

sep 10, 19 15:20

sudoku_matrix_loader.h

Page 1/1

```

1 #ifndef _SUDOKU_MATRIX_LOADER_
2 #define _SUDOKU_MATRIX_LOADER_
3
4 int load_sudoku_matrix(int matrix[9][9]);
5
6 #endif

```

sep 10, 19 15:20

sudoku_matrix_loader.c

Page 1/1

```

1 #include "sudoku_matrix_loader.h"
2 #include <string.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5
6 #define ERROR 1
7 #define SUCCESS 0
8 #define MATRIX_FILE_PATH "board.txt"
9 #define BUFFER_SIZE 19
10
11 #define MATRIX_LOAD_ERROR_MSG "Error while reading matrix data\n"
12
13 int load_file_in_matrix(FILE* file, int matrix[9][9]);
14 void load_line_in_matrix(const char* buffer, int* matrix_row);
15 FILE* open_file();
16
17 int load_sudoku_matrix(int matrix[9][9]) {
18     FILE* matrix_file = open_file();
19     if (!matrix_file) {
20         return ERROR;
21     }
22     if (load_file_in_matrix(matrix_file, matrix) == ERROR) {
23         return 1;
24     }
25     return 0;
26 }
27
28 int load_file_in_matrix(FILE* file, int matrix[9][9]) {
29     char buffer[BUFFER_SIZE];
30     int i = 0;
31     while (fgets(buffer, BUFFER_SIZE, file)) {
32         load_line_in_matrix(buffer, matrix[i]);
33         i++;
34     }
35     fclose(file);
36     if (i < 8) {
37         fprintf(stderr, MATRIX_LOAD_ERROR_MSG);
38         return ERROR;
39     }
40     return SUCCESS;
41 }
42
43 FILE* open_file() {
44     FILE* matrix_file = fopen(MATRIX_FILE_PATH, "r");
45     if (!matrix_file) {
46         fprintf(stderr, MATRIX_LOAD_ERROR_MSG);
47         return NULL;
48     }
49     return matrix_file;
50 }
51
52 void load_line_in_matrix(const char* buffer, int* matrix_row) {
53     int i = 0;
54     for (int j = 0; j < 16; j += 2) {
55         matrix_row[i] = buffer[j] - '0'; //ascii code fix
56         i++;
57     }
58 }

```

sep 10, 19 15:20

sudoku_interface.h

Page 1/1

```

1  #ifndef _SUDOKU_INTERFACE_H_
2  #define _SUDOKU_INTERFACE_H_
3
4  #include "client_interface.h"
5  #include "server_interface.h"
6
7  typedef struct {
8      bool server_mode;
9      client_interface_t client_interface;
10     server_interface_t server_interface;
11 } sudoku_interface_t;
12
13 int sudoku_interface_init(sudoku_interface_t* sudoku_interface,
14     int argc, char* argv[]);
15
16 int sudoku_interface_execute(sudoku_interface_t* sudoku_interface);
17
18 #endif

```

sep 10, 19 15:20

sudoku_interface.c

Page 1/2

```

1  #include "sudoku_interface.h"
2  #include <stdbool.h>
3  #include "string.h"
4
5  #include <stdio.h>
6
7  #define SUCCESS 0
8  #define ERROR 1
9  #define EXIT 2
10
11 int check_arguments(int argc, char* argv[]);
12 int init_mode_executed(sudoku_interface_t* sudoku_interface, char* argv[]);
13 bool first_argument_unsupported(char* arg);
14
15 #define UNSUPPORTED_MODE_MSG "Modo no soportado, el primer parámetro "\
16     "debe ser server o client\n"
17
18 #define SV_MODE_WRONG_USE_MSG "Uso: ./tp server <puerto>\n"
19 #define CL_MODE_WRONG_USE_MSG "Uso: ./tp client <host> <puerto>\n"
20
21 int sudoku_interface_init(sudoku_interface_t* sudoku_interface,
22     int argc, char* argv[]){
23     if (check_arguments(argc, argv) == ERROR) {
24         return ERROR;
25     }
26     if (init_mode_executed(sudoku_interface, argv) == ERROR) {
27         return ERROR;
28     }
29     return SUCCESS;
30 }
31
32
33 int init_mode_executed(sudoku_interface_t* sudoku_interface, char* argv[]) {
34     if (strcmp(argv[1], "server") == 0) {
35         if (server_interface_init(&sudoku_interface->server_interface,
36             argv[2]) == ERROR) {
37             return ERROR;
38         }
39         sudoku_interface->server_mode = true;
40     } else {
41         if (client_interface_init(&sudoku_interface->client_interface,
42             argv[2], argv[3]) == ERROR) {
43             return ERROR;
44         }
45         sudoku_interface->server_mode = false;
46     }
47     return SUCCESS;
48 }
49
50
51 int sudoku_interface_execute(sudoku_interface_t* sudoku_interface) {
52     if (sudoku_interface->server_mode == true) {
53         if (server_interface_process(&sudoku_interface->server_interface) == ERROR) {
54             return ERROR;
55         }
56     } else {
57         int state = client_interface_process(&sudoku_interface->client_interface);
58         if (state == ERROR) {
59             return ERROR;
60         } else if (state == EXIT) {
61             return EXIT;
62         }
63     }
64     return SUCCESS;
65 }
66

```

sep 10, 19 15:20

sudoku_interface.c

Page 2/2

```

67 int check_arguments(int argc, char* argv[]) {
68     if (argc < 2 ∨ first_argument_unsupported(argv[1])) {
69         fprintf(stderr, UNSOPPORTED_MODE_MSG);
70         return ERROR;
71     }
72     if ((strcmp(argv[1], "server") == 0) ∧ (argc ≠ 3)) {
73         fprintf(stderr, SV_MODE_WRONG_USE_MSG);
74         return ERROR;
75     }
76     if ((strcmp(argv[1], "client") == 0) ∧ (argc ≠ 4)) {
77         fprintf(stderr, CL_MODE_WRONG_USE_MSG);
78         return ERROR;
79     }
80     return SUCCESS;
81 }
82
83 bool first_argument_unsupported(char* arg) {
84     int sv_cmp = strcmp(arg, "server");
85     int cl_cmp = strcmp(arg, "client");
86     if (sv_cmp ≠ 0 ∧ cl_cmp ≠ 0) {
87         return true;
88     }
89     return false;
90 }

