Sudoku

Generated by Doxygen 1.13.2

1 File Index 1

1 File Index	1
1.1 File List	. 1
2 File Documentation	2
2.1 sudoku.c File Reference	. 2
2.1.1 Macro Definition Documentation	. 2
2.1.2 Function Documentation	. 3
2.2 sudoku.c	. 3
2.3 sudoku_ncurses.c File Reference	. 4
2.3.1 Macro Definition Documentation	. 4
2.3.2 Function Documentation	. 5
2.4 sudoku_ncurses.c	. 16
2.5 sudoku_ncurses.h File Reference	. 20
2.5.1 Function Documentation	. 20
2.5.2 Variable Documentation	. 32
2.6 sudoku_ncurses.h	. 32
2.7 sudoku_solve.c File Reference	. 32
2.7.1 Function Documentation	. 33
2.7.2 Variable Documentation	. 41
2.8 sudoku_solve.c	. 41
2.9 sudoku_solve.h File Reference	. 43
2.9.1 Macro Definition Documentation	. 44
2.9.2 Function Documentation	. 44
2.10 sudoku_solve.h	. 53
2.11 sudoku_user.c File Reference	. 53
2.11.1 Function Documentation	. 53
2.12 sudoku_user.c	. 57
2.13 sudoku_user.h File Reference	. 58
2.13.1 Macro Definition Documentation	. 59
2.13.2 Function Documentation	. 60
2.13.3 Variable Documentation	. 64
2.14 sudoku_user.h	. 65
Index	67
	σ.
1 File Index	
The maex	
1.1 File List	
Here is a list of all files with brief descriptions:	
sudoku.c	2
sudoku_ncurses.c	4

sudoku_ncurses.h	20
sudoku_solve.c	32
sudoku_solve.h	43
sudoku_user.c	53
sudoku_user.h	58

2 File Documentation

2.1 sudoku.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ncurses.h>
#include "sudoku_ncurses.h"
#include "sudoku_solve.h"
#include "sudoku_user.h"
```

Macros

• #define DEBUG

Functions

• int main (void)

2.1.1 Macro Definition Documentation

DEBUG

#define DEBUG

Definition at line 10 of file sudoku.c.

2.2 sudoku.c 3

2.1.2 Function Documentation

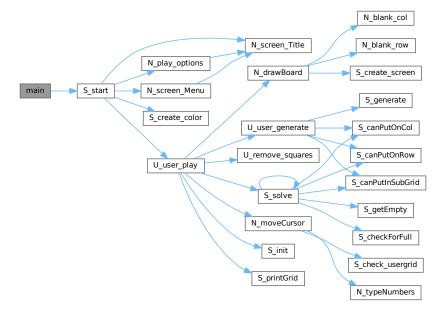
main()

```
int main ( $\operatorname{\text{void}}$)
```

Definition at line 12 of file sudoku.c.

References S_start().

Here is the call graph for this function:



2.2 sudoku.c

Go to the documentation of this file.

```
00001
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <time.h>
00005 #include <ncurses.h>
00006 #include "sudoku_ncurses.h"
00007 #include "sudoku_solve.h"
00008 #include "sudoku_user.h"
00009
00010 #define DEBUG
```

```
00012 int main(void){
00013
         srand(time(NULL));
00014
00015
         initscr(); // start ncurses
00016
00017
         S_start();
00018
00019
         endwin(); // end ncurses
00020
          return 0;
00021
00022 }
```

2.3 sudoku_ncurses.c File Reference

```
#include <ncurses.h>
#include <stdbool.h>
#include "sudoku_ncurses.h"
#include "sudoku_solve.h"
#include "sudoku_user.h"
```

Macros

• #define DEBUG

Functions

WINDOW * S_create_screen (void)

Draw a second screen.

• int S_create_color (void)

Colour options.

void S_start (void)

Calls appropriate functions for the program.

void N_screen_Title (void)

Function to display title.

int N_screen_Menu (void)

Function to display menu.

• int N_play_options (void)

SUbmenu.

void N_drawBoard (void)

Draw board to ncurses window.

- void N_blank_row (int n, WINDOW *s)
- void N blank col (int n, WINDOW *s)
- void N_moveCursor (void)

Traverse along a sudoku board and overwrite numbers if they are zero.

• void N_typeNumbers (int ch, int row, int col)

Allow numbers to be typed over the sudoku board.

2.3.1 Macro Definition Documentation

DEBUG

#define DEBUG

Definition at line 7 of file sudoku_ncurses.c.

2.3.2 Function Documentation

N_blank_col()

Insert a blank column when drawing grid

Parameters

in <i>n</i>	Col number
-------------	------------

Definition at line 247 of file sudoku ncurses.c.

```
00247

00248

00249

00250

00251

00252 }

(n == 0 || n == 3 || n == 6) {

wprintw(s," ");

00251

wrefresh(s);
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



N_blank_row()

```
void N_blank_row (  \mbox{int } n, \\ \mbox{WINDOW } * s)
```

insert a blank row when drawing sudoku

Parameters



Definition at line 238 of file sudoku_ncurses.c.

Referenced by N_drawBoard().



N_drawBoard()

```
void N_{draw}Board ( void )
```

Draw board to ncurses window.

Maybe I should centre this and for every (i + 1) % 3 draw hline and (j + 1) % 3 draw vline.

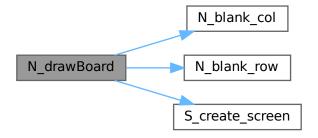
Definition at line 215 of file sudoku_ncurses.c.

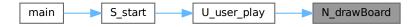
```
00215
00216
              WINDOW *s = S_create_screen();
wbkgd(s,COLOR_PAIR(2));
00217
00218
00219
              for (size_t row = 0; row < BOARD_SIZE; row++) {
   N_blank_row(row, s);
   for (size_t col = 0; col < BOARD_SIZE; col++) {
        N_blank_col(col, s);
   }
}</pre>
00220
00221
00222
00223
                          if (SudokuGrid[row][col] == 0) {
    wprintw(s," ");
00224
00225
00226
00227
                          wprintw(s,"%d ", SudokuGrid[row][col]);
00228
00229
00230
                    wprintw(s,"\n");
00231
              }
00232
              wrefresh(s); // updates current screen » standard screen
00234 }
```

References BOARD_SIZE, N_blank_col(), N_blank_row(), S_create_screen(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:





N_moveCursor()

```
void N_{moveCursor} ( void )
```

Traverse along a sudoku board and overwrite numbers if they are zero.

Definition at line 256 of file sudoku ncurses.c.

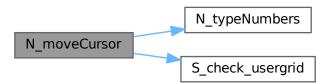
```
00256
          int row, col, value;
row = 0; col = 0;
00257
00258
00259
00260
          curs_set(1);
00261
          refresh();
00262
00263
          int ch, y_loc, x_loc; // current character
00264
          move(1, 1);
00265
00266
          getyx(stdscr, y_loc, x_loc);
00267
          while ((ch = getch()) != '\n'){
00268
              switch(ch) {
    case 'w': // move up
00269
00270
00271
                            if (y_loc == 1) {
00272
                            break;
} else if (y_loc == 5 || y_loc == 9) {
00273
00274
                                move(y_loc -= 2, x_loc);
                                row--;
00275
00276
                               break:
00277
                            } else {
00278
                               move(y_loc -= 1, x_loc);
00279
00280
                                break;
00281
                   case 's':
00282
00283
                           if (y_loc == 11) {
00284
                           break;
} else if (y_loc == 3 || y_loc == 7){
00285
00286
                               move(y_{loc} += 2, x_{loc});
                                row++;
00287
00288
                               break;
00289
                            } else {
00290
                                move(y_loc += 1, x_loc);
00291
                                row++;
00292
00293
                   case 'a':
00294
                           if (x_loc == 1) {
00295
00296
                           break;
} else if (x_loc == 8 || x_loc == 15){
00297
00298
                                move(y_{loc}, x_{loc} = 3);
00299
                                col--;
00300
                                break:
00301
                            } else {
00302
                               move(y_loc, x_loc -= 2);
00303
00304
                                break;
00305
                   case 'd':
00306
00307
                           if (x_loc == 19) {
00308
                           break;
} else if (x_loc == 5 || x_loc == 12){
00309
00310
                               move(y_loc, x_loc += 3);
00311
                                col++;
00312
                                break;
00313
                            } else {
                               move(y_loc, x_loc += 2);
00314
00315
                                col++;
00316
                                break;
00317
00318
00319
00320
              N_typeNumbers(ch, row, col);
00321
          }
00322
00323
          if(ch == ' \n') {
00324
              move (13,0); // move to bottom
               //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00325
00326
00327
               // Put a version of solve sudoku here
00328
               if (S_check_usergrid(SudokuGrid)){
                   printw("Well done! You correctly solved the sudoku!\n");
```

```
00330
              } else {
00331
                  printw("Unfortunately that isn't the correct solution\n");
00332
00333
00334
              refresh();
00335
00336
              nodelay(stdscr, false); // allow getch() to pause execution
00337
              getch();
00338
              clear;
00339
          }
00340 }
```

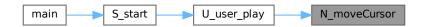
References N_typeNumbers(), S_check_usergrid(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



N_play_options()

SUbmenu.

Definition at line 150 of file sudoku ncurses.c.

```
00150
                int play_menu_selection = 1, ch;
char *play_menu_Options[] = {
    " 1 Easy\n",
    " 2 Normal\n",
    " 3 Hard\n",
    " 4 Back\n"
}
00151
00152
00153
00154
00155
00156
00157
                };
00158
00159
                attroff(A_REVERSE);
00160
                N_screen_Title();
00161
```

```
00162
          for (size_t i = 0; i < 4; i++) {</pre>
             if (play_menu_selection = i) {
00164
                  attron(A_REVERSE);
              } else {
00165
00166
                  attroff(A_REVERSE);
00167
00168
              printw("%s", play_menu_Options[i]);
00169
00170
          while ((ch = getch()) != ' n') {
00171
00172
00173
              switch(ch){
00174
                  case 'w':
00175
                      if (play_menu_selection == 0) {
00176
                          play_menu_selection = 3;
00177
00178
                          play_menu_selection--;
00179
                      }
00180
                      break;
00181
                  case 's':
00182
                      if (play_menu_selection == 3) {
00183
                          play_menu_selection = 0;
                      play_menu_selection++;
}
                      } else {
00184
00185
00186
00187
                      break;
00188
                  default:
00189
              }
00190
00191
00192
              move(4,0);
00193
00194
              for (size_t i = 0; i < 4; i++){</pre>
00195
                  if (play_menu_selection == i) {
00196
                      attron(A_REVERSE);
00197
                  } else {
00198
                      attroff(A_REVERSE);
00199
00200
                  printw("%s", play_menu_Options[i]);
00201
00202
          }
00203
00204
          clear():
00205
          refresh();
00206
00207
          return play_menu_selection;
00208 }
```

References N_screen_Title().

