise true
*
* @see editDistance()
* @see movePiece()
* @see save()
* @see undo()
* @see quit()
* @see help()
* @see ROFLcopter()
*/

bool
newCommand( game_t * game, const char * s, char * msg )
{
    int i;
    bool sol;
    int argc;
    char ** argv;
```

```
64
65      if( errorCode() != NOERROR ){
66          sprintf( msg, "%s", errorMessage( errorCode() ) );
67          return false;
68      }
69
70      argv = newMatrix( MAX_ARGS, MAX_COM_LEN );
71
72      if( errorCode() != NOERROR ){
73          sprintf( msg, "%s", errorMessage( errorCode() ) );
74          return false;
75      }
76
77      msg[0] = 0;
78      // parse the command
79      argc = sscanf( s, "%s %s %s %s %s %s %s %s %s %s",
80                        argv[0], argv[1], argv[2], argv[3], argv[4],
81                        argv[5], argv[6], argv[7], argv[8], argv[9] );
82
83      if( argc < 1  ){
84          freeMatrix( argv, MAX_ARGS );
85          return true;
86      }
87
88      sol = false;
89      sprintf( msg, "Unknown command" );
90      double auxsim, maxsim = 0;
91      int maxi = 0;
92
93      //call the appropiate function
94      for( i = 0 ; i < sizeof(commands)/ sizeof(command_t) ; i++ ){
95          if( strncmp( argv[0], commands[i].com, strlen(commands[i].com) ) == 0
96                                  && !isalpha( argv[0][strlen(commands[i].com)] ) ){
97              maxsim = 0;
98              msg[0] = 0;
99              sol = commands[i].func( game, argc, argv, msg );
100             if( errorCode() != NOERROR ){
101                 sprintf( msg, "%s", errorMessage( errorCode() ) );
102                 sol = false;
103             }
104             break;
105         }
106         // check for command similarity
107         if( ( auxsim = editDistance( argv[0], commands[i].com ) ) > maxsim ){
108             maxsim = auxsim;
109             maxi = i;
110         }
111     }
112     // if not succesful and there was a command similar enough
113     if( maxsim >= MIN_SIMILARITY )
114         sprintf( msg+15, "\nDid you mean: \"%s\"", commands[maxi].com );
115
116     freeMatrix( argv, MAX_ARGS );
117
118     if( errorCode() != NOERROR ){
119         sprintf( msg, "%s", errorMessage( errorCode() ) );
120         sol = false;
121     }
122
123     return sol;
124 }
125
126
```

```
127 /**
128 * Checks if there is a valid path between (@a x1,@a y1) and (@a x2,@a y2)
129 * in the board.
130 *
131 * @param game   contains all information about current game
132 * @param x1     initial x coordenate
133 * @param y1     initial y coordenate
134 * @param x2     final x coordenate
135 * @param y2     final y coordenate
136 *
137 * @return true if there is a valid path, otherwise false
138 */
139
140 bool
141 areConnected( game_t * game, int x1, int y1, int x2, int y2 )
142 {
143     // BFS to find minimum path
144     struct coord{
145         int x,y;
146     } move[4] = { {-1,0}, {0,1}, {1,0}, {0,-1} };
147
148     struct node{
149         int x, y;
150     } queue[ game->players[ game->state.next ].board.emptySpots + 1 ];
151
152     int read = -1, write = 0, x, y, i;
153
154     bool touched[ game->options.height ][ game->options.width ];
155
156     memset( &touched[0][0], false, sizeof(touched) * sizeof(bool) );
157     queue[write++] = (struct node){x1,y1};
158
159     while( ++read < write ){
160         if( queue[read].x == x2 && queue[read].y == y2 )
161             break;
162         for( i = 0 ; i < sizeof(move)/sizeof(struct coord) ; i++ ){
163             x = queue[read].x + move[i].x;
164             y = queue[read].y + move[i].y;
165             if( entre( 0, x, game->options.width )
166                 && entre( 0, y, game->options.height )
167                 && game->players[ game->state.next ].board.matrix[y][x] == 0
168                 && !touched[y][x] ){
169                         touched[y][x] = true;
170                         queue[write++] = (struct node){x,y};
171             }
172         }
173     }
174     return read < write;
175 }
176
177
178 /**
179 * Moves token from a position to another checking for winning plays (lines).
180 *
181 * @param game       contains all information about current game
182 * @param argc       size of @a argv
183 * @param argv       parameters followinf the command
184 * @param[out] msg   output with message to write in the panel
185 *
186 * @return false if some message is to be written
187 *
188 * @see areConected()
189 * @see randFill()
```