```

sep 10, 19 15:20

sudoku_board.h

Page 1/1

```

1  #ifndef _SUDOKU_BOARD_H_
2  #define _SUDOKU_BOARD_H_
3
4  #include "cell.h"
5
6  typedef struct {
7      cell_t cells[9][9];
8  } sudoku_board_t;
9
10 void sudoku_board_init(sudoku_board_t* board,
11     int matrix[9][9]);
12
13 int sudoku_board_put(sudoku_board_t* board,
14     int number, int row, int column);
15
16 int sudoku_board_verify(sudoku_board_t* board);
17
18 void sudoku_board_restart(sudoku_board_t* board);
19
20 void sudoku_board_get(sudoku_board_t* board,
21     int matrix[9][9]);
22
23 #endif

```

sep 10, 19 15:20

sudoku_board.c

Page 1/2

```

1  #include "sudoku_board.h"
2  #include <stdbool.h>
3  #include <stdio.h>
4
5  bool row_is_valid(sudoku_board_t* board, int row_index);
6  bool column_is_valid(sudoku_board_t* board, int column_index);
7  bool sector_is_valid(sudoku_board_t* board, int sector_number);
8  void put_sector_cells_in_array(sudoku_board_t* board,
9      int sector_number, cell_t* array);
10
11 void sudoku_board_init(sudoku_board_t* board, int matrix[9][9]){
12     for (int i = 0; i < 9; i++) {
13         for (int j = 0; j < 9; j++) {
14             int numb = matrix[i][j];
15             if (numb == 0) {
16                 cell_init(&board->cells[i][j], numb, true);
17             } else {
18                 cell_init(&board->cells[i][j], numb, false);
19             }
20         }
21     }
22 }
23
24
25 int sudoku_board_put(sudoku_board_t* board, int number,
26     int row, int column) {
27     cell_t* cell = &board->cells[row - 1][column - 1];
28     if (cell_set_number(cell, number) == 1) {
29         return 1;
30     }
31     return 0;
32 }
33
34 int sudoku_board_verify(sudoku_board_t* board) {
35     int i;
36     for (i = 0; i < 9; i++) {
37         if (!row_is_valid(board, i) || !column_is_valid(board, i)) {
38             return 1;
39         }
40     }
41     for (i = 0; i < 3; i++) {
42         if (!sector_is_valid(board, i)) {
43             return 1;
44         }
45     }
46     return 0;
47 }
48
49 bool row_is_valid(sudoku_board_t* board, int row_index) {
50     int j;
51     for (int i = 0; i < 9; i++) {
52         for (j = i + 1; j < 9; j++) {
53             if (!cell_is_valid(&board->cells[row_index][i],
54                 &board->cells[row_index][j])) {
55                 return false;
56             }
57         }
58     }
59     return true;
60 }
61
62 bool column_is_valid(sudoku_board_t* board, int column_index) {
63     int j;
64     for (int i = 0; i < 9; i++) {
65         for (j = i + 1; j < 9; j++) {
66             if (!cell_is_valid(&board->cells[i][column_index],

```

sep 10, 19 15:20

sudoku_board.c

Page 2/2

```

67         &board->cells[j][column_index])) {
68             return false;
69         }
70     }
71 }
72 return true;
73 }
74
75 bool sector_is_valid(sudoku_board_t* board, int sector_number) {
76     int i, j;
77     cell_t aux[9];
78     put_sector_cells_in_array(board, sector_number, aux);
79     for (i = 0; i < 9; i++) {
80         for (j = i + 1; j < 9; j++) {
81             if (!cell_is_valid(&aux[i], &aux[j])) {
82                 return false;
83             }
84         }
85     }
86     return true;
87 }
88
89 void put_sector_cells_in_array(sudoku_board_t* board,
90     int sector_number, cell_t* array) {
91     int idx = sector_number * 3;
92     int final_idx = idx + 3;
93     int actual_pos = 0;
94     for (int i = idx; i < final_idx; i++) {
95         for (int j = idx; j < final_idx; j++) {
96             array[actual_pos] = board->cells[i][j];
97             actual_pos++;
98         }
99     }
100 }
101
102 void sudoku_board_restart(sudoku_board_t* board) {
103     for (int i = 0; i < 9; i++) {
104         for (int j = 0; j < 9; j++) {
105             cell_restart(&board->cells[i][j]);
106         }
107     }
108 }
109
110 void sudoku_board_get(sudoku_board_t* board, int matrix[9][9]) {
111     for (int i = 0; i < 9; i++) {
112         for (int j = 0; j < 9; j++) {
113             matrix[i][j] = cell_get_number(&board->cells[i][j]);
114         }
115     }
116 }

```

sep 10, 19 15:20

socket.h

Page 1/1

```

1  #ifndef _SOCKET_H_
2  #define _SOCKET_H_
3
4  typedef struct socket {
5      int fd;
6  } socket_t;
7
8  void socket_init(socket_t* socket);
9
10 void socket_release(socket_t* socket);
11
12 int socket_connect(socket_t* socket, const char* host, const char* service);
13
14 int socket_bind_and_listen(socket_t* socket, const char* service,
15     int listen_amount);
16
17 int socket_accept_client(socket_t* sv_skt, socket_t* peer_skt);
18
19 int socket_recv_message(socket_t* socket, char *buf, int size);
20
21 int socket_send_message(socket_t* socket, char *buf, int size);
22
23 #endif