Referenced by S_start().

Here is the call graph for this function:





N_screen_Menu()

```
int N_screen_Menu (
     void )
```

Function to display menu.

- 1. A random board is generated in full (to check it works) and and then numbers are removed before user is given control
- 2. The player can enter some details onto the board and then the computer solves the puzzle (different colour/bold for solutions)

Definition at line 84 of file sudoku_ncurses.c.

```
int menu_selection = 0, ch;
char *menu_Options[] = {
    " 1 Play Sudoku\n",
    " 2 Exit\n",
00085
00086
00087
00088
00089
00090
00091
           attroff(A_REVERSE);
00092
          N_screen_Title();
00093
00094
           move (4, 0);
00095
          for (size_t i = 0; i < 2; i++) {
    if (menu_selection == i) {</pre>
00096
00097
00098
                    attron(A_REVERSE);
00099
               } else {
                   attroff(A_REVERSE);
00100
00101
               printw("%s", menu_Options[i]);
00102
00103
00104
00105
           // hide the cursor, no echo and no delay
00106
           curs_set(0);
00107
           noecho();
00108
          nodelay(stdscr, true);
00109
00110
           while ((ch = getch()) != ' \n') {
00111
00112
               switch(ch){
                    case 'w':
00113
00114
                        if (menu_selection == 0) {
                            menu_selection = 1;
00115
                         } else {
00116
00117
                             menu_selection--;
                        }
00118
00119
                        break:
                    case 's':
00120
00121
                       if (menu_selection == 1) {
00122
                            menu_selection = 0;
00123
                        } else {
00124
                           menu_selection++;
00125
00126
                        break:
00127
                    default:
00128
                        break;
00129
               }
00130
00131
               move(4,0);
00132
00133
               for (size_t i = 0; i < 2; i++) {</pre>
                    if (menu_selection == i) {
00134
00135
                        attron(A_REVERSE);
00136
                        attroff(A_REVERSE);
00137
00138
                    printw("%s", menu_Options[i]);
00139
00140
               }
00141
00142
00143
           clear();
00144
           refresh();
00145
00146
           return menu_selection;
00147 }
```

References N_screen_Title().

Referenced by S_start().

Here is the call graph for this function:



Here is the caller graph for this function:



N_screen_Title()

```
void N_screen_Title (
    void )
```

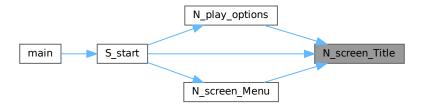
Function to display title.

Definition at line 65 of file sudoku_ncurses.c.

```
00065
           int MenuChoice = 0;
00066
00067
           move(0,1);
attron(A_BOLD);
attron(A_UNDERLINE);
00068
00069
00070
           addstr("Sudoku\n");
00072
           move(2,1);
00073
           attroff(A_BOLD);
00074
           attroff(A\_UNDERLINE);
00075
00076
           addstr("Please choose an option:\n\n");
```

Referenced by N_play_options(), N_screen_Menu(), and S_start().

Here is the caller graph for this function:



N_typeNumbers()

```
void N_typeNumbers (
          int ch,
          int row,
          int col)
```

Allow numbers to be typed over the sudoku board.

Parameters

in	ch	Character
in	row	Row
in	col	Col

Definition at line 347 of file sudoku_ncurses.c.

```
00347
                                                             {
00348
            int curs_row, curs_col;
00349
00350
            getyx(stdscr,curs_row,curs_col);
00351
            echo();
00352
            int count = 0;
00353
00354
            switch(ch){
                case '0':
00355
                     SudokuGrid[row][col] = 0;
addch('0' | A_REVERSE);
00356
00357
00358
                break; case '1':
00359
                    SudokuGrid[row][col] = 1;
00360
00361
                     addch('1' | A_REVERSE);
                break; case '2':
00362
00363
                     SudokuGrid[row][col] = 2;
addch('2' | A_REVERSE);
00364
00365
00366
                     break;
00367
                 case '3':
00368
                     SudokuGrid[row][col] = 3;
00369
                     addch('3' | A_REVERSE);
                break;
case '4':
SudokuGrid[row][col] = 4;
addch('4' | A_REVERSE);
00370
00371
00372
00373
00374
00375
                   SudokuGrid[row][col] = 5;
addch('5' | A_REVERSE);
00376
00377
00378
                     break:
00379
                 case '6':
00380
                     SudokuGrid[row][col] = 6;
```

```
00381
                   addch('6' | A_REVERSE);
               break; case '7':
00382
00383
                   SudokuGrid[row][col] = 7;
addch('7' | A_REVERSE);
00384
00385
00386
                   break:
00387
               case '8':
00388
                   SudokuGrid[row][col] = 8;
00389
                   addch('8' | A_REVERSE);
               break; case '9':
00390
00391
00392
                  SudokuGrid[row][col] = 9;
                   addch('9' | A_REVERSE);
00393
00394
                   break;
00395
               default:
00396
                   break;
00397
          }
00398
00399
          move(curs_row, curs_col);
00400
           refresh();
00401
           noecho();
00402 }
```

References SudokuGrid.

Referenced by N_moveCursor().

Here is the caller graph for this function:



S_create_color()

```
int S_create_color (
     void )
```

Colour options.

Definition at line 27 of file sudoku_ncurses.c.

Referenced by S_start().



S_create_screen()

Draw a second screen.

Create a different window for the sudoku board. More of an experiment than actually required

Returns

s_screen A pointer to a variable of type WINDOW

Definition at line 14 of file sudoku_ncurses.c.

```
00014 {
00015 WINDOW *s_screen;
00016
00017 s_screen = newwin(0,0,0,0);
00018
00019 if(s_screen == NULL) {
00020 endwin();
00021 } else {
00022 return s_screen;
00023 }
00024 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



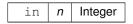
S_start()

```
void S_start (
     void )
```

Calls appropriate functions for the program.

Receives integer n representing menu choice and calls relevant functions related to user choice.

Parameters



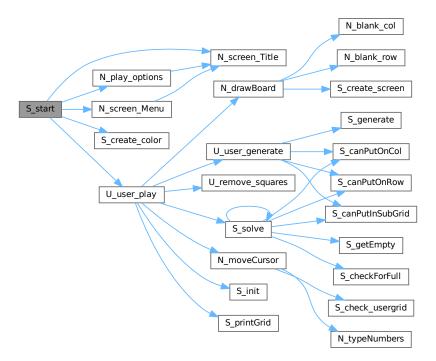
Definition at line 42 of file sudoku_ncurses.c.

```
00051
          while (Selection_Menu != 1) {
              Selection_Menu = N_screen_Menu(); // Main menu
00052
00053
00054
              // MAIN MENU - PLAY
00055
              if (Selection_Menu == 0) {
00056
                  // OPEN SUBMENU
00057
                  Selection_Menu_Sub = N_play_options();
00058
                  U_user_play(Selection_Menu_Sub);
00059
00060
              clear();
00061
          }
00062 }
```

References N_play_options(), N_screen_Menu(), N_screen_Title(), S_create_color(), and U_user_play().

Referenced by main().

Here is the call graph for this function:





2.4 sudoku_ncurses.c

Go to the documentation of this file.

```
00001 #include <ncurses.h>
00002 #include <stdbool.h>
00003 #include "sudoku_ncurses.h"
00004 #include "sudoku_solve.h"
00005 #include "sudoku_user.h"
00006
00007 #define DEBUG
80000
00014 WINDOW* S_create_screen(void){
00015
          WINDOW *s_screen;
00016
00017
           s\_screen = newwin(0,0,0,0);
00018
00019
          if(s_screen == NULL){
00020
              endwin();
           } else {
00021
00022
              return s_screen;
00023
00024 }
00025
00027 int S_create_color(void){
00028
          start_color();
00030
           init_pair(1, COLOR_RED, COLOR_WHITE); // stdscr
00031
           init_pair(2, COLOR_BLUE, COLOR_WHITE); // sudoku screen
00032
00033
          bkgd(COLOR PAIR(1));
00034 }
00035
00042 void S_start(void){
00043
          S_create_color();
00044
00045
           // menu options
          int Selection_Menu = 0;
00046
00047
          int Selection_Menu_Sub = 0;
00049
          N_screen_Title();
00050
          while (Selection Menu != 1) {
00051
00052
               Selection_Menu = N_screen_Menu(); // Main menu
00053
               // MAIN MENU - PLAY
00055
               if (Selection_Menu == 0) {
00056
                    // OPEN SUBMENU
00057
                    Selection_Menu_Sub = N_play_options();
00058
                   U_user_play(Selection_Menu_Sub);
00059
00060
               clear();
00061
           }
00062 }
00063
00065 void N_screen_Title(void){
00066    int MenuChoice = 0;
00067
          move(0,1);
00069
           attron(A_BOLD);
00070
           attron (A_UNDERLINE);
00071
          addstr("Sudoku\n");
          move(2,1);
attroff(A_BOLD);
00072
00073
00074
          attroff(A_UNDERLINE);
00075
00076
           addstr("Please choose an option:\n\n");
00077 }
00078
00084 int N_screen_Menu(void){
          int menu_selection = 0, ch;
           char *menu_Options[] = {
    " 1 Play Sudoku\n",
    " 2 Exit\n",
00086
00087
00088
00089
          };
00090
00091
          attroff(A_REVERSE);
00092
          N_screen_Title();
00093
00094
          move(4,0);
00095
00096
           for (size_t i = 0; i < 2; i++) {</pre>
               if (menu_selection == i) {
00097
00098
                   attron(A_REVERSE);
00099
00100
                   attroff(A_REVERSE);
```

