Sudoku

Generated by Doxygen 1.13.2

1 File Index

1.1 File List

Here is a list of all files with brief descriptions:

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.

2.1.2 Function Documentation

main()

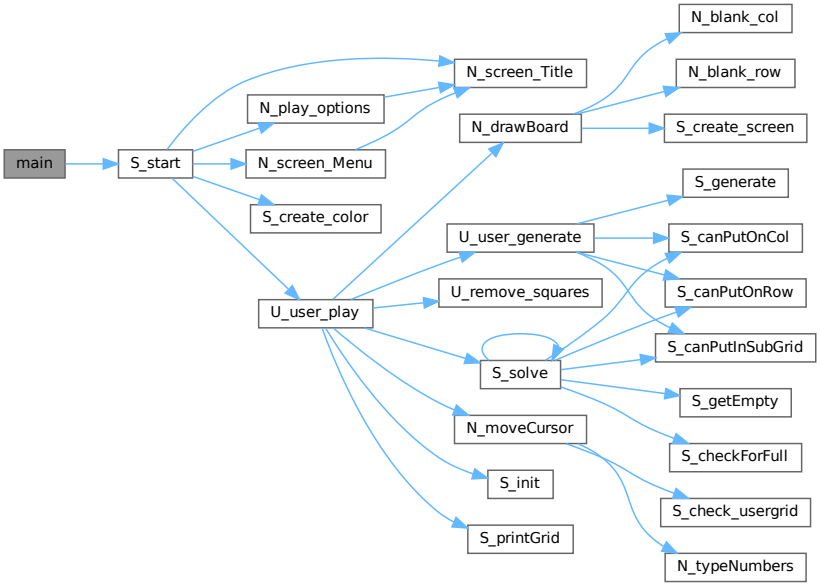
```
int main (
            void )
```

Definition at line 12 of file sudoku.c.

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

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
00011
```

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

```
void N_blank_col (
            int n,
            WINDOW * s)
```

Insert a blank column when drawing grid

Parameters

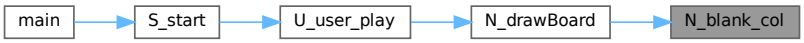
in	*n*	Col number

Definition at line 247 of file sudoku_ncurses.c.

```
00247                                   {
00248     if (n == 0 || n == 3 || n == 6){
00249         wprintw(s," ");
00250     }
00251         wrefresh(s);
00252 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



N_blank_row()

```
void N_blank_row (
            int n,
            WINDOW * s)
```

insert a blank row when drawing sudoku

Parameters

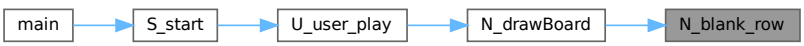
in	*n*	Row number

Definition at line 238 of file sudoku_ncurses.c.

```
00238                                   {
00239     if (n == 0 || n == 3 || n == 6) {
00240         wprintw(s,"\n");
00241     }
00242         wrefresh(s);
00243 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



N_drawBoard()

```
void N_drawBoard (
             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.

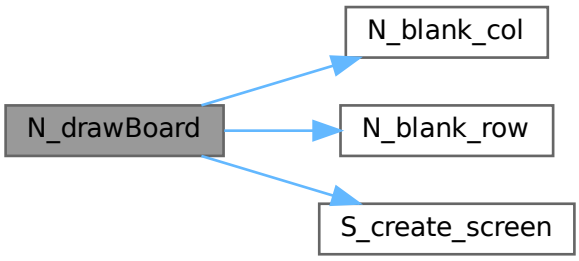
Definition at line 215 of file sudoku_ncurses.c.

```
00215                       {
00216
00217     WINDOW *s = S_create_screen();
00218     wbkgd(s,COLOR_PAIR(2));
00219
00220     for (size_t row = 0; row < BOARD_SIZE; row++){
00221         N_blank_row(row, s);
00222         for (size_t col = 0; col < BOARD_SIZE; col++){
00223             N_blank_col(col, s);
00224             if (SudokuGrid[row][col] == 0){
00225                 wprintw(s,"  ");
00226             } else {
00227             wprintw(s,"%d ", SudokuGrid[row][col]);
00228             }
00229         }
00230         wprintw(s,"\n");
00231     }
00232
00233     wrefresh(s); // updates current screen » standard screen
00234 }
```

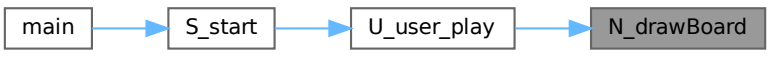
References BOARD_SIZE, N_blank_col(), N_blank_row(), S_create_screen(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller 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.