```

sep 10, 19 15:20

socket.c

Page 1/3

```

1  #include "socket.h"
2  #define _POSIX_C_SOURCE 200112L
3
4  #include <sys/types.h>
5  #include <sys/socket.h>
6  #include <netdb.h>
7  #include <unistd.h>
8
9  #include <string.h>
10 #include <stdio.h>
11 #include <errno.h>
12 #include <stdbool.h>
13
14 //forward declarations
15 void socket_addr_iterate(socket_t* skt, struct addrinfo* result,
16     bool* connection_established);
17
18 int socket_getaddrinfo(struct addrinfo **result, const char* host,
19     const char* service, bool pasive);
20
21 int socket_bind(int sockfd, struct addrinfo* ptr);
22 int socket_listen(int sockfd, int listen_amount);
23
24
25 void socket_init(socket_t* socket) {
26     socket->fd = -1; //initialize to invalid fd
27 }
28
29 void socket_release(socket_t* skt) {
30     shutdown(skt->fd, SHUT_RDWR);
31     close(skt->fd);
32 }
33
34 int socket_connect(socket_t* skt, const char* host, const char* service) {
35     struct addrinfo *result = NULL;
36     int s = socket_getaddrinfo(&result, host, service, false);
37     if (s != 0) {
38         fprintf(stderr, "Error in getaddrinfo: %s\n", gai_strerror(s));
39         freeaddrinfo(result);
40         return 1;
41     }
42     bool connection_established = false;
43     socket_addr_iterate(skt, result, &connection_established);
44     if (!connection_established) {
45         fprintf(stderr, "Error: connection couldn't be established\n");
46         freeaddrinfo(result);
47         return 1;
48     }
49     freeaddrinfo(result);
50     return 0;
51 }
52
53 int socket_bind_and_listen(socket_t* skt, const char* service,
54     int listen_amount) {
55     struct addrinfo *ptr = NULL;
56     int s = socket_getaddrinfo(&ptr, NULL, service, true);
57     if (s != 0) {
58         fprintf(stderr, "Error in getaddrinfo: %s\n", gai_strerror(s));
59         return 1;
60     }
61     skt->fd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
62     if (skt->fd == -1) {
63         fprintf(stderr, "Error: %s\n", strerror(errno));
64         freeaddrinfo(ptr);
65         return 1;
66     }

```

sep 10, 19 15:20

socket.c

Page 2/3

```

67  if (socket_bind(skt->fd, ptr) == 1) {
68      freeaddrinfo(ptr);
69      return 1;
70  }
71  freeaddrinfo(ptr);
72  if (socket_listen(skt->fd, listen_amount) == 1) {
73      return 1;
74  }
75  return 0;
76  }
77
78  int socket_accept_client(socket_t* sv_skt, socket_t* peer_skt) {
79      peer_skt->fd = accept(sv_skt->fd, NULL, NULL);
80      if (peer_skt->fd == -1) {
81          fprintf(stderr, "Error: %s\n", strerror(errno));
82          return 1;
83      }
84      return 0;
85  }
86
87  int socket_recv_message(socket_t* skt, char *buf, int size){
88      int received = 0;
89      int s = 0;
90      while (received < size) {
91          s = recv(skt->fd, &buf[received], size-received, MSG_NOSIGNAL);
92          if (s == 0 || s == -1) { //socket was closed or error occurred
93              return -1;
94          } else {
95              received += s;
96          }
97      }
98      return received;
99  }
100
101  int socket_send_message(socket_t* skt, char *buf, int size){
102      int sent = 0;
103      int s = 0;
104      while (sent < size) {
105          s = send(skt->fd, &buf[sent], size-sent, MSG_NOSIGNAL);
106          if (s == 0 || s == -1) { //socket was closed or error occurred
107              return -1;
108          } else {
109              sent += s;
110          }
111      }
112      return sent;
113  }
114
115  void socket_addr_iterate(socket_t* skt, struct addrinfo* result,
116      bool* connection_established) {
117      struct addrinfo* ptr;
118      int s;
119      for (ptr = result; ptr != NULL ^ *connection_established == false;
120          ptr = ptr->ai_next) {
121          skt->fd = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
122          if (skt->fd == -1) {
123              fprintf(stderr, "Error: %s\n", strerror(errno));
124          } else {
125              s = connect(skt->fd, ptr->ai_addr, ptr->ai_addrlen);
126              if (s == -1) {
127                  fprintf(stderr, "Error: %s\n", strerror(errno));
128                  close(skt->fd);
129              }
130              *connection_established = (s != -1); //are we connected now?
131          }
132      }

```

sep 10, 19 15:20

socket.c

Page 3/3

```

133  }
134
135  int socket_getaddrinfo(struct addrinfo **addrinfo_ptr, const char* host,
136      const char* service, bool passive) {
137      struct addrinfo hints;
138      memset(&hints, 0, sizeof(struct addrinfo));
139      hints.ai_family = AF_INET;
140      hints.ai_socktype = SOCK_STREAM;
141
142      if (passive){
143          hints.ai_flags = AI_PASSIVE;
144          return getaddrinfo(NULL, service, &hints, addrinfo_ptr);
145      } else {
146          hints.ai_flags = 0;
147          return getaddrinfo(host, service, &hints, addrinfo_ptr);
148      }
149  }
150
151  int socket_bind(int sockfd, struct addrinfo* ptr) {
152      int s = bind(sockfd, ptr->ai_addr, ptr->ai_addrlen);
153      if (s == -1) {
154          fprintf(stderr, "Error: %s\n", strerror(errno));
155          close(sockfd);
156          return 1;
157      }
158      return 0;
159  }
160
161  int socket_listen(int sockfd, int listen_amount) {
162      int s = listen(sockfd, listen_amount);
163      if (s == -1) {
164          fprintf(stderr, "Error: %s\n", strerror(errno));
165          close(sockfd);
166          return 1;
167      }
168      return 0;
169  }