```
00101
00102
              printw("%s", menu_Options[i]);
00103
00104
          // hide the cursor, no echo and no delay
00105
00106
          curs_set(0);
00107
          noecho();
00108
          nodelay(stdscr, true);
00109
          while ((ch = getch()) != ' \n'){
00110
00111
00112
              switch(ch){
                  case 'w':
00113
00114
                      if (menu_selection == 0) {
00115
                           menu_selection = 1;
00116
                       } else {
00117
                         menu_selection--;
00118
00119
                       break;
                   case 's':
00120
00121
                      if (menu_selection == 1) {
00122
                           menu_selection = 0;
                      menu_selection++;
}
                       } else {
00123
00124
00125
00126
                       break;
00127
                   default:
00128
                      break;
              }
00129
00130
00131
              move(4,0);
00132
00133
               for (size_t i = 0; i < 2; i++){</pre>
00134
                  if (menu_selection == i) {
00135
                       attron(A\_REVERSE);
                   } else {
00136
00137
                      attroff(A_REVERSE);
00138
00139
                  printw("%s", menu_Options[i]);
00140
00141
          }
00142
          clear():
00143
00144
          refresh();
00145
00146
          return menu_selection;
00147 }
00148
00150 int N_play_options(void){
        int play_menu_selection = 1, ch;
00151
          char *play_menu_Options[] = {
    " 1 Easy\n",
00152
00153
              " 2 Normal\n",
00154
              " 3 Hard\n",
" 4 Back\n"
00155
00156
00157
          };
00159
          attroff(A_REVERSE);
00160
          N_screen_Title();
00161
          for (size_t i = 0; i < 4; i++){</pre>
00162
00163
              if (play_menu_selection = i) {
00164
                  attron(A_REVERSE);
00165
00166
                  attroff(A_REVERSE);
00167
00168
              printw("%s", play_menu_Options[i]);
00169
          }
00170
          while ((ch = getch()) != ' n') {
00172
00173
              switch(ch){
                  case 'w':
00174
                      if (play_menu_selection == 0) {
00175
00176
                           play_menu_selection = 3;
                       } else {
00177
                         play_menu_selection--;
00178
00179
00180
                      break;
                   case 's':
00181
                      if (play_menu_selection == 3) {
00182
00183
                          play_menu_selection = 0;
00184
                         play_menu_selection++;
00185
00186
00187
                       break;
00188
                  default:
```

```
00189
                        break;
00190
               }
00191
00192
                move(4,0);
00193
00194
                for (size_t i = 0; i < 4; i++){</pre>
00195
                    if (play_menu_selection == i) {
00196
                         attron(A_REVERSE);
00197
00198
                        attroff(A_REVERSE);
00199
                    printw("%s", play_menu_Options[i]);
00200
00201
               }
00202
00203
00204
           clear();
00205
           refresh();
00206
00207
           return play_menu_selection;
00208 }
00209
00215 void N_drawBoard(void) {
00216
           WINDOW *s = S_create_screen();
00217
00218
           wbkqd(s,COLOR_PAIR(2));
00219
00220
           for (size_t row = 0; row < BOARD_SIZE; row++) {</pre>
               N_blank_row(row, s);
for (size_t col = 0; col < BOARD_SIZE; col++) {
    N_blank_col(col, s);</pre>
00221
00222
00223
                    if (SudokuGrid[row][col] == 0) {
   wprintw(s," ");
00224
00225
                    } else {
00226
00227
                    wprintw(s,"%d ", SudokuGrid[row][col]);
00228
00229
00230
                wprintw(s,"\n");
00231
00232
00233
           wrefresh(s); // updates current screen » standard screen
00234 }
00235
00238 void N_blank_row(int n, WINDOW *s) {
00239         if (n == 0 || n == 3 || n == 6) {
00240
               wprintw(s,"\n");
00241
00242
               wrefresh(s);
00243 }
00244
00247 void N_blank_col(int n, WINDOW *s) {
00248         if (n == 0 || n == 3 || n == 6) {
00249             wprintw(s," ");
00250
           }
00251
               wrefresh(s);
00252 }
00253
00256 void N_moveCursor(void){
00257
          int row, col, value;
00258
          row = 0; col = 0;
00259
00260
           curs set(1):
00261
           refresh();
00262
00263
           int ch, y_loc, x_loc; // current character
00264
           move(1,1);
00265
00266
           getyx(stdscr, y_loc, x_loc);
00267
00268
           while ((ch = getch()) != ' \n') {
00269
               switch(ch){
                   case 'w': // move up
00270
00271
                             if (y_loc == 1) {
                              break;
} else if (y_loc == 5 || y_loc == 9) {
00272
00273
                                 move(y_loc -= 2, x_loc);
00274
00275
                                  row--;
00276
                                  break;
00277
                              } else {
00278
                                  move(y_{loc} -= 1, x_{loc});
00279
                                  row--:
00280
                                  break;
00281
00282
                     case 's':
00283
                             if (y_loc == 11) {
                             00284
00285
00286
```

```
00287
                                row++;
00288
                                break;
00289
                            } else {
00290
                                move(y_loc += 1, x_loc);
00291
                                row++:
00292
                                break:
00293
00294
                   case 'a':
00295
                            if (x_loc == 1) {
00296
                                 break;
                            } else if (x_loc == 8 || x_loc == 15) {
00297
                                move(y_loc, x_loc -= 3);
00298
00299
                                col--;
00300
                                break;
00301
                            } else {
00302
                                move(y_loc, x_loc -= 2);
00303
                                col--:
00304
                                break:
00305
                   case 'd':
00306
00307
                            if (x_loc == 19) {
                            break;
} else if (x_loc == 5 || x_loc == 12){
00308
00309
00310
                                move(y_loc, x_loc += 3);
00311
                                col++;
00312
                                break;
00313
                            } else {
00314
                                move(y_{loc}, x_{loc} += 2);
                                col++;
00315
00316
                                break:
00317
                            }
00318
               }
00319
00320
               N_typeNumbers(ch, row, col);
00321
          }
00322
00323
          if(ch == '\n'){
              move (13,0); // move to bottom
00325
               //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00326
00327
               // Put a version of solve sudoku here
               if (S_check_usergrid(SudokuGrid)) {
    printw("Well done! You correctly solved the sudoku!\n");
00328
00329
00330
               } else {
00331
                   printw("Unfortunately that isn't the correct solution\n");
00332
               }
00333
00334
               refresh();
00335
00336
               nodelay(stdscr, false); // allow getch() to pause execution
00337
               getch();
00338
00339
           }
00340 }
00341
00347 void N typeNumbers (int ch, int row, int col) {
          int curs_row, curs_col;
00349
00350
           getyx(stdscr,curs_row,curs_col);
00351
00352
          echo():
00353
          int count = 0;
00354
          switch(ch){
00355
              case '0':
00356
                   SudokuGrid[row][col] = 0;
00357
                   addch('0' | A_REVERSE);
00358
               break; case '1':
00359
00360
                  SudokuGrid[row][col] = 1;
00361
                   addch('1' | A_REVERSE);
00362
               break; case '2':
00363
                  SudokuGrid[row][col] = 2;
addch('2' | A_REVERSE);
00364
00365
00366
                   break;
               case '3':
00367
00368
                  SudokuGrid[row][col] = 3;
00369
                   addch('3' | A_REVERSE);
               break; case '4':
00370
00371
00372
                  SudokuGrid[row][col] = 4;
                   addch('4' | A_REVERSE);
00373
00374
                   break;
00375
               case '5':
                  SudokuGrid[row][col] = 5;
addch('5' | A_REVERSE);
00376
00377
00378
                   break:
```

```
00379
              case '6':
                 SudokuGrid[row][col] = 6;
addch('6' | A_REVERSE);
00380
00381
              break; case '7':
00382
00383
               SudokuGrid[row][col] = 7;
00384
00385
                   addch('7' | A_REVERSE);
00386
00387
              SudokuGrid[row][col] = 8;
addch('8' | A_REVERSE);
00388
00389
00390
                   break;
              case '9':
00391
00392
                 SudokuGrid[row][col] = 9;
00393
                   addch('9' | A_REVERSE);
00394
                   break;
              default:
00395
00396
                  break;
00397
          }
00398
00399
          move(curs_row, curs_col);
00400
           refresh();
00401
          noecho();
00402 }
```

2.5 sudoku_ncurses.h File Reference

Functions

WINDOW * S_create_screen (void)

Draw a second screen.

• int S_create_color (void)

Colour options.

void S start (void)

Calls appropriate functions for the program.

void N_screen_Title (void)

Function to display title.

• int N_screen_Menu (void)

Function to display menu.

• int N play options (void)

SUbmenu.

void N_drawBoard (void)

Draw board to ncurses window.

- void N_blank_row (int n, WINDOW *s)
- void N blank col (int n, WINDOW *s)
- void N_moveCursor (void)

Traverse along a sudoku board and overwrite numbers if they are zero.

void N_typeNumbers (int ch, int row, int col)

Allow numbers to be typed over the sudoku board.

Variables

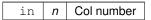
• int SudokuGrid [][9]

2.5.1 Function Documentation

N_blank_col()

Insert a blank column when drawing grid

Parameters



Definition at line 247 of file sudoku_ncurses.c.

Referenced by N_drawBoard().

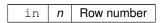
Here is the caller graph for this function:



N_blank_row()

insert a blank row when drawing sudoku

Parameters



Definition at line 238 of file sudoku_ncurses.c.

```
00238

00239

00240

00240

00241

00242

00242

00242

00243 }

(n == 0 || n == 3 || n == 6) {

wprintw(s,"\n");

004;

wrefresh(s);
```

Referenced by N_drawBoard().



N_drawBoard()

```
void N_{draw}Board ( void )
```

Draw board to ncurses window.

Maybe I should centre this and for every (i + 1) % 3 draw hline and (j + 1) % 3 draw vline.

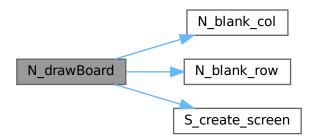
Definition at line 215 of file sudoku_ncurses.c.