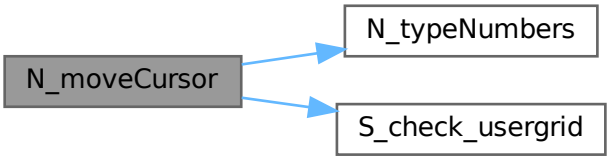
```
00256                       {
00257     int row, col, value;
00258     row = 0; col = 0;
00259
00260     curs_set(1);
00261     refresh();
00262
00263     int ch, y_loc, x_loc; // current character
00264     move(1,1);
00265
00266     getyx(stdscr, y_loc, x_loc);
00267
00268     while ((ch = getch()) != '\n'){
00269         switch(ch){
00270             case 'w': // move up
00271                     if (y_loc == 1){
00272                         break;
00273                     } else if (y_loc == 5 || y_loc == 9){
00274                         move(y_loc -= 2, x_loc);
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                         break;
00285                     } else if (y_loc == 3 || y_loc == 7){
00286                         move(y_loc += 2, x_loc);
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;
00297                     } else if (x_loc == 8 || x_loc == 15){
00298                         move(y_loc, x_loc -= 3);
00299                         col--;
00300                         break;
00301                     } else {
00302                         move(y_loc, x_loc -= 2);
00303                         col--;
00304                         break;
00305                     }
00306             case 'd':
00307                     if (x_loc == 19){
00308                         break;
00309                     } else if (x_loc == 5 || x_loc == 12){
00310                         move(y_loc, x_loc += 3);
00311                         col++;
00312                         break;
00313                     } else {
00314                         move(y_loc, x_loc += 2);
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
00325         //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00326
00327         // Put a version of solve sudoku here
00328         if (S_check_usergrid(SudokuGrid)){
00329             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 }
```

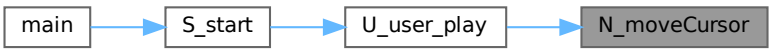
References N_typeNumbers(), S_check_usergrid(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



N_play_options()

```
int N_play_options (
            void )
```

SUbmenu.

Definition at line 150 of file sudoku_ncurses.c.

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

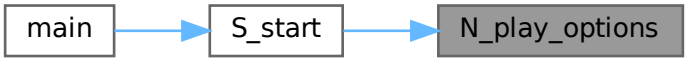
```
00162     for (size_t i = 0; i < 4; i++){
00163         if (play_menu_selection = i){
00164             attron(A_REVERSE);
00165         } else {
00166             attroff(A_REVERSE);
00167         }
00168         printw("%s", play_menu_Options[i]);
00169     }
00170 
00171     while ((ch = getch()) != '\n'){
00172 
00173         switch(ch){
00174             case 'w':
00175                 if (play_menu_selection == 0){
00176                     play_menu_selection = 3;
00177                 } else {
00178                     play_menu_selection--;
00179                 }
00180                 break;
00181             case 's':
00182                 if (play_menu_selection == 3){
00183                     play_menu_selection = 0;
00184                 } else {
00185                     play_menu_selection++;
00186                 }
00187                 break;
00188             default:
00189                 break;
00190         }
00191 
00192         move(4,0);
00193 
00194         for (size_t i = 0; i < 4; i++){
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:

Here is the caller 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.

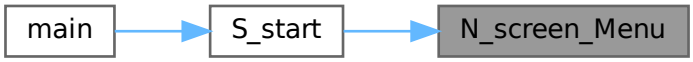
```
00084                         {
00085     int menu_selection = 0, ch;
00086     char *menu_Options[] = {
00087         " 1 Play Sudoku\n",
00088         " 2 Exit\n",
00089     };
00090
00091     attroff(A_REVERSE);
00092     N_screen_Title();
00093
00094     move(4,0);
00095
00096     for (size_t i = 0; i < 2; i++){
00097         if (menu_selection == i){
00098             attron(A_REVERSE);
00099         } else {
00100             attroff(A_REVERSE);
00101         }
00102         printw("%s", menu_Options[i]);
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){
00113             case 'w':
00114                 if (menu_selection == 0){
00115                     menu_selection = 1;
00116                 } else {
00117                     menu_selection--;
00118                 }
00119                 break;
00120             case 's':
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++){
00134             if (menu_selection == i){
00135                 attron(A_REVERSE);
00136             } else {
00137                 attroff(A_REVERSE);
00138             }
00139             printw("%s", menu_Options[i]);
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 )
```

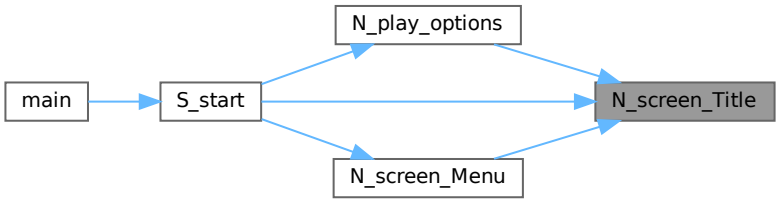
Function to display title.

Definition at line 65 of file sudoku_ncurses.c.

```
00065                           {
00066     int MenuChoice = 0;
00067
00068     move(0,1);
00069     attron(A_BOLD);
00070     attron(A_UNDERLINE);
00071     addstr("Sudoku\n");
00072     move(2,1);
00073     attroff(A_BOLD);
00074     attroff(A_UNDERLINE);
00075
00076     addstr("Please choose an option:\n\n");
00077 }
```

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

```
00381             addch('6' | A_REVERSE);
00382             break;
00383         case '7':
00384             SudokuGrid[row][col] = 7;
00385             addch('7' | A_REVERSE);
00386             break;
00387         case '8':
00388             SudokuGrid[row][col] = 8;
00389             addch('8' | A_REVERSE);
00390             break;
00391         case '9':
00392             SudokuGrid[row][col] = 9;
00393             addch('9' | A_REVERSE);
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 )
```

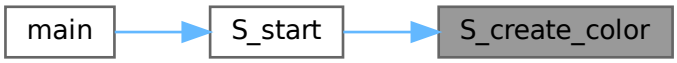
Colour options.

Definition at line 27 of file sudoku_ncurses.c.