```

sep 10, 19 15:20

server_socket.h

Page 1/1

```

1  #ifndef _SERVER_H_
2  #define _SERVER_H_
3
4  #include "socket.h"
5
6  typedef struct server {
7      socket_t sv_skt;
8      socket_t peer_skt;
9  } server_socket_t;
10
11 int server_socket_init(server_socket_t* self, const char* port);
12
13 int server_socket_accept_client(server_socket_t* self);
14
15 int server_socket_recv_message(server_socket_t* self, char* buf, int size);
16
17 int server_socket_send_message(server_socket_t* self, char* buf, int size);
18
19 void server_socket_release(server_socket_t* self);
20
21 #endif

```

sep 10, 19 15:20

server_socket.c

Page 1/1

```

1  #include "server_socket.h"
2  #define BACKLOG 1
3
4  int server_socket_init(server_socket_t* self, const char* port) {
5      socket_init(&self->sv_skt);
6      socket_init(&self->peer_skt);
7      return socket_bind_and_listen(&self->sv_skt, port, BACKLOG);
8  }
9
10 int server_socket_accept_client(server_socket_t* self) {
11     return socket_accept_client(&self->sv_skt, &self->peer_skt);
12 }
13
14 int server_socket_recv_message(server_socket_t* self, char* buf, int size) {
15     return (socket_recv_message(&self->peer_skt, buf, size) == -1);
16 }
17
18 int server_socket_send_message(server_socket_t* self, char* buf, int size) {
19     return (socket_send_message(&self->peer_skt, buf, size) == -1);
20 }
21
22 void server_socket_release(server_socket_t* self) {
23     socket_release(&self->sv_skt);
24     socket_release(&self->peer_skt);
25 }

```

sep 10, 19 15:20

server_protocol.h

Page 1/1

```

1  #ifndef _SERVER_PROTOCOL_H
2  #define _SERVER_PROTOCOL_H
3
4  #include "sudoku_board.h"
5  #include "server_socket.h"
6
7  typedef struct {
8      server_socket_t* skt;
9      sudoku_board_t* board;
10 } server_protocol_t;
11
12 void server_protocol_init(server_protocol_t* protocol,
13     server_socket_t* skt, sudoku_board_t* board);
14
15 int server_protocol_process(server_protocol_t* protocol);
16
17 #endif

```

sep 10, 19 15:20

server_protocol.c

Page 1/3

```

1  #include "server_protocol.h"
2  #include <string.h>
3  #include <stdint.h>
4  #include <arpa/inet.h>
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include "board_representation_maker.h"
8
9  #define PUT_ROW_BUFFER_IDX 0
10 #define PUT_COL_BUFFER_IDX 1
11 #define PUT_NUMB_BUFFER_IDX 2
12
13 #define PUT_COMMAND_MES 'P'
14 #define VERIFY_COMMAND_MES 'V'
15 #define RESET_COMMAND_MES 'R'
16 #define GET_COMMAND_MES 'G'
17
18 #define VALID_BOARD_MSG "OK\n"
19 #define INVALID_BOARD_MSG "ERROR\n"
20 #define UNMODIFIABLE_CELL_MSG "La celda indicada no es modificable\nâM-^@M-^K"
21
22 //forward declarations
23
24 uint32_t calculate_str_len(char* str);
25
26 int process_message(server_protocol_t* protocol, char message);
27
28 int process_p_message(server_protocol_t* protocol);
29 int process_g_message(server_protocol_t* protocol);
30 int process_r_message(server_protocol_t* protocol);
31 int process_v_message(server_protocol_t* protocol);
32
33 int show_board_to_client(server_protocol_t* protocol);
34
35 int send_message_to_client(server_protocol_t* protocol, char* mes);
36 int send_unmodifiable_cell_message(server_protocol_t* protocol);
37 int send_modifiable_cell_message(server_protocol_t* protocol);
38 int send_invalid_board_message(server_protocol_t* protocol);
39 int send_valid_board_message(server_protocol_t* protocol);
40
41 void server_protocol_init(server_protocol_t* protocol,
42     server_socket_t* skt, sudoku_board_t* board) {
43     protocol->skt = skt;
44     protocol->board = board;
45 }
46
47 int server_protocol_process(server_protocol_t* protocol) {
48     char buffer;
49     if (server_socket_rcv_message(protocol->skt, &buffer, 1)) {
50         return 1;
51     }
52     if (process_message(protocol, buffer)) {
53         return 1;
54     }
55     return 0;
56 }
57
58 int process_message(server_protocol_t* protocol, char message) {
59     if (message == PUT_COMMAND_MES) {
60         return process_p_message(protocol);
61     }
62     else if (message == GET_COMMAND_MES) {
63         return process_g_message(protocol);
64     }
65     else if (message == RESET_COMMAND_MES) {
66         return process_r_message(protocol);