```
00215
00216
              WINDOW *s = S_create_screen();
wbkgd(s,COLOR_PAIR(2));
00217
00218
00219
              for (size_t row = 0; row < BOARD_SIZE; row++) {
   N_blank_row(row, s);
   for (size_t col = 0; col < BOARD_SIZE; col++) {
        N_blank_col(col, s);
   }
}</pre>
00220
00221
00222
00223
                          if (SudokuGrid[row][col] == 0) {
    wprintw(s," ");
00224
00225
00226
00227
                          wprintw(s,"%d ", SudokuGrid[row][col]);
00228
00229
00230
                    wprintw(s,"\n");
00231
              }
00232
              wrefresh(s); // updates current screen » standard screen
00234 }
```

References BOARD_SIZE, N_blank_col(), N_blank_row(), S_create_screen(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:





N_moveCursor()

```
void N_{moveCursor} ( void )
```

Traverse along a sudoku board and overwrite numbers if they are zero.

Definition at line 256 of file sudoku ncurses.c.

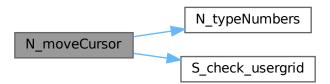
```
00256
          int row, col, value;
row = 0; col = 0;
00257
00258
00259
00260
          curs_set(1);
00261
          refresh();
00262
00263
          int ch, y_loc, x_loc; // current character
00264
          move(1, 1);
00265
00266
          getyx(stdscr, y_loc, x_loc);
00267
          while ((ch = getch()) != '\n'){
00268
              switch(ch) {
    case 'w': // move up
00269
00270
00271
                           if (y_loc == 1) {
00272
                            break;
} else if (y_loc == 5 || y_loc == 9) {
00273
00274
                                move(y_loc -= 2, x_loc);
                                row--;
00275
00276
                               break:
00277
                            } else {
00278
                               move(y_loc -= 1, x_loc);
00279
00280
                                break;
00281
                   case 's':
00282
00283
                           if (y_loc == 11) {
00284
                           break;
} else if (y_loc == 3 || y_loc == 7){
00285
00286
                               move(y_{loc} += 2, x_{loc});
                                row++;
00287
00288
                               break;
00289
                            } else {
00290
                                move(y_loc += 1, x_loc);
00291
                                row++;
00292
00293
                   case 'a':
00294
                           if (x_loc == 1) {
00295
00296
                           break;
} else if (x_loc == 8 || x_loc == 15){
00297
00298
                                move(y_{loc}, x_{loc} = 3);
00299
                                col--;
00300
                                break:
00301
                            } else {
00302
                               move(y_loc, x_loc -= 2);
00303
00304
                                break;
00305
                   case 'd':
00306
00307
                           if (x_loc == 19) {
00308
                           break;
} else if (x_loc == 5 || x_loc == 12){
00309
00310
                               move(y_loc, x_loc += 3);
00311
                                col++;
00312
                                break;
00313
                            } else {
                               move(y_loc, x_loc += 2);
00314
00315
                                col++;
00316
                                break;
00317
00318
00319
00320
              N_typeNumbers(ch, row, col);
00321
          }
00322
00323
          if(ch == ' \n') {
00324
              move (13,0); // move to bottom
               //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00325
00326
00327
              // Put a version of solve sudoku here
00328
               if (S_check_usergrid(SudokuGrid)) {
                   printw("Well done! You correctly solved the sudoku!\n");
```

```
00330
              } else {
00331
                  printw("Unfortunately that isn't the correct solution\n");
00332
00333
00334
              refresh();
00335
00336
              nodelay(stdscr, false); // allow getch() to pause execution
00337
              getch();
00338
              clear;
00339
          }
00340 }
```

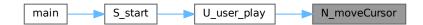
References N_typeNumbers(), S_check_usergrid(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



N_play_options()

SUbmenu.

Definition at line 150 of file sudoku ncurses.c.

```
00150
                int play_menu_selection = 1, ch;
char *play_menu_Options[] = {
    " 1 Easy\n",
    " 2 Normal\n",
    " 3 Hard\n",
    " 4 Back\n"
}
00151
00152
00153
00154
00155
00156
00157
                };
00158
00159
                attroff(A_REVERSE);
00160
                N_screen_Title();
00161
```

```
00162
          for (size_t i = 0; i < 4; i++) {</pre>
             if (play_menu_selection = i) {
00164
                  attron(A_REVERSE);
              } else {
00165
00166
                  attroff(A_REVERSE);
00167
00168
              printw("%s", play_menu_Options[i]);
00169
00170
          while ((ch = getch()) != ' n') {
00171
00172
00173
              switch(ch){
00174
                  case 'w':
00175
                      if (play_menu_selection == 0) {
00176
                          play_menu_selection = 3;
00177
00178
                          play_menu_selection--;
00179
                      }
00180
                      break;
00181
                  case 's':
00182
                      if (play_menu_selection == 3) {
00183
                          play_menu_selection = 0;
                      play_menu_selection++;
}
                      } else {
00184
00185
00186
00187
                      break;
00188
                  default:
00189
              }
00190
00191
00192
              move(4,0);
00193
00194
              for (size_t i = 0; i < 4; i++){</pre>
00195
                  if (play_menu_selection == i) {
00196
                      attron(A_REVERSE);
00197
                  } else {
00198
                      attroff(A_REVERSE);
00199
00200
                  printw("%s", play_menu_Options[i]);
00201
00202
          }
00203
00204
          clear():
00205
          refresh();
00206
00207
          return play_menu_selection;
00208 }
```

References N_screen_Title().

Referenced by S_start().

Here is the call graph for this function:





N_screen_Menu()

```
int N_screen_Menu (
    void )
```

Function to display menu.

- 1. A random board is generated in full (to check it works) and and then numbers are removed before user is given control
- 2. The player can enter some details onto the board and then the computer solves the puzzle (different colour/bold for solutions)

Definition at line 84 of file sudoku_ncurses.c.

```
int menu_selection = 0, ch;
char *menu_Options[] = {
    " 1 Play Sudoku\n",
    " 2 Exit\n",
00085
00086
00087
00088
00089
          };
00090
00091
           attroff(A_REVERSE);
00092
          N_screen_Title();
00093
00094
          move (4, 0);
00095
          for (size_t i = 0; i < 2; i++) {
    if (menu_selection == i) {</pre>
00096
00097
00098
                    attron(A_REVERSE);
00099
               } else {
                   attroff(A_REVERSE);
00100
00101
               printw("%s", menu_Options[i]);
00102
00103
00104
00105
           // hide the cursor, no echo and no delay
00106
           curs_set(0);
00107
           noecho();
00108
          nodelay(stdscr, true);
00109
00110
           while ((ch = getch()) != ' \n') {
00111
00112
               switch(ch){
                    case 'w':
00113
00114
                        if (menu_selection == 0) {
                            menu_selection = 1;
00115
                        } else {
00116
00117
                             menu_selection--;
                        }
00118
00119
                        break:
                    case 's':
00120
00121
                       if (menu_selection == 1) {
00122
                            menu_selection = 0;
00123
                        } else {
00124
                           menu_selection++;
00125
00126
                        break:
00127
                   default:
00128
                        break;
00129
               }
00130
00131
               move(4,0);
00132
00133
               for (size_t i = 0; i < 2; i++) {</pre>
                    if (menu_selection == i) {
00134
00135
                        attron(A_REVERSE);
00136
                        attroff(A_REVERSE);
00137
00138
                    printw("%s", menu_Options[i]);
00139
00140
               }
00141
00142
00143
           clear();
00144
           refresh();
00145
00146
           return menu_selection;
00147 }
```

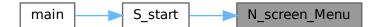
References N_screen_Title().

Referenced by S_start().

Here is the call graph for this function:



Here is the caller graph for this function:



N_screen_Title()

```
void N_screen_Title (
    void )
```

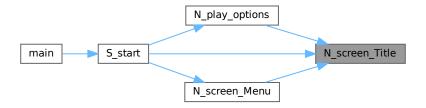
Function to display title.

Definition at line 65 of file sudoku_ncurses.c.

```
00065
           int MenuChoice = 0;
00066
00067
           move(0,1);
attron(A_BOLD);
attron(A_UNDERLINE);
00068
00069
00070
           addstr("Sudoku\n");
00072
           move(2,1);
00073
           attroff(A_BOLD);
00074
           attroff(A\_UNDERLINE);
00075
00076
           addstr("Please choose an option:\n\n");
```

Referenced by N_play_options(), N_screen_Menu(), and S_start().

Here is the caller graph for this function:



N_typeNumbers()

```
void N_typeNumbers (
          int ch,
          int row,
          int col)
```

Allow numbers to be typed over the sudoku board.

Parameters

in	ch	Character
in	row	Row
in	col	Col

Definition at line 347 of file sudoku_ncurses.c.

```
00347
                                                             {
00348
            int curs_row, curs_col;
00349
00350
            getyx(stdscr,curs_row,curs_col);
00351
            echo();
00352
            int count = 0;
00353
00354
            switch(ch){
               case '0':
00355
                     SudokuGrid[row][col] = 0;
addch('0' | A_REVERSE);
00356
00357
00358
                break; case '1':
00359
                    SudokuGrid[row][col] = 1;
00360
00361
                     addch('1' | A_REVERSE);
                break; case '2':
00362
00363
                    SudokuGrid[row][col] = 2;
addch('2' | A_REVERSE);
00364
00365
00366
                     break;
00367
                 case '3':
00368
                     SudokuGrid[row][col] = 3;
00369
                     addch('3' | A_REVERSE);
                break;
case '4':
SudokuGrid[row][col] = 4;
addch('4' | A_REVERSE);
00370
00371
00372
00373
00374
00375
                   SudokuGrid[row][col] = 5;
addch('5' | A_REVERSE);
00376
00377
00378
                     break:
00379
                 case '6':
00380
                     SudokuGrid[row][col] = 6;
```

```
00381
                   addch('6' | A_REVERSE);
               break; case '7':
00382
00383
                   SudokuGrid[row][col] = 7;
addch('7' | A_REVERSE);
00384
00385
00386
                   break:
00387
               case '8':
00388
                   SudokuGrid[row][col] = 8;
00389
                   addch('8' | A_REVERSE);
               break; case '9':
00390
00391
00392
                  SudokuGrid[row][col] = 9;
                   addch('9' | A_REVERSE);
00393
00394
                   break;
00395
               default:
00396
                   break;
00397
          }
00398
00399
          move(curs_row, curs_col);
00400
           refresh();
00401
           noecho();
00402 }
```

References SudokuGrid.

Referenced by N_moveCursor().

Here is the caller graph for this function:



S_create_color()

```
int S_create_color (
     void )
```

Colour options.

Definition at line 27 of file sudoku_ncurses.c.

Referenced by S_start().



S_create_screen()

Draw a second screen.

Create a different window for the sudoku board. More of an experiment than actually required

Returns

s_screen A pointer to a variable of type WINDOW

Definition at line 14 of file sudoku_ncurses.c.