```
00027                       {
00028     start_color();
00029
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 }
```

Referenced by S_start().

Here is the caller graph for this function:



S_create_screen()

```
WINDOW * S_create_screen (
            void )
```

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

in	*n*	Integer

Definition at line 42 of file sudoku_ncurses.c.

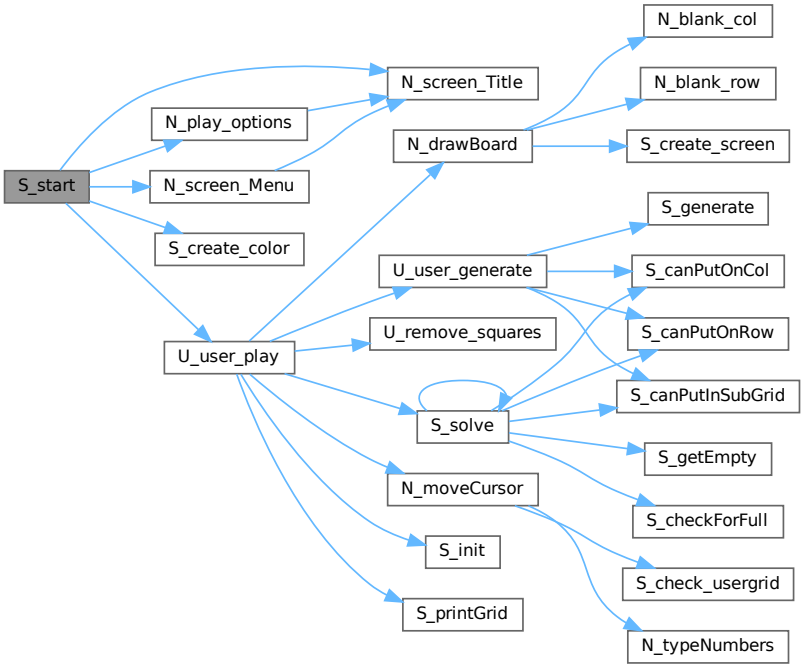
```
00042                   {
00043     S_create_color();
00044
00045     // menu options
00046     int Selection_Menu = 0;
00047     int Selection_Menu_Sub = 0;
00048
00049     N_screen_Title();
```