```


sep 10, 19 15:20

server_protocol.c

Page 2/3

```

67     } else {
68     }
69     return process_v_message(protocol);
70 }
71 }
72 }
73 }
74 int process_p_message(server_protocol_t* protocol) {
75     char buffer[3];
76     if (server_socket_recv_message(protocol->skt, buffer, 3)) {
77         return 1;
78     }
79     int numb = buffer[PUT_NUMB_BUFFER_IDX] - '0';
80     int row = buffer[PUT_ROW_BUFFER_IDX] - '0';
81     int col = buffer[PUT_COL_BUFFER_IDX] - '0';
82
83     int unmodifiable_cell = sudoku_board_put(protocol->board, numb, row, col);
84     if (unmodifiable_cell) {
85         return send_unmodifiable_cell_message(protocol);
86     } else {
87         return send_modifiable_cell_message(protocol);
88     }
89 }
90
91 int send_unmodifiable_cell_message(server_protocol_t* protocol) {
92     if (send_message_to_client(protocol, UNMODIFIABLE_CELL_MSG) == 1) {
93         return 1;
94     }
95     return 0;
96 }
97
98 int send_modifiable_cell_message(server_protocol_t* protocol) {
99     return show_board_to_client(protocol);
100 }
101
102 int process_g_message(server_protocol_t* protocol) {
103     return show_board_to_client(protocol);
104 }
105
106 int process_r_message(server_protocol_t* protocol) {
107     sudoku_board_restart(protocol->board);
108     return show_board_to_client(protocol);
109 }
110
111 int process_v_message(server_protocol_t* protocol) {
112     if (sudoku_board_verify(protocol->board)) {
113         return send_invalid_board_message(protocol);
114     }
115     return send_valid_board_message(protocol);
116 }
117
118 int send_invalid_board_message(server_protocol_t* protocol) {
119     if (send_message_to_client(protocol, INVALID_BOARD_MSG) == 1) {
120         return 1;
121     }
122     return 0;
123 }
124
125 int send_valid_board_message(server_protocol_t* protocol) {
126     if (send_message_to_client(protocol, VALID_BOARD_MSG) == 1) {
127         return 1;
128     }
129     return 0;
130 }
131
132 int show_board_to_client(server_protocol_t* protocol) {

```

sep 10, 19 15:20

server_protocol.c

Page 3/3

```

133     int matrix[9][9];
134     sudoku_board_get(protocol->board, matrix);
135     char* board_representation = assemble_board_representation(matrix);
136     int error = send_message_to_client(protocol, board_representation);
137     free(board_representation);
138     return error;
139 }
140
141 int send_message_to_client(server_protocol_t* protocol, char* msg) {
142     uint32_t len = calculate_str_len(msg);
143     uint32_t len_ton = htonl(len);
144     if (server_socket_send_message(protocol->skt, (char*)&len_ton, 4)) {
145         return 1;
146     }
147     for (int i = 0; i < len; i++) {
148         if (server_socket_send_message(protocol->skt, &msg[i], 1)) {
149             return 1;
150         }
151     }
152     return 0;
153 }
154
155 uint32_t calculate_str_len(char* str) {
156     uint32_t i = 0;
157     while (str[i] != '\0') {
158         i++;
159     }
160     return i;
161 }

```

sep 10, 19 15:20

server_interface.h

Page 1/1

```

1  #ifndef _SERVER_INTERFACE_H_
2  #define _SERVER_INTERFACE_H_
3
4  #include "server_protocol.h"
5  #include "server_socket.h"
6  #include "sudoku_board.h"
7
8  typedef struct {
9      sudoku_board_t board;
10     server_socket_t skt;
11     server_protocol_t protocol;
12 } server_interface_t;
13
14 int server_interface_init(server_interface_t* self, const char* port);
15
16 int server_interface_process(server_interface_t* self);
17
18 #endif

```

sep 10, 19 15:20

server_interface.c

Page 1/1

```

1  #include "server_interface.h"
2  #include "sudoku_matrix_loader.h"
3
4  #define ERROR 1
5  #define SUCCESS 0
6
7  int server_interface_init(server_interface_t* self, const char* port) {
8      int matrix[9][9];
9      if (load_sudoku_matrix(matrix) == ERROR) {
10         return ERROR;
11     }
12     sudoku_board_init(&self->board, matrix);
13     if (server_socket_init(&self->skt, port) == ERROR) {
14         return ERROR;
15     }
16     if (server_socket_accept_client(&self->skt) == ERROR) {
17         server_socket_release(&self->skt);
18         return ERROR;
19     }
20     server_protocol_init(&self->protocol, &self->skt, &self->board);
21     return SUCCESS;
22 }
23
24 int server_interface_process(server_interface_t* self) {
25     if (server_protocol_process(&self->protocol)) {
26         server_socket_release(&self->skt);
27         return ERROR;
28     }
29     return SUCCESS;
30 }

```

sep 10, 19 15:20

protocol_message_maker.h

Page 1/1

```

1 #ifndef _PROTOCOL_MESSAGE_MAKER_H_
2 #define _PROTOCOL_MESSAGE_MAKER_H_
3
4 char* build_protocol_message(char* command);
5
6 #endif

```

sep 10, 19 15:20

protocol_message_maker.c

Page 1/2

```

1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdbool.h>
4 #include "protocol_message_maker.h"
5
6 #define RESET_COMMAND_MES 'R'
7 #define PUT_COMMAND_MES 'P'
8 #define VERIFY_COMMAND_MES 'V'
9 #define GET_COMMAND_MES 'G'
10
11 #define GET_COMMAND "get"
12 #define RESET_COMMAND "reset"
13 #define VERIFY_COMMAND "verify"
14 #define PUT_COMMAND "put"
15
16 #define PUT_COMMAND_NUMB_POS 4
17 #define PUT_COMMAND_IDX_A_POS 9
18 #define PUT_COMMAND_IDX_B_POS 11
19
20 char* build_get_message();
21 char* build_reset_message();
22 char* build_verify_message();
23 char* build_put_message(char* command);
24 char* init_message(int len);
25
26 char* build_protocol_message(char* command) {
27     if (strcmp(GET_COMMAND, command) == 0) {
28         return build_get_message(command);
29     }
30     else if (strcmp(RESET_COMMAND, command) == 0) {
31         return build_reset_message(command);
32     }
33     else if (strcmp(VERIFY_COMMAND, command) == 0) {
34         return build_verify_message(command);
35     }
36     else {
37         return build_put_message(command);
38     }
39 }
40
41 char* build_get_message() {
42     char* message = init_message(1);
43     message[0] = GET_COMMAND_MES;
44     return message;
45 }
46
47 char* build_reset_message() {
48     char* message = init_message(1);
49     message[0] = RESET_COMMAND_MES;
50     return message;
51 }
52
53 char* build_verify_message() {
54     char* message = init_message(1);
55     message[0] = VERIFY_COMMAND_MES;
56     return message;
57 }
58
59 char* build_put_message(char* command) { //command format: "put x in y,z"
60     char* message = init_message(4);
61     message[0] = PUT_COMMAND_MES;
62     message[1] = command[PUT_COMMAND_IDX_A_POS];
63     message[2] = command[PUT_COMMAND_IDX_B_POS];
64     message[3] = command[PUT_COMMAND_NUMB_POS];
65     return message;
66 }