```
190 * @see winningPlay()
191 */
192
193 bool
194 movePiece( game_t * game, int argc, char ** argv, char * msg )
195 {
196     int i;
197     char s[ argc * MAX_ARGS ];
198
199     s[0] = 0;
200     for( i = 0 ; i < argc ; i++ )
201         strcat( s, argv[i] );
202
203     int x1, y1, x2, y2;
204     i = sscanf( s, "[%d,%d][%d,%d]", &y1, &x1, &y2, &x2 );
205     if( i < 4 ){
206         sprintf( msg, "Format error:\n"
207                     "Must be: [ row_1, column_1 ][ row_2, column_2 ]" );
208         return false;
209     }
210     if( ! entre( 0, y1, game->options.height ) ){
211         sprintf( msg, "Rank error:\nThe first row must belong to the "
212                     "interval [0,%d]", game->options.height - 1 );
213         return false;
214     }
215     if( ! entre( 0, x1, game->options.width ) ){
216         sprintf( msg, "Rank error:\nThe first column must belong to the "
217                     "interval [0,%d]", game->options.width - 1 );
218         return false;
219     }
220     if( ! entre( 0, y2, game->options.height ) ){
221         sprintf( msg, "Rank error:\nThe second row must belong to the "
222                     "interval [0,%d]", game->options.height - 1 );
223         return false;
224     }
225     if( ! entre( 0, x2, game->options.width ) ){
226         sprintf( msg, "Rank error:\nThe second column must belong to the "
227                     "interval [0,%d]", game->options.width - 1 );
228         return false;
229     }
230     if( game->players[ game->state.next ].board.matrix[y1][x1] == 0 ){
231         sprintf( msg, "The origin position must not be empty" );
232         return false;
233     }
234     if( game->players[ game->state.next ].board.matrix[y2][x2] != 0 ){
235         sprintf( msg, "The target position must not be occupied" );
236         return false;
237     }
238     if( !areConnected( game, x1, y1, x2, y2 ) ){
239         sprintf( msg, "There must be a path of unoccupied spaces from the "
240                     "origin position to the target position" );
241         return false;
242     }
243     // maintain undo
244     game->players[ game->state.next ].canUndo = true;
245
246     copyMatrix( game->players[ game->state.next ].lastBoard. matrix,
247                 game->players[ game->state.next ].board.matrix,
248                 game->options.height, game->options.width );
249
250     game->players[ game->state.next ].lastBoard.points =
251                                 game->players[ game->state.next ].board.points;
252
```

```
253     game->players[ game->state.next ].lastBoard.emptySpots =
254                             game->players[ game->state.next ].board.emptySpots;
255     // maintain board
256     game->players[ game->state.next ].board.matrix[y2][x2] =
257                     game->players[ game->state.next ].board.matrix[y1][x1];
258 
259     game->players[ game->state.next ].board.matrix[y1][x1] = 0;
260 
261     // if made a line, erase it and count points (winning play)
262     if( ! winningPlay( game, game->state.next, x2, y2, true ) ){
263         // else, randFill()
264         randFill( game, game->state.next, game->options.tokensPerTurn, false );
265         // change turn
266         i = 0;
267         do{
268             game->state.next++;
269             game->state.next %= game->numPlayers;
270             i++;
271         }while( game->players[ game->state.next ].board.emptySpots <= 0 &&
272                     i <= game->numPlayers );
273     }else{
274         // print number of points just made
275         sprintf( msg, "%d point/s move",
276                         game->players[ game->state.next ].board.points -
277                             game->players[ game->state.next ].lastBoard.points );
278         // if board is empty
279         if( game->players[ game->state.next ].board.emptySpots >=
280                                     game->options.width * game->options.height ){
281 
282             randFill( game, game->state.next, game->options.tokensPerTurn, true );
283             if( errorCode() != NOERROR ){
284                 sprintf( msg, "%s", errorMessage( errorCode() ) );
285                 return false;
286             }
287         }
288     }
289 
290     return true;
291 }
292 
293 
294 /**
295 * Saves current game to file.
296 *
297 * @param game       contains all information about current game
298 * @param argc       size of @a argv
299 * @param argv       parameters followinf the command
300 * @param[out] msg   output with message to write in the panel
301 *
302 * @return false if some message is to be written
303 *
304 * @see writeGame()
305 */
306 
307 bool
308 save( game_t * game, int argc, char ** argv, char * msg )
309 {
310     if( argc > 2 || strcmp( "save", argv[0] ) != 0 ){
311         sprintf( msg, "Wrong usage\n"
312         "Try 'save --help' for more information");
313         return false;
314     }
315     if( argc == 1 ){
```