```
00050
00051     while(Selection_Menu != 1){
00052         Selection_Menu = N_screen_Menu(); // Main menu
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 }
```

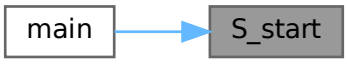
References N_play_options(), N_screen_Menu(), N_screen_Title(), S_create_color(), and U_user_play().

Referenced by main().

Here is the call graph for this function:



Here is the caller 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
00008
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();
00021     } else {
00022         return s_screen;
00023     }
00024 }
00025
00027 int S_create_color(void){
00028     start_color();
00029
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
00046     int Selection_Menu = 0;
00047     int Selection_Menu_Sub = 0;
00048
00049     N_screen_Title();
00050
00051     while(Selection_Menu != 1){
00052         Selection_Menu = N_screen_Menu(); // Main menu
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 }
00063
00065 void N_screen_Title(void){
00066     int MenuChoice = 0;
00067
00068     move(0,1);
00069     attron(A_BOLD);
00070     attron(A_UNDERLINE);
00071     addstr("Sudoku\n");
00072     move(2,1);
00073     attroff(A_BOLD);
00074     attroff(A_UNDERLINE);
00075
00076     addstr("Please choose an option:\n\n");
00077 }
00078
00084 int N_screen_Menu(void){
00085     int menu_selection = 0, ch;
00086     char *menu_Options[] = {
00087         " 1 Play Sudoku\n",
00088         " 2 Exit\n",
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++){
00097         if (menu_selection == i){
00098             attron(A_REVERSE);
00099         } else {
00100             attroff(A_REVERSE);
```

```
00101         }
00102         printw("%s", menu_Options[i]);
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){
00113             case 'w':
00114                 if (menu_selection == 0){
00115                     menu_selection = 1;
00116                 } else {
00117                     menu_selection--;
00118                 }
00119                 break;
00120             case 's':
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++){
00134             if (menu_selection == i){
00135                 attron(A_REVERSE);
00136             } else {
00137                 attroff(A_REVERSE);
00138             }
00139             printw("%s", menu_Options[i]);
00140         }
00141     }
00142 
00143     clear();
00144     refresh();
00145 
00146     return menu_selection;
00147 }
00148 
00150 int N_play_options(void){
00151     int play_menu_selection = 1, ch;
00152     char *play_menu_Options[] = {
00153         " 1 Easy\n",
00154         " 2 Normal\n",
00155         " 3 Hard\n",
00156         " 4 Back\n"
00157     };
00158 
00159     attroff(A_REVERSE);
00160     N_screen_Title();
00161 
00162     for (size_t i = 0; i < 4; i++){
00163         if (play_menu_selection = i){
00164             attron(A_REVERSE);
00165         } else {
00166             attroff(A_REVERSE);
00167         }
00168         printw("%s", play_menu_Options[i]);
00169     }
00170 
00171     while ((ch = getch()) != '\n'){
00172 
00173         switch(ch){
00174             case 'w':
00175                 if (play_menu_selection == 0){
00176                     play_menu_selection = 3;
00177                 } else {
00178                     play_menu_selection--;
00179                 }
00180                 break;
00181             case 's':
00182                 if (play_menu_selection == 3){
00183                     play_menu_selection = 0;
00184                 } else {
00185                     play_menu_selection++;
00186                 }
00187                 break;
00188             default:
```

```
00189                 break;
00190         }
00191
00192         move(4,0);
00193
00194         for (size_t i = 0; i < 4; i++){
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 }
00209
00215 void N_drawBoard(void){
00216
00217     WINDOW *s = S_create_screen();
00218     wbkgd(s,COLOR_PAIR(2));
00219
00220     for (size_t row = 0; row < BOARD_SIZE; row++){
00221         N_blank_row(row, s);
00222         for (size_t col = 0; col < BOARD_SIZE; col++){
00223             N_blank_col(col, s);
00224             if (SudokuGrid[row][col] == 0){
00225                 wprintw(s,"  ");
00226             } else {
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){
00270             case 'w': // move up
00271                     if (y_loc == 1){
00272                         break;
00273                     } else if (y_loc == 5 || y_loc == 9){
00274                         move(y_loc -= 2, x_loc);
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                         break;
00285                     } else if (y_loc == 3 || y_loc == 7){
00286                         move(y_loc += 2, x_loc);
```

```
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;
00297                       } else if (x_loc == 8 || x_loc == 15){
00298                           move(y_loc, x_loc -= 3);
00299                           col--;
00300                           break;
00301                       } else {
00302                           move(y_loc, x_loc -= 2);
00303                           col--;
00304                           break;
00305                       }
00306               case 'd':
00307                       if (x_loc == 19){
00308                           break;
00309                       } else if (x_loc == 5 || x_loc == 12){
00310                           move(y_loc, x_loc += 3);
00311                           col++;
00312                           break;
00313                       } else {
00314                           move(y_loc, x_loc += 2);
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
00325           //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00326
00327           // Put a version of solve sudoku here
00328           if (S_check_usergrid(SudokuGrid)){
00329               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 }
00341
00347 void N_typeNumbers(int ch, int row, int col){
00348       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;
00359           case '1':
00360               SudokuGrid[row][col] = 1;
00361               addch('1' | A_REVERSE);
00362               break;
00363           case '2':
00364               SudokuGrid[row][col] = 2;
00365               addch('2' | A_REVERSE);
00366               break;
00367           case '3':
00368               SudokuGrid[row][col] = 3;
00369               addch('3' | A_REVERSE);
00370               break;
00371           case '4':
00372               SudokuGrid[row][col] = 4;
00373               addch('4' | A_REVERSE);
00374               break;
00375           case '5':
00376               SudokuGrid[row][col] = 5;
00377               addch('5' | A_REVERSE);
00378               break;
```

```
00379         case '6':
00380             SudokuGrid[row][col] = 6;
00381             addch('6' | A_REVERSE);
00382             break;
00383         case '7':
00384             SudokuGrid[row][col] = 7;
00385             addch('7' | A_REVERSE);
00386             break;
00387         case '8':
00388             SudokuGrid[row][col] = 8;
00389             addch('8' | A_REVERSE);
00390             break;
00391         case '9':
00392             SudokuGrid[row][col] = 9;
00393             addch('9' | A_REVERSE);
00394             break;
00395         default:
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()

```
void N_blank_col (
            int n,
            WINDOW * s)
```

Insert a blank column when drawing grid

Parameters

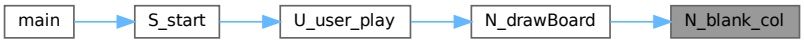
in	*n*	Col number

Definition at line 247 of file sudoku_ncurses.c.

```
00247                                   {
00248     if (n == 0 || n == 3 || n == 6){
00249         wprintw(s," ");
00250     }
00251         wrefresh(s);
00252 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



N_blank_row()

```
void N_blank_row (
            int n,
            WINDOW * s)
```

insert a blank row when drawing sudoku

Parameters

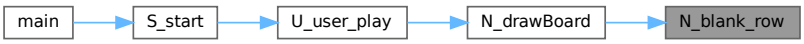
in	*n*	Row number

Definition at line 238 of file sudoku_ncurses.c.

```
00238                                   {
00239     if (n == 0 || n == 3 || n == 6) {
00240         wprintw(s,"\n");
00241     }
00242         wrefresh(s);
00243 }
```

Referenced by N_drawBoard().

Here is the caller graph for this function:



N_drawBoard()

```
void N_drawBoard (
             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.

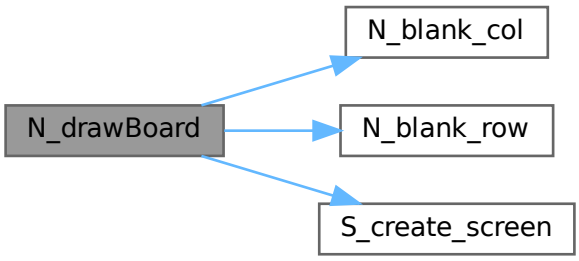
Definition at line 215 of file sudoku_ncurses.c.