```

sep 10, 19 15:20

protocol_message_maker.c

Page 2/2

```

67
68 char* init_message(int len) {
69     char* mes = malloc(sizeof(char)*len + 1);
70     mes[len] = '\0';
71     return mes;
72 }

```

sep 10, 19 15:20

main.c

Page 1/1

```

1  #include "sudoku_interface.h"
2
3  #define SUCCESS 0
4  #define ERROR 1
5  #define EXIT 2
6
7  int main(int argc, char* argv[]){
8      sudoku_interface_t sudoku_interface;
9
10     int game_init = sudoku_interface_init(&sudoku_interface, argc, argv);
11     if (game_init == ERROR) {
12         return 1;
13     }
14
15     int execution;
16     while(1) {
17         execution = sudoku_interface_execute(&sudoku_interface);
18         if (execution == EXIT) {
19             return SUCCESS;
20         } else if (execution == ERROR) {
21             return ERROR;
22         }
23     }
24     return 0;
25 }

```

sep 10, 19 15:20

client_socket.h

Page 1/1

```

1  #ifndef _CLIENT_SOCKET_H_
2  #define _CLIENT_SOCKET_H_
3
4  #include "socket.h"
5
6  typedef struct client {
7      socket_t skt;
8  } client_socket_t;
9
10 int client_socket_init(client_socket_t* self, const char* host,
11     const char* service);
12
13 int client_socket_rcv_message(client_socket_t* self, char* buf, int size);
14
15 int client_socket_snd_message(client_socket_t* self, char* buf, int size);
16
17 void client_socket_release(client_socket_t* self);
18
19 #endif

```

sep 10, 19 15:20

client_socket.c

Page 1/1

```

1  #include "client_socket.h"
2
3  int client_socket_init(client_socket_t* self, const char* host,
4      const char* service) {
5      socket_init(&self->skt);
6      return socket_connect(&self->skt, host, service);
7  }
8
9  int client_socket_rcv_message(client_socket_t* self, char* buf, int size) {
10     return (socket_rcv_message(&self->skt, buf, size) == -1);
11 }
12
13 int client_socket_snd_message(client_socket_t* self, char* buf, int size) {
14     return (socket_snd_message(&self->skt, buf, size) == -1);
15 }
16
17 void client_socket_release(client_socket_t* self) {
18     socket_release(&self->skt);
19 }

```

sep 10, 19 15:20

client_protocol.h

Page 1/1

```

1  #ifndef _CLIENT_PROTOCOL_H_
2  #define _CLIENT_PROTOCOL_H_
3
4  #include "client_socket.h"
5
6  typedef struct {
7      client_socket_t* skt;
8  } client_protocol_t;
9
10 void client_protocol_init(client_protocol_t* protocol, client_socket_t* skt);
11
12 int client_protocol_send_message(client_protocol_t* protocol, char* command);
13
14 char* client_protocol_recv_answer(client_protocol_t* protocol);
15
16 #endif

```

sep 10, 19 15:20

client_protocol.c

Page 1/1

```

1  #include "client_protocol.h"
2  #include "protocol_message_maker.h"
3  #include <stdio.h>
4  #include <arpa/inet.h>
5  #include "string.h"
6  #include <stdint.h>
7  #include <stdlib.h>
8  #include <stdbool.h>
9
10 void client_protocol_init(client_protocol_t* protocol, client_socket_t* skt) {
11     protocol->skt = skt;
12 }
13
14 char* client_protocol_recv_answer(client_protocol_t* protocol) {
15     char len_buffer[4];
16     if (client_socket_recv_message(protocol->skt, len_buffer, 4)) {
17         return NULL;
18     }
19     uint32_t text_len;
20     memcpy(&text_len, len_buffer, 4);
21     text_len = ntohl(text_len);
22     char* text_buffer = malloc(sizeof(char)*text_len + 1);
23     text_buffer[text_len] = '\0';
24
25     if (client_socket_recv_message(protocol->skt, text_buffer, text_len)) {
26         free(text_buffer);
27         return NULL;
28     }
29     return text_buffer;
30 }
31
32 int client_protocol_send_message(client_protocol_t* protocol, char* command) {
33     char* message = build_protocol_message(command);
34     if (client_socket_send_message(protocol->skt, message, strlen(message))) {
35         free(message);
36         return 1;
37     }
38     free(message);
39     return 0;
40 }

```

sep 10, 19 15:20

client_interface.h

Page 1/1

```

1  #ifndef _CLIENT_INTERFACE_H
2  #define _CLIENT_INTERFACE_H
3
4  #include "client_socket.h"
5  #include "client_protocol.h"
6
7  typedef struct {
8      client_socket_t skt;
9      client_protocol_t protocol;
10 } client_interface_t;
11
12 int client_interface_init(client_interface_t* client_interface,
13     const char* host, const char* service);
14
15 int client_interface_process(client_interface_t* client_interface);
16
17 #endif