```
316         sprintf( msg, "Missing file operand\n"
317                     "Try 'save --help' for more information");
318         return false;
319     }
320     if( strcmp( argv[1], "--help" ) == 0 ){
321         sprintf( msg, "Usage: save filename\n"
322                     "Saves the current game to file 'filename'");
323         return false;
324     }
325     writeGame( game, argv[1] );
326 
327     if( errorCode() != NOERROR ){
328         sprintf( msg, "%s", errorMessage( errorCode() ) );
329         return false;
330     }
331 
332     return true;
333 }
334 
335 
336 /**
337 * Undoes the last move made.
338 *
339 * @param game       contains all information about current game
340 * @param argc       size of @a argv
341 * @param argv       parameters followinf the command
342 * @param[out] msg   output with message to write in the panel
343 *
344 * @return false if some message is to be written
345 */
346 
347 bool
348 undo( game_t * game, int argc, char ** argv, char * msg )
349 {
350     if( argc == 2 && strcmp( argv[1], "--help" ) == 0 ){
351         sprintf( msg, "Usage: undo\n"
352                     "Undoes the last move\n"
353                     "It can only be used once some move has been done, "
354                     "and it can't be used two consecutive times" );
355         return false;
356     }
357     if( argc > 1  || strcmp( "undo", argv[0] ) != 0 ){
358         sprintf( msg, "Wrong usage\n"
359                     "Try 'undo --help' for more information");
360         return false;
361     }
362     if( game->options.mode == MULTIPLMODE ){
363         sprintf( msg, "'undo' command is only available in one player mode" );
364         return false;
365     }
366     if( game->players[ game->state.next ].canUndo == false ){
367         sprintf( msg, "'undo' command cannot be used twice in a row or in "
368                     "the first turn" );
369         return false;
370     }
371     game->players[ game->state.next ].canUndo = false;
372     // swap boards;
373     board_t aux = game->players[ game->state.next ].board;
374 
375     game->players[ game->state.next ].board =
376                                     game->players[ game->state.next ].lastBoard;
377 
378     game->players[ game->state.next ].lastBoard = aux;
```

```
379
380     return true;
381 }
382
383
384 /**
385 * Quits the game.
386 *
387 * @param game       contains all information about current game
388 * @param argc       size of @a argv
389 * @param argv       parameters followinf the command
390 * @param[out] msg   output with message to write in the panel
391 *
392 * @return false if some message is to be written
393 */
394
395 bool
396 quit( game_t * game, int argc, char ** argv, char * msg )
397 {
398     if( argc == 2 && strcmp( argv[1], "--help" ) == 0 ){
399         sprintf( msg, "Usage: quit\n"
400                       "Quits the current game" );
401         return false;
402     }
403     if( argc > 1 || strcmp( "quit", argv[0] ) != 0 ){
404         sprintf( msg, "Wrong usage\n"
405             "Try 'quit --help' for more information");
406         return false;
407     }
408     game->state.quit = true;
409     return true;
410 }
411
412
413 /**
414 * Prints help for the user. Secret commands are not shown.
415 *
416 * @param game       contains all information about current game
417 * @param argc       size of @a argv
418 * @param argv       parameters followinf the command
419 * @param[out] msg   output with message to write in the panel
420 *
421 * @return false if some message is to be written
422 */
423
424 bool
425 help( game_t * game, int argc, char ** argv, char * msg )
426 {
427     if( argc == 2 && strcmp( argv[1], "--help" ) == 0 ){
428         sprintf( msg, "Usage: help\n"
429                       "Shows available commands" );
430         return false;
431     }
432     if( argc > 1 || strcmp( "help", argv[0] ) != 0 ){
433         sprintf( msg, "Wrong usage\n"
434         "Try 'help --help' for more information");
435         return false;
436     }
437     sprintf( msg,
438                 "Type 'name --help' to find out more about the function 'name'"
439                 "\n \n"
440                 "  [row_1,column_1][row_2,column_2]\n"
441                 "  save filename\n"
```

```
442                 "  undo\n"
443                 "  quit\n"
444                 "  help\n" );
445     return true;
446 }
447 
448 
449 /**
450 * Secret command.
451 * Prints a ROFLcopter.
452 *
453 * @param game       contains all information about current game
454 * @param argc       size of @a argv
455 * @param argv       parameters followinf the command
456 * @param[out] msg   output with message to write in the panel
457 *
458 * @return false if some message is to be written
459 */
460 
461 bool
462 roflcopter( game_t * game, int argc, char ** argv, char * msg )
463 {
464     sprintf( msg,
465 #ifdef __unix__
466             "\033[5mROFL:ROFL\033[25m:LOL:\033[5mROFL:ROFL\033[25m\n"
467             "           |\n"
468             " \033[5mL\033[25m   /---------\n"
469             "\033[5mL\033[25mO\033[5mL\033[25m===       []\\\n"
470             " \033[5mL\033[25m    \\         \\\n"
471             "       \\_________\\\n"
472             "         |     |\n"
473             "      ------------/\n"
474 #else
475             ".............<ROFL ROFL ROFL ROFL>.\n"
476             "........................| |.........\n"
477             ".................. __\\||/___.......\n"
478             ".\\\\\..............|'-|-|.\\\\\...\\....\n"
479             "..\\ \\_...........|--|--|..\\\\\...\\...\n"
480             "../ L \\_______,/---------\\___\\___\\..\n"
481             ".|L O L|------------O------ ----,\\.\n"
482             "..\\ L /________,---''----------, /.\n"
483             "../ /...........\\_____________ ,/...\n"
484             ".//...........\\___//__/\\__\\\\__/...\n"
485 #endif
486             );
487 
488     return false;
489 }
490 
```