```
00014 {
00015 WINDOW *s_screen;
00016
00017 s_screen = newwin(0,0,0,0);
00018
00019 if(s_screen == NULL) {
00020 endwin();
00021 } else {
00022 return s_screen;
00023 }
00024 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



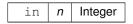
S_start()

```
void S_start (
     void )
```

Calls appropriate functions for the program.

Receives integer n representing menu choice and calls relevant functions related to user choice.

Parameters



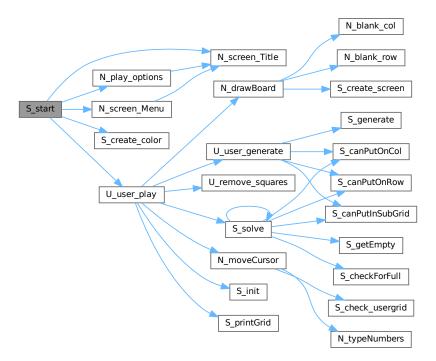
Definition at line 42 of file sudoku_ncurses.c.

```
00051
          while (Selection_Menu != 1) {
              Selection_Menu = N_screen_Menu(); // Main menu
00052
00053
00054
              // MAIN MENU - PLAY
00055
              if (Selection_Menu == 0) {
00056
                  // OPEN SUBMENU
00057
                  Selection_Menu_Sub = N_play_options();
00058
                  U_user_play(Selection_Menu_Sub);
00059
00060
              clear();
00061
          }
00062 }
```

References N_play_options(), N_screen_Menu(), N_screen_Title(), S_create_color(), and U_user_play().

Referenced by main().

Here is the call graph for this function:





2.5.2 Variable Documentation

SudokuGrid

```
int SudokuGrid[][9] [extern]
```

Definition at line 13 of file sudoku_solve.c.

Referenced by N_drawBoard(), N_moveCursor(), N_typeNumbers(), S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_getEmpty(), S_printGrid(), S_solve(), U_remove_squares(), U_user_generate(), and U_user_play().

2.6 sudoku_ncurses.h

Go to the documentation of this file.

```
00001
00002 #ifndef SUDOKU_NCURSES_H
00003 #define SUDOKU_NCURSES_H
00004
00005 extern int SudokuGrid[][9];
00006
00007 WINDOW* S_create_screen(void);
00008 int S_create_color(void);
00009 void S_start(void);
00010 void N_screen_Title(void);
00011 int N_screen_Menu(void);
00012 int N_play_options(void);
00013 void N_drawBoard(void);
00014 void N_blank_row(int n, WINDOW *s);
00015 void N_blank_col(int n, WINDOW *s);
00016
00017 void N_moveCursor(void);
00018 void N_{typeNumbers} (int ch, int row, int col);
00019 #endif
```

2.7 sudoku solve.c File Reference

```
#include <stdio.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include <ncurses.h>
#include <math.h>
#include "sudoku_solve.h"
#include "sudoku_user.h"
#include "sudoku_ncurses.h"
```

Functions

• bool S check usergrid (int grid[][BOARD SIZE])

Return true if count for each row is 45.

void S_init (int grid[][BOARD_SIZE])

Initialise all grid elements to zero.

void S generate (int *r, int *c, int *n)

Produce a random number, in a random place on the board.

• bool S_solve (int a[][BOARD_SIZE])

Solve a sudoku puzzle, if blank, generate one.

bool S_getEmpty (int *r, int *c)

find an empty grid square

bool S_checkForFull (int grid[][BOARD_SIZE])

Check if grid is full.

bool S_canPutOnRow (int row, int n)

Function to check if a number can be placed on row.

bool S canPutOnCol (int col, int n)

Function to check if a number can be placed on col.

bool S_canPutInSubGrid (int row, int col, int n)

Check if number exists in subgrid.

void S_printGrid (void)

DEBUG - print grids to console.

Variables

int SudokuGrid [BOARD_SIZE][BOARD_SIZE]

2.7.1 Function Documentation

S_canPutInSubGrid()

Check if number exists in subgrid.

Provides bounds to check if the number has already appeared

Definition at line 154 of file sudoku solve.c.

```
00154
00155
          int len = BOARD SIZE:
00156
         int index = row * len + col;
         int x = sqrt(len);
00157
00158
         int box_index = (index % len) / x + x * (index / (len * x));
00159
00160
00161
         int col_start;
00162
         switch (box_index) {
00163
         case 0: row_start = 0;
00164
00165
                     col_start = 0;
00166
00167
             case 1: row_start = 0;
                     col_start = 3;
00168
00169
                     break:
00170
             case 2: row_start = 0;
00171
                     col_start = 6;
00172
00173
             case 3: row_start = 3;
                     col_start = 0;
00174
00175
                     break:
00176
             case 4: row_start = 3;
                     col_start = 3;
00178
                     break;
00179
             case 5: row_start = 3;
                    col_start = 6;
00180
00181
                     break:
00182
             case 6: row start = 6:
00183
                    col_start = 0;
00184
                     break;
```

```
case 7: row_start = 6;
00186
                          col_start = 3;
00187
                          break;
                 00188
00189
00190
                          break:
00191
           }
00192
            for (size_t i = 0; i < 3; i++) {
    for (size_t j = 0; j < 3; j++) {
        if (SudokuGrid[row_start + i][col_start + j] == n) {</pre>
00193
00194
00195
00196
                          return false;
00197
00198
00199
00200
00201
            return true;
00202 }
```

References BOARD SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_canPutOnCol()

```
bool S_canPutOnCol (
          int col,
          int n)
```

Function to check if a number can be placed on col.

Parameters

ſ	in	col	Proposed col of varaible n
ſ	in	n	Integer number from 1 to 9 to check

Returns

bool True if n can be placed

Definition at line 142 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S canPutOnRow()

Function to check if a number can be placed on row.

For each iteration, i is checked to see if it contains the number to check, n. If it's found, return false.

Parameters

in	row	Proposed row of varaible n
in	n	Integer number from 1 to 9 to check

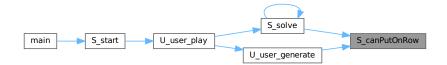
Returns

bool True if n can be placed

Definition at line 128 of file sudoku solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().



S_check_usergrid()

```
bool S_check_usergrid ( int \ grid \cite{board_size})
```

Return true if count for each row is 45.

The sum of any row should equal 45. Rather than use resources to resolve a puzzle thats already known, I decided to add up each row

Returns

bool

Definition at line 20 of file sudoku_solve.c.

```
00021
             int count;
00022
            for(size_t row = 0; row < BOARD_SIZE; row++) {
    count = 0;</pre>
00023
00024
                  for (size_t col = 0; col < BOARD_SIZE; col++) {
    count += grid[row][col];</pre>
00025
00026
00028
                  if (count == 45) {
                 continue;
} else {
00029
00030
00031
                      return false;
00032
00033
00034
             return true;
00035 }
```

References BOARD_SIZE.

Referenced by N_moveCursor().

Here is the caller graph for this function:



S_checkForFull()

Check if grid is full.

Parameters



Definition at line 110 of file sudoku_solve.c.

References BOARD_SIZE.

Referenced by S_solve().

Here is the caller graph for this function:



S_generate()

```
void S_generate (
    int * r,
    int * c,
    int * n)
```

Produce a random number, in a random place on the board.

Parameters

in,out	* r	Pointer to integer representing a row	
in,out	* <i>r</i>	Pointer to integer representing a col	
in,out	* <i>r</i>	Pointer to integer representing a number	

Definition at line 55 of file sudoku_solve.c.

References BOARD_SIZE.

Referenced by U_user_generate().

Here is the caller graph for this function:



S_getEmpty()

find an empty grid square

Parameters

in,out	* <i>r</i>	pointer to an empty row
in,out	*C	pointer to an empty col

Definition at line 95 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve().

Here is the caller graph for this function:



S_init()

Initialise all grid elements to zero.

This is used on first start up and also if a potential solution is not possible before trying another

Parameters

in,out	grid	2D array of size BOARD_SIZE

Definition at line 42 of file sudoku solve.c.

```
00042
00043
for(size_t row = 0; row < BOARD_SIZE; row++) {
00044
    for (size_t col = 0; col < BOARD_SIZE; col++) {
        grid[row][col] = 0;
00046
    }
00047
}
00048 }</pre>
```

References BOARD_SIZE.

Referenced by U_user_play().

Here is the caller graph for this function:



S_printGrid()

```
void S_printGrid (
     void )
```

DEBUG - print grids to console.

Definition at line 206 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by U_user_play().

Here is the caller graph for this function:



S_solve()

```
bool S_solve ( \label{eq:solve} \text{int } a[\,][\texttt{BOARD\_SIZE}])
```

Solve a sudoku puzzle, if blank, generate one.

Uses a backtracing algorithm to place a value 1 through 9 into an empty space and then move on to another empty square. If an empty square has no solutions, the algorithm resets that square and 'backtracks' to try another option.

Parameters

```
in, out | a | 2D array
```

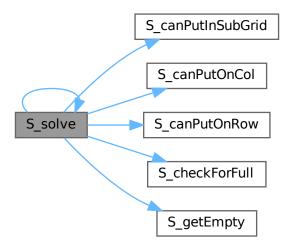
Definition at line 68 of file sudoku_solve.c.

```
00068
00069
          int row, col;
00070
          S\_getEmpty(\&row, \&col); // this needs to be counted
00071
                                   // doesn't matter which ones are removed at the end
00072
00073
          if (S_checkForFull(a)){
00075
              return true;
00076
00077
          for (size_t i = 1; i <= 9; i++) {</pre>
00078
00079
             if (S_canPutOnRow(row,i) &&
08000
                      S_canPutOnCol(col,i) &&
00081
                           S_canPutInSubGrid(row, col, i)){
00082
                  SudokuGrid[row][col] = i;
00083
                  if (S_solve(a)) {
00084
                       return true;
00085
00086
00087
              SudokuGrid[row][col] = 0;
00088
00089
          return false;
00090 }
```

References BOARD_SIZE, S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_checkForFull(), S_getEmpty(), S_solve(), and SudokuGrid.