```

sep 10, 19 15:20

client_interface.c

Page 1/3

```

1  #include "client_interface.h"
2  #include "client_protocol.h"
3  #include <string.h>
4  #include <stdbool.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  #define EXIT_COMMAND "exit"
9  #define PUT_COMMAND "put"
10
11 #define FGSETS_SIZE 25
12 #define COMMAND_MAX_SIZE 15
13
14 #define SUCCESS 0
15 #define ERROR 1
16 #define EXIT 2
17 #define INVALID_COMMAND 3
18
19 #define INDEX_ERROR_MES "Error en los Ã-ndices. Rango soportado: [1,9]"
20 #define VALUE_ERROR_MES "Error en el valor ingresado. Rango soportado: [1,9]"
21
22 int process_user_input(char* input);
23 int execute_command(client_protocol_t* protocol, char* command);
24 bool command_has_valid_indexes(char* input);
25 bool command_has_valid_values(char* input);
26 void get_command_first_arg(char* buffer, char* input);
27 bool index_is_allowed(char* index);
28 bool value_is_allowed(char* value);
29
30 int client_interface_init(client_interface_t* client_interface,
31     const char* host, const char* service) {
32     if (client_socket_init(&client_interface->skt, host, service)) {
33         return ERROR;
34     }
35     client_protocol_init(&client_interface->protocol, &client_interface->skt);
36     return SUCCESS;
37 }
38
39 int client_interface_process(client_interface_t* client_interface) {
40     char buffer[FGSETS_SIZE];
41     int input_state = process_user_input(buffer);
42     if (input_state == EOF) {
43         client_socket_release(&client_interface->skt);
44         return ERROR;
45     }
46     if (input_state == INVALID_COMMAND) {
47         return SUCCESS;
48     }
49     if (strcmp(buffer, EXIT_COMMAND) == 0) {
50         client_socket_release(&client_interface->skt);
51         return EXIT;
52     }
53     if (execute_command(&client_interface->protocol, buffer)) {
54         client_socket_release(&client_interface->skt);
55         return ERROR;
56     }
57     return SUCCESS;
58 }
59
60 int execute_command(client_protocol_t* protocol, char* command) {
61     int command_exec = client_protocol_send_message(protocol, command);
62     if (command_exec == ERROR) {
63         return ERROR;
64     }
65     char* sv_ans = client_protocol_rcv_answer(protocol);
66     if (!sv_ans) {

```

sep 10, 19 15:20

client_interface.c

Page 2/3

```

67     return ERROR;
68 }
69 printf("%s", sv_ans);
70 free(sv_ans);
71 return SUCCESS;
72 }
73
74 int process_user_input(char* input) {
75     if (!fgets(input, FGETS_SIZE, stdin)) {
76         return EOF;
77     }
78     input[strlen(input) - 1] = '\0'; //replace '\n' with '\0'
79     if (!command_has_valid_indexes(input)) {
80         fprintf(stderr, "%s\n", INDEX_ERROR_MES);
81         return INVALID_COMMAND;
82     } else if (!command_has_valid_values(input)) {
83         fprintf(stderr, "%s\n", VALUE_ERROR_MES);
84         return INVALID_COMMAND;
85     }
86     return SUCCESS;
87 }
88
89 bool command_has_valid_indexes(char* input) {
90     char command_first_arg[COMMAND_MAX_SIZE];
91     get_command_first_arg(command_first_arg, input);
92     if (strcmp(command_first_arg, PUT_COMMAND) == 0) {
93         char index_a[4], index_b[4];
94         index_a[3] = '\0';
95         index_b[3] = '\0';
96         sscanf(input, "%s%s%s%s%3[^\n]%3s", index_a, index_b);
97         if (!index_is_allowed(index_a) || !index_is_allowed(index_b)) {
98             return false;
99         }
100     }
101     return true;
102 }
103
104 bool command_has_valid_values(char* input) {
105     char command_first_arg[COMMAND_MAX_SIZE];
106     get_command_first_arg(command_first_arg, input);
107     if (strcmp(command_first_arg, PUT_COMMAND) == 0) {
108         char value[4];
109         value[3] = '\0';
110         sscanf(input, "%s%3s", value);
111         if (!value_is_allowed(value)) {
112             return false;
113         }
114     }
115     return true;
116 }
117
118 void get_command_first_arg(char* buffer, char* input) {
119     sscanf(input, "%14s", buffer);
120     buffer[COMMAND_MAX_SIZE - 1] = '\0';
121 }
122
123 bool index_is_allowed(char* index) {
124     int idx = atoi(index);
125     if (idx > 9 || idx < 1) {
126         return false;
127     }
128     return true;
129 }
130
131 bool value_is_allowed(char* value) {
132     int v = atoi(value);

```

sep 10, 19 15:20

client_interface.c

Page 3/3

```

133     if (v > 9 || v < 1) {
134         return false;
135     }
136     return true;
137 }

```


sep 10, 19 15:20

cell.h

Page 1/1

```

1  #ifndef _CELL_H_
2  #define _CELL_H_
3
4  #include <stdbool.h>
5
6  typedef struct cell {
7      bool modifiable;
8      int number;
9  } cell_t;
10
11 void cell_init(cell_t* cell, int number, bool modifiable);
12
13 int cell_set_number(cell_t* cell, int numb);
14
15 int cell_get_number(cell_t* cell);
16
17 bool cell_is_valid(cell_t* cell_a, cell_t* cell_b);
18
19 void cell_restart(cell_t* cell);
20
21 #endif

```

sep 10, 19 15:20

cell.c

Page 1/1

```

1  #include "cell.h"
2
3  void cell_init(cell_t* cell, int number, bool modifiable) {
4      cell->modifiable = modifiable;
5      cell->number = number;
6  }
7
8  int cell_set_number(cell_t* cell, int numb) {
9      if (!cell->modifiable) {
10         return 1;
11     }
12     cell->number = numb;
13     return 0;
14 }
15
16 int cell_get_number(cell_t* cell) {
17     return cell->number;
18 }
19
20 bool cell_is_valid(cell_t* cell_a, cell_t* cell_b) {
21     return (cell_a->number != cell_b->number || cell_a->number == 0);
22 }
23
24
25 void cell_restart(cell_t* cell) {
26     if (cell->modifiable) {
27         cell->number = 0;
28     }
29 }