```
00215                       {
00216
00217     WINDOW *s = S_create_screen();
00218     wbkgd(s,COLOR_PAIR(2));
00219
00220     for (size_t row = 0; row < BOARD_SIZE; row++){
00221         N_blank_row(row, s);
00222         for (size_t col = 0; col < BOARD_SIZE; col++){
00223             N_blank_col(col, s);
00224             if (SudokuGrid[row][col] == 0){
00225                 wprintw(s,"  ");
00226             } else {
00227             wprintw(s,"%d ", SudokuGrid[row][col]);
00228             }
00229         }
00230         wprintw(s,"\n");
00231     }
00232
00233     wrefresh(s); // updates current screen » standard screen
00234 }
```

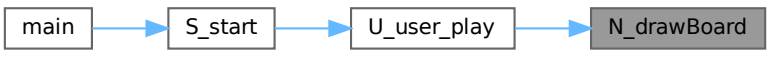
References BOARD_SIZE, N_blank_col(), N_blank_row(), S_create_screen(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller 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.

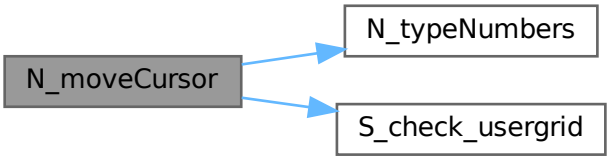
```
00256                       {
00257     int row, col, value;
00258     row = 0; col = 0;
00259
00260     curs_set(1);
00261     refresh();
00262
00263     int ch, y_loc, x_loc; // current character
00264     move(1,1);
00265
00266     getyx(stdscr, y_loc, x_loc);
00267
00268     while ((ch = getch()) != '\n'){
00269         switch(ch){
00270             case 'w': // move up
00271                     if (y_loc == 1){
00272                         break;
00273                     } else if (y_loc == 5 || y_loc == 9){
00274                         move(y_loc -= 2, x_loc);
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                         break;
00285                     } else if (y_loc == 3 || y_loc == 7){
00286                         move(y_loc += 2, x_loc);
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;
00297                     } else if (x_loc == 8 || x_loc == 15){
00298                         move(y_loc, x_loc -= 3);
00299                         col--;
00300                         break;
00301                     } else {
00302                         move(y_loc, x_loc -= 2);
00303                         col--;
00304                         break;
00305                     }
00306             case 'd':
00307                     if (x_loc == 19){
00308                         break;
00309                     } else if (x_loc == 5 || x_loc == 12){
00310                         move(y_loc, x_loc += 3);
00311                         col++;
00312                         break;
00313                     } else {
00314                         move(y_loc, x_loc += 2);
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
00325         //printw("Row: %d, Col: %d Value: %d\n", row,col, SudokuGrid[row][col]);
00326
00327         // Put a version of solve sudoku here
00328         if (S_check_usergrid(SudokuGrid)){
00329             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 }
```

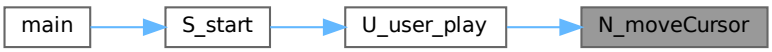
References N_typeNumbers(), S_check_usergrid(), and SudokuGrid.

Referenced by U_user_play().

Here is the call graph for this function:



Here is the caller graph for this function:



N_play_options()

```
int N_play_options (
             void )
```

SUbmenu.

Definition at line 150 of file sudoku_ncurses.c.

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

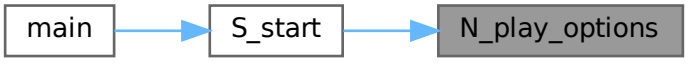
```
00162     for (size_t i = 0; i < 4; i++){
00163         if (play_menu_selection = i){
00164             attron(A_REVERSE);
00165         } else {
00166             attroff(A_REVERSE);
00167         }
00168         printw("%s", play_menu_Options[i]);
00169     }
00170 
00171     while ((ch = getch()) != '\n'){
00172 
00173         switch(ch){
00174             case 'w':
00175                 if (play_menu_selection == 0){
00176                     play_menu_selection = 3;
00177                 } else {
00178                     play_menu_selection--;
00179                 }
00180                 break;
00181             case 's':
00182                 if (play_menu_selection == 3){
00183                     play_menu_selection = 0;
00184                 } else {
00185                     play_menu_selection++;
00186                 }
00187                 break;
00188             default:
00189                 break;
00190         }
00191 
00192         move(4,0);
00193 
00194         for (size_t i = 0; i < 4; i++){
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:

Here is the caller 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.