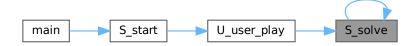
Referenced by S_solve(), and U_user_play().

Here is the call graph for this function:



2.8 sudoku solve.c 41

Here is the caller graph for this function:



2.7.2 Variable Documentation

SudokuGrid

```
int SudokuGrid[BOARD_SIZE] [BOARD_SIZE]
```

Definition at line 13 of file sudoku solve.c.

Referenced by N_drawBoard(), N_moveCursor(), N_typeNumbers(), S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_getEmpty(), S_printGrid(), S_solve(), U_remove_squares(), U_user_generate(), and U_user_play().

2.8 sudoku_solve.c

Go to the documentation of this file.

```
00001 //https://stackoverflow.com/questions/32343262/sudoku-checker-accessing-3x3-subgrid-in-c
00002 //https://iq.opengenus.org/backtracking-sudoku/
00003 #include <stdio.h>
00004 #include <stdbool.h>
00005 #include <stddef.h>
00006 #include <stdlib.h>
00007 #include <ncurses.h>
00008 #include <math.h>
00009 #include "sudoku_solve.h"
00010 #include "sudoku_user.h"
00011 #include "sudoku_ncurses.h"
00012
00013 int SudokuGrid[BOARD_SIZE][BOARD_SIZE];
00014
00020 bool S_check_usergrid(int grid[][BOARD_SIZE]){
00021
         int count;
00022
00023
          for(size_t row = 0; row < BOARD_SIZE; row++) {</pre>
00024
              count = 0:
00025
              for (size_t col = 0; col < BOARD_SIZE; col++) {</pre>
                  count += grid[row][col];
00027
00028
              if (count == 45) {
00029
                  continue;
              } else {
00030
00031
                  return false;
00032
00033
00034
          return true;
00035 }
00036
00045
                  grid[row][col] = 0;
00046
00047
00048 }
00055 void S_generate(int *r, int *c, int *n){
```

```
*r = rand() % BOARD_SIZE;
         *c = rand() % BOARD_SIZE;
*n = rand() % (9 - 1 + 1) + 1;
00057
00058
00059 }
00060
00068 bool S_solve(int a[][BOARD_SIZE]){
00069
         int row, col;
00070
00071
          S\_getEmpty(\&row, \&col); // this needs to be counted
                                  // doesn't matter which ones are removed at the end \,
00072
00073
00074
          if (S checkForFull(a)){
            return true;
00075
00076
00077
00078
          for (size_t i = 1; i <= 9; i++) {</pre>
00079
              if (S_canPutOnRow(row,i) &&
08000
                     S_canPutOnCol(col,i) &&
                         S_canPutInSubGrid(row, col, i)){
00082
                  SudokuGrid[row][col] = i;
00083
                  if (S_solve(a)) {
00084
                      return true;
00085
00086
00087
              SudokuGrid[row][col] = 0;
00088
00089
          return false;
00090 }
00091
00099
                      *r = i;
                      *c = j;
00100
00101
                      return true;
00102
                  }
            }
00104
00105
          return false;
00106 }
00107
00110 bool S_checkForFull(int grid[][BOARD_SIZE]){
         for (size_t i = 0; i < BOARD_SIZE; i++){
    for (size_t j = 0; j < BOARD_SIZE; j++){
        if (grid[i][j] == 0) {</pre>
00111
00112
00113
00114
                      return false;
00115
                  }
            }
00116
00117
          }
00118
          return true;
00119 }
00120
return false;
00132
00133
00134
          return true;
00135 }
00136
00142 bool S_canPutOnCol(int col, int n){
00143
        for (size_t i = 0; i < BOARD_SIZE; i++) {</pre>
00144
             if (SudokuGrid[i][col] == n) {
00145
                  return false;
00146
             }
00147
          }
00148
          return true:
00149 }
00150
00154 bool S_canPutInSubGrid(int row, int col, int n){
00155
         int len = BOARD_SIZE;
00156
          int index = row * len + col;
00157
          int x = sqrt(len);
00158
          int box_index = (index % len) / x + x * (index / (len * x));
00159
00160
          int row_start;
00161
         int col_start;
00162
00163
          switch (box_index) {
00164
            case 0: row_start = 0;
00165
                      col_start = 0;
00166
                      break;
00167
              case 1: row_start = 0;
                     col_start = 3;
00168
00169
                      break:
```

```
00170
             case 2: row_start = 0;
00171
                    col_start = 6;
00172
                     break;
             case 3: row_start = 3;
00173
             col_start = 0;
00174
00175
                     break:
00176
             case 4: row_start = 3;
00177
                    col_start = 3;
00178
00179
             case 5: row_start = 3;
                    col_start = 6;
00180
00181
                     break:
00182
             case 6: row_start = 6;
                    col_start = 0;
00183
00184
                     break;
00185
             case 7: row_start = 6;
             col_start = 3;
00186
00187
                     break;
00188
             case 8: row_start = 6;
               col_start = 6;
break;
00189
00190
00191
         }
00192
         for (size_t i = 0; i < 3; i++) {
    for (size_t j = 0; j < 3; j++) {
        if (SudokuGrid[row_start + i][col_start + j] == n) {</pre>
00193
00194
00195
00196
                      return false;
00197
00198
             }
00199
         }
00200
00201
         return true;
00202 }
00203
00210
00211
             printf("\n");
00212
         }
00213 }
```

2.9 sudoku_solve.h File Reference

Macros

• #define BOARD_SIZE 9

Size of board.

Functions

• bool S_check_usergrid (int grid[][BOARD_SIZE])

Return true if count for each row is 45.

• void S_init (int grid[][BOARD_SIZE])

Initialise all grid elements to zero.

• void S_generate (int *r, int *c, int *n)

Produce a random number, in a random place on the board.

bool S_solve (int a[][BOARD_SIZE])

Solve a sudoku puzzle, if blank, generate one.

bool S_getEmpty (int *r, int *c)

find an empty grid square

bool S_checkForFull (int grid[][BOARD_SIZE])

Check if grid is full.

bool S_canPutOnRow (int row, int n)

Function to check if a number can be placed on row.

bool S_canPutOnCol (int col, int n)

Function to check if a number can be placed on col.

• bool S_canPutInSubGrid (int row, int col, int n)

Check if number exists in subgrid.

void S_printGrid ()

DEBUG - print grids to console.

2.9.1 Macro Definition Documentation

BOARD_SIZE

```
#define BOARD_SIZE 9
```

Size of board.

Definition at line 8 of file sudoku_solve.h.

2.9.2 Function Documentation

S_canPutInSubGrid()

Check if number exists in subgrid.

Provides bounds to check if the number has already appeared

Definition at line 154 of file sudoku_solve.c.

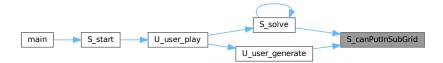
```
00154
                                                      {
00155
          int len = BOARD_SIZE;
00156
          int index = row * len + col;
00157
          int x = sqrt(len);
         int box_index = (index % len) / x + x * (index / (len * x));
00158
00159
00160
          int row_start;
         int col_start;
00161
00162
00163
         switch (box_index) {
             case 0: row_start = 0;
00164
                      col_start = 0;
00165
00166
                      break:
00167
             case 1: row_start = 0;
                      col_start = 3;
00168
00170
              case 2: row_start = 0;
                      col_start = 6;
00171
00172
                      break:
00173
              case 3: row_start = 3;
                      col_start = 0;
00175
                      break;
00176
              case 4: row_start = 3;
                     col_start = 3;
00177
00178
                      break;
00179
              case 5: row start = 3:
00180
                     col_start = 6;
00181
                      break;
```

```
case 6: row_start = 6;
                          col\_start = 0;
00184
                         break;
                00185
00186
00187
                          break:
00188
                case 8: row_start = 6;
00189
                         col_start = 6;
00190
00191
           }
00192
           for (size_t i = 0; i < 3; i++) {
    for (size_t j = 0; j < 3; j++) {
        if (SudokuGrid[row_start + i][col_start + j] == n) {</pre>
00193
00194
00195
00196
                         return false;
00197
00198
00199
           }
00200
00201
           return true;
00202 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_canPutOnCol()

Function to check if a number can be placed on col.

Parameters

in	col	Proposed col of varaible n
in	n	Integer number from 1 to 9 to check

Returns

bool True if n can be placed

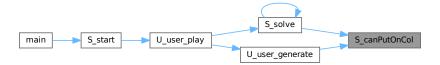
Definition at line 142 of file sudoku_solve.c.

```
00149 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_canPutOnRow()

Function to check if a number can be placed on row.

For each iteration, i is checked to see if it contains the number to check, n. If it's found, return false.

Parameters

in	row	Proposed row of varaible n
in	n	Integer number from 1 to 9 to check

Returns

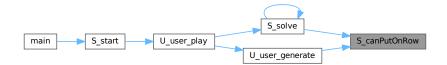
bool True if n can be placed

Definition at line 128 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_check_usergrid()

Return true if count for each row is 45.

The sum of any row should equal 45. Rather than use resources to resolve a puzzle thats already known, I decided to add up each row

Returns

bool

Definition at line 20 of file sudoku_solve.c.

```
00021
            int count;
00022
            for(size_t row = 0; row < BOARD_SIZE; row++) {
    count = 0;</pre>
00023
00024
                 for (size_t col = 0; col < BOARD_SIZE; col++) {
    count += grid[row][col];</pre>
00025
00026
00028
                 if (count == 45) {
00029
00030
                 } else {
00031
                      return false;
00032
00033
00034
            return true;
00035 }
```

References BOARD_SIZE.

Referenced by N_moveCursor().

Here is the caller graph for this function:



S_checkForFull()

Check if grid is full.

Parameters

```
in grid[][9] A sudoku array
```

Definition at line 110 of file sudoku_solve.c.

References BOARD_SIZE.

Referenced by S_solve().

Here is the caller graph for this function:



S_generate()

Produce a random number, in a random place on the board.