```

sep 10, 19 15:20

board_representation_maker.h

Page 1/1

```

1  #ifndef _BOARD_REPRESENTATION_MAKER_
2  #define _BOARD_REPRESENTATION_MAKER_
3
4  char* assemble_board_representation(int matrix[9][9]);
5
6  #endif

```

sep 10, 19 15:20

board_representation_maker.c

Page 1/2

```

1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #define BOARD_REPR_SIZE 723
5  #define BOARD_REPR_ROWS 19
6
7  void add_sector_separator(char* buffer);
8  void add_normal_separator(char* buffer);
9  void add_matrix_row(char* buffer, int matrix[9][9], int row_index);
10 void iterate_row_representation(char* row_representation, int* row, int len);
11
12 char* assemble_board_representation(int matrix[9][9]) {
13     char* board_representation = malloc(sizeof(char)*BOARD_REPR_SIZE);
14     int matrix_row_index = 0;
15     for (int i = 0; i < BOARD_REPR_ROWS; i++) {
16         int actual_index = i*38;
17         if (i % 6 == 0) {
18             add_sector_separator(&board_representation[actual_index]);
19         } else if (i % 2 == 0) {
20             add_normal_separator(&board_representation[actual_index]);
21         } else {
22             add_matrix_row(&board_representation[actual_index],
23                           matrix, matrix_row_index);
24             matrix_row_index += 1;
25         }
26     }
27     board_representation[BOARD_REPR_SIZE-1] = '\0';
28     return board_representation;
29 }
30
31 void add_sector_separator(char* buffer) {
32     char sector_separator[] = "U=====U=====U\n";
33     size_t len = strlen(sector_separator);
34     for (int i = 0; i < len; i++) {
35         buffer[i] = sector_separator[i];
36     }
37 }
38
39 void add_normal_separator(char* buffer) {
40     char normal_separator[] = "U-----U-----U\n";
41     size_t len = strlen(normal_separator);
42     for (int i = 0; i < len; i++) {
43         buffer[i] = normal_separator[i];
44     }
45 }
46
47 void add_matrix_row(char* buffer, int matrix[9][9], int row_index) {
48     char row_representation[] = "U || U || U || U\n";
49     size_t len = strlen(row_representation);
50     iterate_row_representation(row_representation, matrix[row_index], len);
51     for (int i = 0; i < len; i++) {
52         buffer[i] = row_representation[i];
53     }
54 }
55
56 void iterate_row_representation(char* row_representation, int* row, int len) {
57     int i = 2;
58     int j = 0;
59     char aux[2];
60     while (i < len) {
61         if (row[j] != 0) {
62             snprintf(aux, sizeof(aux)/sizeof(char), "%d", row[j]);
63             row_representation[i] = aux[0];
64         }
65         i += 4;
66     }

```

sep 10, 19 15:20

board_representation_maker.c

Page 2/2

```
67         j++;
68     }
69 }
```

sep 10, 19 15:20

Table of Content

Page 1/1

1	Table of Contents				
2	1	sudoku_matrix_loader.h	sheets	1 to 1 (1)	pages 1- 1 7 lines
3	2	sudoku_matrix_loader.c	sheets	1 to 1 (1)	pages 2- 2 59 lines
4	3	sudoku_interface.h..	sheets	2 to 2 (1)	pages 3- 3 19 lines
5	4	sudoku_interface.c..	sheets	2 to 3 (2)	pages 4- 5 91 lines
6	5	sudoku_board.h.....	sheets	3 to 3 (1)	pages 6- 6 24 lines
7	6	sudoku_board.c.....	sheets	4 to 4 (1)	pages 7- 8 117 lines
8	7	socket.h.....	sheets	5 to 5 (1)	pages 9- 9 24 lines
9	8	socket.c.....	sheets	5 to 6 (2)	pages 10- 12 170 lines
10	9	server_socket.h.....	sheets	7 to 7 (1)	pages 13- 13 22 lines
11	10	server_socket.c.....	sheets	7 to 7 (1)	pages 14- 14 26 lines
12	11	server_protocol.h...	sheets	8 to 8 (1)	pages 15- 15 18 lines
13	12	server_protocol.c...	sheets	8 to 9 (2)	pages 16- 18 162 lines
14	13	server_interface.h..	sheets	10 to 10 (1)	pages 19- 19 19 lines
15	14	server_interface.c..	sheets	10 to 10 (1)	pages 20- 20 31 lines
16	15	protocol_message_maker.h	sheets	11 to 11 (1)	pages 21- 21 7 lines
17	16	protocol_message_maker.c	sheets	11 to 12 (2)	pages 22- 23 73 lines
18	17	main.c.....	sheets	12 to 12 (1)	pages 24- 24 26 lines
19	18	client_socket.h.....	sheets	13 to 13 (1)	pages 25- 25 20 lines
20	19	client_socket.c.....	sheets	13 to 13 (1)	pages 26- 26 20 lines
21	20	client_protocol.h...	sheets	14 to 14 (1)	pages 27- 27 17 lines
22	21	client_protocol.c...	sheets	14 to 14 (1)	pages 28- 28 41 lines
23	22	client_interface.h..	sheets	15 to 15 (1)	pages 29- 29 18 lines
24	23	client_interface.c..	sheets	15 to 16 (2)	pages 30- 32 138 lines
25	24	cell.h.....	sheets	17 to 17 (1)	pages 33- 33 22 lines
26	25	cell.c.....	sheets	17 to 17 (1)	pages 34- 34 30 lines
27	26	board_representation_maker.h	sheets	18 to 18 (1)	pages 35- 35 7 lines
28	27	board_representation_maker.c	sheets	18 to 19 (2)	pages 36- 37 70 lines