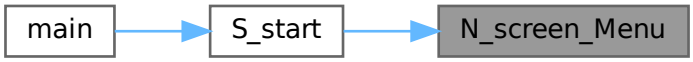
```
00084                         {
00085     int menu_selection = 0, ch;
00086     char *menu_Options[] = {
00087         " 1 Play Sudoku\n",
00088         " 2 Exit\n",
00089     };
00090
00091     attroff(A_REVERSE);
00092     N_screen_Title();
00093
00094     move(4,0);
00095
00096     for (size_t i = 0; i < 2; i++){
00097         if (menu_selection == i){
00098             attron(A_REVERSE);
00099         } else {
00100             attroff(A_REVERSE);
00101         }
00102         printw("%s", menu_Options[i]);
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){
00113             case 'w':
00114                 if (menu_selection == 0){
00115                     menu_selection = 1;
00116                 } else {
00117                     menu_selection--;
00118                 }
00119                 break;
00120             case 's':
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++){
00134             if (menu_selection == i){
00135                 attron(A_REVERSE);
00136             } else {
00137                 attroff(A_REVERSE);
00138             }
00139             printw("%s", menu_Options[i]);
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 )
```

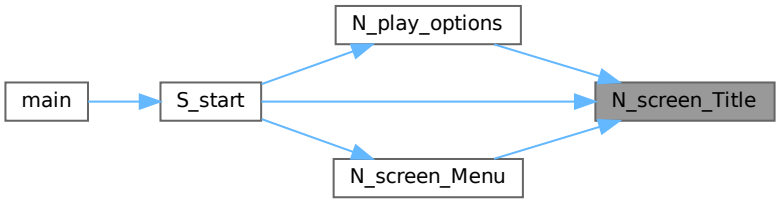
Function to display title.

Definition at line 65 of file sudoku_ncurses.c.

```
00065                           {
00066     int MenuChoice = 0;
00067
00068     move(0,1);
00069     attron(A_BOLD);
00070     attron(A_UNDERLINE);
00071     addstr("Sudoku\n");
00072     move(2,1);
00073     attroff(A_BOLD);
00074     attroff(A_UNDERLINE);
00075
00076     addstr("Please choose an option:\n\n");
00077 }
```

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

```
00381             addch('6' | A_REVERSE);
00382             break;
00383         case '7':
00384             SudokuGrid[row][col] = 7;
00385             addch('7' | A_REVERSE);
00386             break;
00387         case '8':
00388             SudokuGrid[row][col] = 8;
00389             addch('8' | A_REVERSE);
00390             break;
00391         case '9':
00392             SudokuGrid[row][col] = 9;
00393             addch('9' | A_REVERSE);
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 )
```

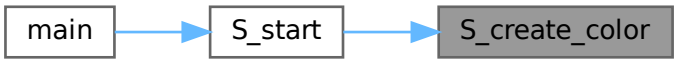
Colour options.

Definition at line 27 of file sudoku_ncurses.c.

```
00027                       {
00028     start_color();
00029
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 }
```

Referenced by S_start().

Here is the caller graph for this function:



S_create_screen()

```
WINDOW * S_create_screen (
            void )
```

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

in	*n*	Integer

Definition at line 42 of file sudoku_ncurses.c.

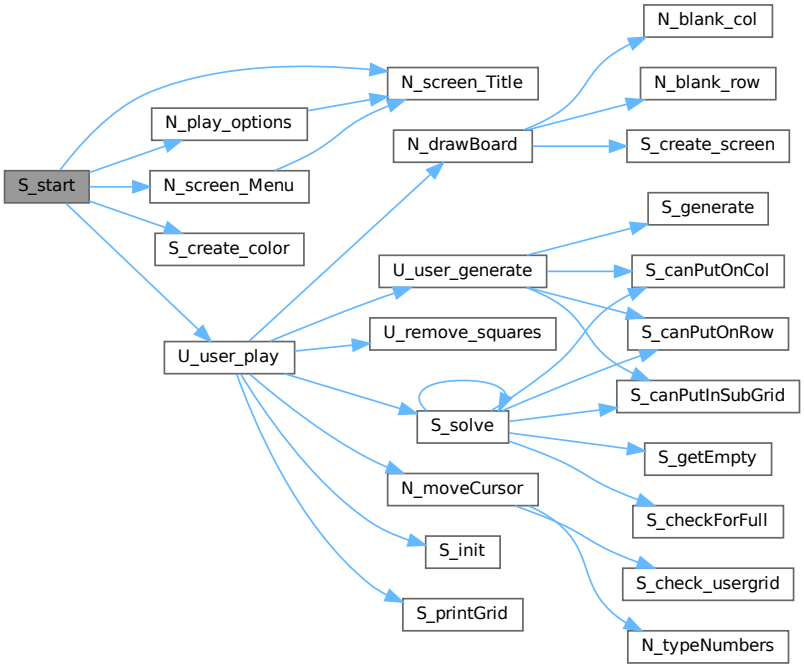
```
00042                     {
00043     S_create_color();
00044
00045     // menu options
00046     int Selection_Menu = 0;
00047     int Selection_Menu_Sub = 0;
00048
00049     N_screen_Title();
```

```
00050
00051     while(Selection_Menu != 1){
00052         Selection_Menu = N_screen_Menu(); // Main menu
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 }
```

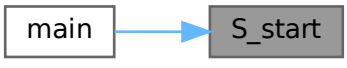
References N_play_options(), N_screen_Menu(), N_screen_Title(), S_create_color(), and U_user_play().

Referenced by main().

Here is the call graph for this function:



Here is the caller 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()

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

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.