Parameters

in,out	* r	Pointer to integer representing a row	
in,out	* <i>r</i>	Pointer to integer representing a col	
in,out	* <i>r</i>	Pointer to integer representing a number	

Definition at line 55 of file sudoku_solve.c.

References BOARD_SIZE.

Referenced by U_user_generate().

Here is the caller graph for this function:



S_getEmpty()

find an empty grid square

Parameters

in,out	* <i>r</i>	pointer to an empty row
in,out	*C	pointer to an empty col

Definition at line 95 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve().

Here is the caller graph for this function:



S_init()

```
void S_init (
          int grid[][BOARD_SIZE])
```

Initialise all grid elements to zero.

This is used on first start up and also if a potential solution is not possible before trying another

Parameters

in,out	grid	2D array of size BOARD_SIZE

Definition at line 42 of file sudoku_solve.c.

References BOARD SIZE.

Referenced by U_user_play().

Here is the caller graph for this function:



S_printGrid()

```
void S_printGrid ()
```

DEBUG - print grids to console.

Definition at line 206 of file sudoku_solve.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by U_user_play().

Here is the caller graph for this function:



S_solve()

```
bool S_solve ( \label{eq:solve} \text{int } a[\,][\texttt{BOARD\_SIZE}])
```

Solve a sudoku puzzle, if blank, generate one.

Uses a backtracing algorithm to place a value 1 through 9 into an empty space and then move on to another empty square. If an empty square has no solutions, the algorithm resets that square and 'backtracks' to try another option.

Parameters

in,out	а	2D array
--------	---	----------

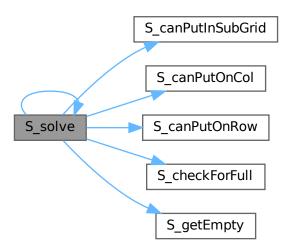
Definition at line 68 of file sudoku solve.c.

```
00068
00069
          int row, col;
00070
00071
          S_getEmpty(&row, &col); // this needs to be counted
                                   // doesn't matter which ones are removed at the end
00072
00073
00073
          if (S_checkForFull(a)){
00075
              return true;
00076
00077
00078
          for (size_t i = 1; i <= 9; i++) {</pre>
           if (S_canPutOnRow(row,i) &&
00079
                     S_canPutOnCol(col,i) &&
    S_canPutInSubGrid(row, col, i)){
00080
00081
                  SudokuGrid[row][col] = i;
00082
00083
                  if (S_solve(a)){
00084
                      return true;
00085
00086
              SudokuGrid[row][col] = 0;
00087
00088
00089
          return false;
00090 }
```

References BOARD_SIZE, S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_checkForFull(), S_getEmpty(), S_solve(), and SudokuGrid.

Referenced by S_solve(), and U_user_play().

Here is the call graph for this function:



2.10 sudoku solve.h 53

Here is the caller graph for this function:



2.10 sudoku_solve.h

Go to the documentation of this file.

```
00001 #ifndef SUDOKU_SOLVE_H
00002 #define SUDOKU_SOLVE_H
00003
00008 #define BOARD_SIZE 9
00009
00010 bool S_check_usergrid(int grid[][BOARD_SIZE]);
00011 void S_init(int grid[][BOARD_SIZE]);
00012 void S_generate(int *r, int *c, int *n);
00013 bool S_solve(int a[][BOARD_SIZE]);
00014 bool S_getEmpty(int *r, int *c);
00015 bool S_checkForFull(int grid[][BOARD_SIZE]);
00016 bool S_canPutOnRow(int row, int n);
00017 bool S_canPutOnCol(int col, int n);
00018 bool S_canPutInSubGrid(int row, int col, int n);
00019 void S_printGrid();
00020
00021 #endif
```

2.11 sudoku_user.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <ncurses.h>
#include "sudoku_user.h"
#include "sudoku_solve.h"
#include "sudoku_ncurses.h"
```

Functions

bool U_user_play (int n)

Calls a number of function to solve a randomly generated sudoku puzzle.

void U_user_generate (int n)

Generate a sudoku board for the user.

void U_remove_squares (int n)

Removes a designated number of squares from a solvable board.

2.11.1 Function Documentation

U_remove_squares()

```
void U_remove_squares ( int n)
```

Removes a designated number of squares from a solvable board.

Parameters

in	n	Number of squares to remove
----	---	-----------------------------

Definition at line 79 of file sudoku_user.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by U user play().

Here is the caller graph for this function:



U_user_generate()

```
void U_user_generate (
          int n)
```

Generate a sudoku board for the user.

Takes as an argument n number of squares to fill on the board. For each number, check if it's possible to place. Increment counter if possible

Parameters

```
in Number of squares to fill.
```

Definition at line 52 of file sudoku_user.c.

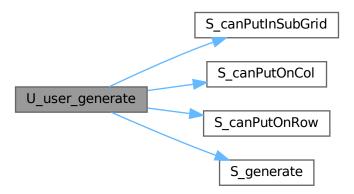
```
00052
00053
          int count, row, col, number;
00054
          count = 0;
00055
00056
          while (count < n) {</pre>
              S_generate(&row, &col, &number);
00057
00058
              if (SudokuGrid[row][col] != 0){
00059
                  continue;
              } else if(
00060
                           S canPutOnRow(row, number) &&
00061
00062
                          S_canPutOnCol(col,number) &&
00063
                           S_canPutInSubGrid(row, col, number)){
00064
                  SudokuGrid[row][col] = number;
00065 #ifdef DEBUG
                  printf("Coord - Row: %d Col: %d Value: %d\n", row, col, number);
00066
00067 #endif
00068
                  count++;
00069
              } else {
00070
                  continue;
```

```
00071 }
00072 }
00073 }
```

 $References \ S_canPutInSubGrid(), \ S_canPutOnCol(), \ S_canPutOnRow(), \ S_generate(), \ and \ SudokuGrid.$

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



U_user_play()

```
bool U_user_play ( \quad \text{int } n)
```

Calls a number of function to solve a randomly generated sudoku puzzle.

Initialise a SudokuGrid and generate solvable puzzle. Remove squares then hand over control to user

Parameters

in	n	Difficulty level selected by user input
----	---	---

Definition at line 16 of file sudoku_user.c.

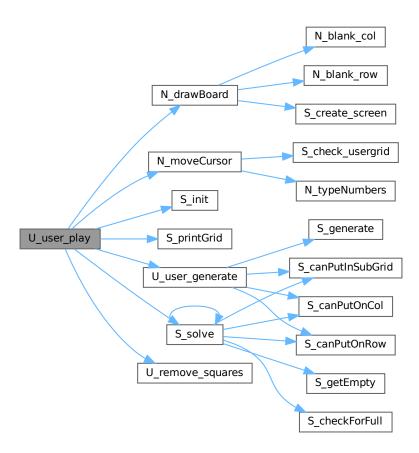
```
00016
00017
          int squaresToRemove = 0;
00018
00019
          switch(n){
00020
             case 0: squaresToRemove = EASY;
00021
              case 1: squaresToRemove = NORMAL;
00022
00023
                      break:
00024
              case 2: squaresToRemove = HARD;
                      break;
00026
              case 3:
00027
                      return false;
00028
00029
00030
          S_init(SudokuGrid);
00031
          U_user_generate(PLACEMENTS);
00032
00033
          if(S_solve(SudokuGrid)){
00034
              U_remove_squares(squaresToRemove); // eventually a user choice
00035 #ifdef DEBUG
00036
              S_printGrid();
00037 #endif
00038
              N_drawBoard();
00039
             N_moveCursor();
00040
          } else {
            puts("Not solvable");
00041
00042
00043
          return true;
00045 }
```

References EASY, HARD, N_drawBoard(), N_moveCursor(), NORMAL, PLACEMENTS, S_init(), S_printGrid(), S_solve(), SudokuGrid, U_remove_squares(), and U_user_generate().

Referenced by S_start().

2.12 sudoku user.c 57

Here is the call graph for this function:



Here is the caller graph for this function:



2.12 sudoku_user.c

Go to the documentation of this file.

```
00001
00002 #include <stdio.h>
00003 #include <stdlib.h>
00004 #include <stdbool.h>
00005 #include <ncurses.h>
00006 #include "sudoku_user.h"
00007 #include "sudoku_solve.h" // had to include stdbool as THIS references bool
00008 #include "sudoku_ncurses.h"
```

```
00009
00016 bool U_user_play(int n){
         int squaresToRemove = 0;
00017
00018
00019
         switch(n){
            case 0: squaresToRemove = EASY;
00020
                      break;
00022
              case 1: squaresToRemove = NORMAL;
00023
00024
              case 2: squaresToRemove = HARD;
00025
                     break;
00026
             case 3:
00027
                      return false;
00028
00029
         S_init(SudokuGrid);
00030
         U_user_generate(PLACEMENTS);
00031
00032
         if(S_solve(SudokuGrid)){
00034
              U_remove_squares(squaresToRemove); // eventually a user choice
00035 #ifdef DEBUG
00036
             S_printGrid();
00037 #endif
00038
              N drawBoard();
00039
             N_moveCursor();
00040
         } else {
            puts("Not solvable");
00041
         }
00042
00043
00044
         return true:
00045 }
00046
00052 void U_user_generate(int n){
00053
         int count, row, col, number;
00054
         count = 0;
00055
00056
         while (count < n) {
             S_generate(&row, &col, &number);
00058
             if (SudokuGrid[row][col] != 0) {
00059
                  continue;
00060
             } else if(
                          S canPutOnRow(row, number) &&
00061
                          S_canPutOnCol(col,number) &&
00062
                          S_canPutInSubGrid(row, col, number)){
00063
                 SudokuGrid[row][col] = number;
00065 #ifdef DEBUG
00066
                 printf("Coord - Row: %d Col: %d Value: %d\n", row, col, number);
00067 #endif
00068
                 count++;
00069
             } else {
                 continue;
00071
             }
00072
         }
00073 }
00074
00079 void U_remove_squares(int n){
       int rand_row, rand_col;
00081
          for (size_t i = 0; i <= n; i++) {</pre>
              rand_row = rand() % BOARD_SIZE; rand_col = rand() % BOARD_SIZE;
00082
00083
              SudokuGrid[rand_row][rand_col] = 0;
00084
         }
00085 }
```

2.13 sudoku user.h File Reference

Macros