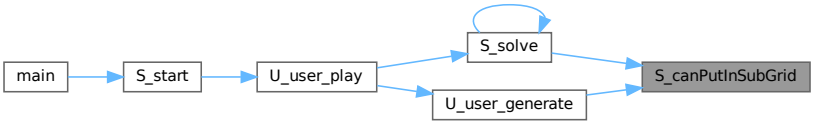
```
00154                                                {
00155     int len = BOARD_SIZE;
00156     int index = row * len + col;
00157     int x = sqrt(len);
00158     int box_index = (index % len) / x + x * (index / (len * x));
00159
00160     int row_start;
00161     int col_start;
00162
00163     switch (box_index){
00164         case 0: row_start = 0;
00165                 col_start = 0;
00166                 break;
00167         case 1: row_start = 0;
00168                 col_start = 3;
00169                 break;
00170         case 2: row_start = 0;
00171                 col_start = 6;
00172                 break;
00173         case 3: row_start = 3;
00174                 col_start = 0;
00175                 break;
00176         case 4: row_start = 3;
00177                 col_start = 3;
00178                 break;
00179         case 5: row_start = 3;
00180                 col_start = 6;
00181                 break;
00182         case 6: row_start = 6;
00183                 col_start = 0;
00184                 break;
```

```
00185         case 7: row_start = 6;
00186                 col_start = 3;
00187                 break;
00188         case 8: row_start = 6;
00189                 col_start = 6;
00190                 break;
00191     }
00192 
00193     for (size_t i = 0; i < 3 ; i++){
00194         for (size_t j = 0; j < 3; j++){
00195             if (SudokuGrid[row_start + i][col_start + j] == n){
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

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

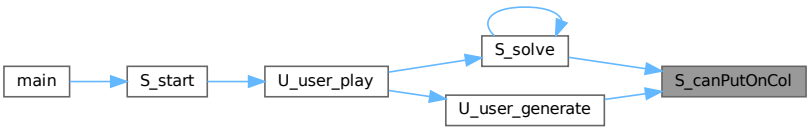
Definition at line 142 of file sudoku_solve.c.

```
00142                                   {
00143     for (size_t i = 0; i < BOARD_SIZE; i++){
00144         if (SudokuGrid[i][col] == n){
00145             return false;
00146         }
00147     }
00148     return true;
00149 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_canPutOnRow()

```
bool S_canPutOnRow (
            int row,
            int n)
```

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

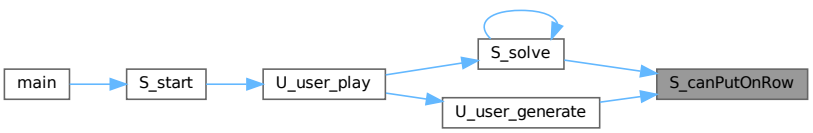
Definition at line 128 of file sudoku_solve.c.

```
00128                                   {
00129     for (size_t i = 0; i < BOARD_SIZE; i++){
00130         if (SudokuGrid[row][i] == n){
00131             return false;
00132         }
00133     }
00134     return true;
00135 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_check_usergrid()

```
bool S_check_usergrid (
            int grid[][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.

```
00020                                                {
00021     int count;
00022
00023     for(size_t row = 0; row < BOARD_SIZE; row++){
00024         count = 0;
00025         for (size_t col = 0; col < BOARD_SIZE; col++){
00026             count += grid[row][col];
00027         }
00028         if (count == 45){
00029             continue;
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()

```
bool S_checkForFull (
            int grid[][BOARD_SIZE])
```

Check if grid is full.

Parameters

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

Definition at line 110 of file sudoku_solve.c.

```
00110                                           {
00111     for (size_t i = 0; i < BOARD_SIZE; i++){
00112         for (size_t j = 0; j < BOARD_SIZE; j++){
00113             if (grid[i][j] == 0){
00114                 return false;
00115             }
00116         }
00117     }
00118     return true;
00119 }
```

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	*r	Pointer to integer representing a col
in,out	*r	Pointer to integer representing a number

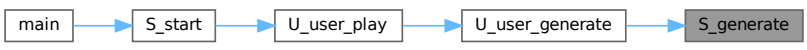
Definition at line 55 of file sudoku_solve.c.

```
00055                                           {
00056     *r = rand() % BOARD_SIZE;
00057     *c = rand() % BOARD_SIZE;
00058     *n = rand() % (9 - 1 + 1) + 1;
00059 }
```

References BOARD_SIZE.

Referenced by U_user_generate().

Here is the caller graph for this function:



S_getEmpty()

```
bool S_getEmpty (
            int * r,
            int * c)
```

find an empty grid square

Parameters

in,out	*r	pointer to an empty row
in,out	*c	pointer to an empty col

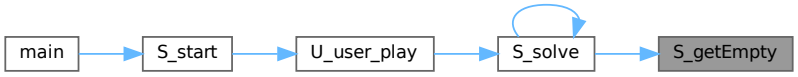
Definition at line 95 of file sudoku_solve.c.

```
00095                                   {
00096     for (size_t i = 0; i < BOARD_SIZE; i++){
00097         for (size_t j = 0; j < BOARD_SIZE; j++){
00098             if (SudokuGrid[i][j] == 0){
00099                 *r = i;
00100                 *c = j;
00101                 return true;
00102             }
00103         }
00104     }
00105     return false;
00106 }
```

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

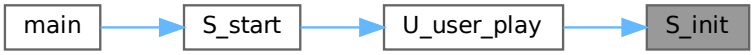
Definition at line 42 of file sudoku_solve.c.