```
• #define PLACEMENTS 10
```

Places 10 numbers.

• #define EASY 2

difficulty

• #define NORMAL 20

difficulty

#define HARD 40

difficulty

Functions

bool U_user_play (int n)

Calls a number of function to solve a randomly generated sudoku puzzle.

void U_user_generate (int n)

Generate a sudoku board for the user.

void U_remove_squares (int n)

Removes a designated number of squares from a solvable board.

Variables

• int SudokuGrid [][9]

2.13.1 Macro Definition Documentation

EASY

```
#define EASY 2
difficulty
Definition at line 27 of file sudoku_user.h.
Referenced by U_user_play().
```

HARD

```
#define HARD 40
```

difficulty

Definition at line 29 of file sudoku_user.h.

Referenced by U_user_play().

NORMAL

```
#define NORMAL 20
```

difficulty

Definition at line 28 of file sudoku_user.h.

Referenced by U_user_play().

PLACEMENTS

```
#define PLACEMENTS 10
```

Places 10 numbers.

When the sudoku grid is generated, the program places 10 numbers in randomly chosen squares

Definition at line 24 of file sudoku_user.h.

Referenced by U_user_play().

2.13.2 Function Documentation

U_remove_squares()

```
void U_remove_squares (
          int n)
```

Removes a designated number of squares from a solvable board.

Parameters

in Number of squares to remove	/e
--------------------------------	----

Definition at line 79 of file sudoku_user.c.

References BOARD_SIZE, and SudokuGrid.

Referenced by U user play().

Here is the caller graph for this function:



U_user_generate()

```
void U_user_generate (
          int n)
```

Generate a sudoku board for the user.

Takes as an argument n number of squares to fill on the board. For each number, check if it's possible to place. Increment counter if possible

Parameters

```
in Number of squares to fill.
```

Definition at line 52 of file sudoku_user.c.

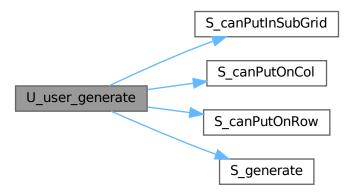
```
00052
00053
          int count, row, col, number;
00054
          count = 0;
00055
00056
         while (count < n) {
             S_generate(&row, &col, &number);
00057
00058
              if (SudokuGrid[row][col] != 0){
00059
                  continue;
00060
              } else if(
                          S_canPutOnRow(row, number) &&
00061
00062
                          S_canPutOnCol(col,number) &&
00063
                          S_canPutInSubGrid(row, col, number)){
00064
                  SudokuGrid[row][col] = number;
00065 #ifdef DEBUG
                  printf("Coord - Row: %d Col: %d Value: %d\n", row, col, number);
00066
00067 #endif
00068
                  count++;
00069
             } else {
00070
                  continue;
```

```
00071
00072 }
00073 }
```

 $References \ S_canPutInSubGrid(), \ S_canPutOnCol(), \ S_canPutOnRow(), \ S_generate(), \ and \ SudokuGrid.$

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



U_user_play()

bool U_user_play (
$$\quad \text{int } n)$$

Calls a number of function to solve a randomly generated sudoku puzzle.

Initialise a SudokuGrid and generate solvable puzzle. Remove squares then hand over control to user

Parameters

in	n	Difficulty level selected by user input
----	---	---

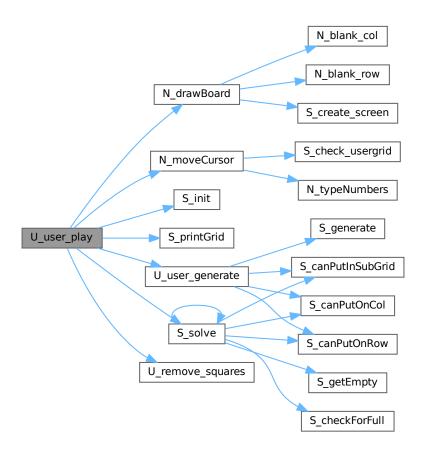
Definition at line 16 of file sudoku_user.c.

```
00016
00017
         int squaresToRemove = 0;
00018
         switch(n){
00020
             case 0: squaresToRemove = EASY;
00021
             case 1: squaresToRemove = NORMAL;
00022
00023
                      break:
00024
             case 2: squaresToRemove = HARD;
                     break;
00026
              case 3:
00027
                      return false;
00028
00029
00030
         S_init(SudokuGrid);
00031
         U_user_generate(PLACEMENTS);
00032
00033
         if(S_solve(SudokuGrid)){
00034
             U_remove_squares(squaresToRemove); // eventually a user choice
00035 #ifdef DEBUG
00036
             S_printGrid();
00037 #endif
00038
             N_drawBoard();
00039
             N_moveCursor();
00040
         } else {
            puts("Not solvable");
00041
00042
00043
          return true;
00045 }
```

References EASY, HARD, N_drawBoard(), N_moveCursor(), NORMAL, PLACEMENTS, S_init(), S_printGrid(), S_solve(), SudokuGrid, U_remove_squares(), and U_user_generate().

Referenced by S_start().

Here is the call graph for this function:



Here is the caller graph for this function:



2.13.3 Variable Documentation

SudokuGrid

int SudokuGrid[][9] [extern]

Definition at line 13 of file sudoku_solve.c.

2.14 sudoku_user.h 65

2.14 sudoku_user.h

Go to the documentation of this file.

```
00001
00002 #ifndef SUDOOKU_USER_H
00003 #define SUDOOKU_USER_H
00001
00011
00015
00019
00023
00024 #define PLACEMENTS 10
00025
00026 // difficulty levels
00027 #define EASY 2
00028 #define NORMAL 20
00029 #define HARD 40
00030
00031 extern int SudokuGrid[][9];
00032
00033 bool U_user_play(int n);
00034 void U_user_generate(int n);
00035 void U_remove_squares(int n);
```

Index

BOARD_SIZE	sudoku_solve.h, 46			
sudoku_solve.h, 44	S_checkForFull			
	sudoku_solve.c, 36			
DEBUG	sudoku_solve.h, 47			
sudoku.c, 2	S_create_color			
sudoku_ncurses.c, 4	sudoku_ncurses.c, 13			
FAOV	sudoku_ncurses.h, 29			
EASY	S_create_screen			
sudoku_user.h, 59	sudoku_ncurses.c, 13			
HARD	sudoku_ncurses.h, 29			
sudoku_user.h, 59	S_generate			
Sudoku_user.n, ss	sudoku_solve.c, 37			
main	sudoku_solve.h, 48			
sudoku.c, 3	S_getEmpty			
,	sudoku_solve.c, 37			
N_blank_col	sudoku_solve.h, 48			
sudoku_ncurses.c, 5	S_init			
sudoku_ncurses.h, 20	sudoku_solve.c, 38			
N_blank_row	sudoku_solve.h, 49			
sudoku_ncurses.c, 5	S_printGrid			
sudoku_ncurses.h, 21	sudoku_solve.c, 39			
N drawBoard	sudoku_solve.h, 50			
sudoku_ncurses.c, 5	S_solve			
sudoku_ncurses.h, 21	sudoku_solve.c, 39			
N_moveCursor	sudoku_solve.h, 50			
sudoku_ncurses.c, 6	S_start			
sudoku_ncurses.h, 22	sudoku_ncurses.c, 14			
N_play_options	sudoku_ncurses.h, 30			
sudoku_ncurses.c, 8	sudoku.c, 2			
sudoku_ncurses.h, 24	DEBUG, 2			
N_screen_Menu	main, 3			
sudoku_ncurses.c, 10	sudoku_ncurses.c, 4			
sudoku_ncurses.h, 26	DEBUG, 4			
N_screen_Title	N_blank_col, 5			
sudoku_ncurses.c, 11	N_blank_row, 5			
sudoku_ncurses.h, 27	N_drawBoard, 5			
N_typeNumbers	N_moveCursor, 6			
sudoku_ncurses.c, 12	N_play_options, 8			
sudoku_ncurses.h, 28	N_screen_Menu, 10			
NORMAL	N_screen_Title, 11			
sudoku user.h, 59	N_typeNumbers, 12			
	S_create_color, 13			
PLACEMENTS	S_create_screen, 13			
sudoku_user.h, 59	S_start, 14			
	sudoku_ncurses.h, 20			
S_canPutInSubGrid	N_blank_col, 20			
sudoku_solve.c, 33	N_blank_row, 21			
sudoku_solve.h, 44	N_drawBoard, 21			
S_canPutOnCol	N_moveCursor, 22			
sudoku_solve.c, 34	N_play_options, 24			
sudoku_solve.h, 45	N_screen_Menu, 26			
S_canPutOnRow	N_screen_Title, 27			
sudoku_solve.c, 35	N_typeNumbers, 28			
sudoku_solve.h, 46	S_create_color, 29			
S_check_usergrid	S_create_screen, 29			
sudoku_solve.c, 35	_			

68 INDEX

```
S_start, 30
    SudokuGrid, 32
sudoku_solve.c, 32
    S_canPutInSubGrid, 33
    S_canPutOnCol, 34
    S canPutOnRow, 35
    S_check_usergrid, 35
    S_checkForFull, 36
    S generate, 37
    S_getEmpty, 37
    S_init, 38
    S_printGrid, 39
    S_solve, 39
    SudokuGrid, 41
sudoku_solve.h, 43
    BOARD_SIZE, 44
    S canPutInSubGrid, 44
    S canPutOnCol, 45
    S_canPutOnRow, 46
    S_check_usergrid, 46
    S _checkForFull, 47
    S_generate, 48
    S_getEmpty, 48
    S_init, 49
    S printGrid, 50
    S_solve, 50
sudoku_user.c, 53
    U remove squares, 53
    U user generate, 54
    U_user_play, 55
sudoku_user.h, 58
    EASY, 59
    HARD, 59
    NORMAL, 59
    PLACEMENTS, 59
    SudokuGrid, 64
    U_remove_squares, 60
    U_user_generate, 61
    U_user_play, 62
SudokuGrid
    sudoku ncurses.h, 32
    sudoku_solve.c, 41
    sudoku_user.h, 64
U_remove_squares
    sudoku_user.c, 53
    sudoku user.h, 60
U_user_generate
    sudoku_user.c, 54
    sudoku_user.h, 61
U user play
    sudoku_user.c, 55
    sudoku_user.h, 62
```