```
00042                                   {
00043     for(size_t row = 0; row < BOARD_SIZE; row++){
00044         for (size_t col = 0; col < BOARD_SIZE; col++){
00045             grid[row][col] = 0;
00046         }
00047     }
00048 }
```

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.

```
00206                       {
00207     for (size_t i = 0; i < BOARD_SIZE; i++){
00208         for (size_t j = 0; j < BOARD_SIZE; j++){
00209             printf("%d ", SudokuGrid[i][j]);
00210         }
00211         printf("\n");
00212     }
00213 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by U_user_play().

Here is the caller graph for this function:

S_solve()

```
bool S_solve (
            int a[][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

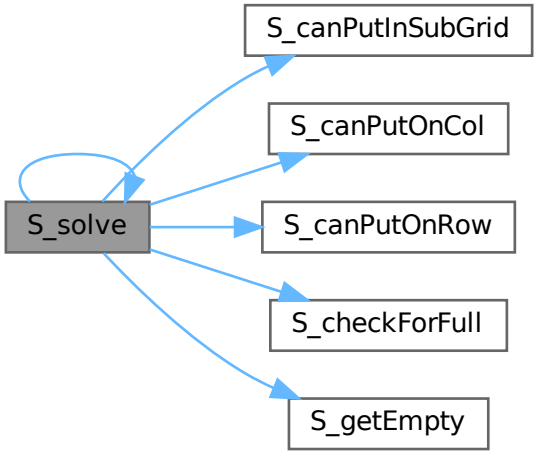
Definition at line 68 of file sudoku_solve.c.

```
00068                                   {
00069     int row, col;
00070
00071     S_getEmpty(&row, &col); // this needs to be counted
00072                             // doesn't matter which ones are removed at the end
00073
00074     if (S_checkForFull(a)){
00075         return true;
00076     }
00077
00078     for (size_t i = 1; i <= 9; i++){
00079         if (S_canPutOnRow(row,i) &&
00080                 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 }
```

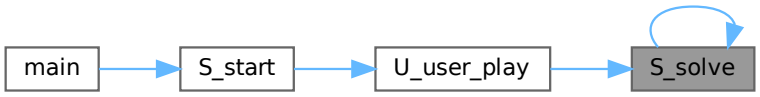
References BOARD_SIZE, S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_checkForFull(), S_getEmpty(), S_solve(), and SudokuGrid.

Referenced by S_solve(), and U_user_play().

Here is the call graph for this function:



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++){
00024         count = 0;
00025         for (size_t col = 0; col < BOARD_SIZE; col++){
00026             count += grid[row][col];
00027         }
00028         if (count == 45){
00029             continue;
00030         } else {
00031             return false;
00032         }
00033     }
00034     return true;
00035 }
00036
00042 void S_init(int grid[][BOARD_SIZE]){
00043     for(size_t row = 0; row < BOARD_SIZE; row++){
00044         for (size_t col = 0; col < BOARD_SIZE; col++){
00045             grid[row][col] = 0;
00046         }
00047     }
00048 }
00049
00055 void S_generate(int *r, int *c, int *n){
```

```
00056     *r = rand() % BOARD_SIZE;
00057     *c = rand() % BOARD_SIZE;
00058     *n = rand() % (9 - 1 + 1) + 1;
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
00072                             // doesn't matter which ones are removed at the end
00073
00074     if (S_checkForFull(a)){
00075         return true;
00076     }
00077
00078     for (size_t i = 1; i <= 9; i++){
00079         if (S_canPutOnRow(row,i) &&
00080                 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 }
00091
00095 bool S_getEmpty(int *r, int *c){
00096     for (size_t i = 0; i < BOARD_SIZE; i++){
00097         for (size_t j = 0; j < BOARD_SIZE; j++){
00098             if (SudokuGrid[i][j] == 0){
00099                 *r = i;
00100                 *c = j;
00101                 return true;
00102             }
00103         }
00104     }
00105     return false;
00106 }
00107
00110 bool S_checkForFull(int grid[][BOARD_SIZE]){
00111     for (size_t i = 0; i < BOARD_SIZE; i++){
00112         for (size_t j = 0; j < BOARD_SIZE; j++){
00113             if (grid[i][j] == 0){
00114                 return false;
00115             }
00116         }
00117     }
00118     return true;
00119 }
00120
00128 bool S_canPutOnRow(int row, int n){
00129     for (size_t i = 0; i < BOARD_SIZE; i++){
00130         if (SudokuGrid[row][i] == n){
00131             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++){
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;
00168                 col_start = 3;
00169                 break;
```

```
00170         case 2: row_start = 0;
00171                 col_start = 6;
00172                 break;
00173         case 3: row_start = 3;
00174                 col_start = 0;
00175                 break;
00176         case 4: row_start = 3;
00177                 col_start = 3;
00178                 break;
00179         case 5: row_start = 3;
00180                 col_start = 6;
00181                 break;
00182         case 6: row_start = 6;
00183                 col_start = 0;
00184                 break;
00185         case 7: row_start = 6;
00186                 col_start = 3;
00187                 break;
00188         case 8: row_start = 6;
00189                 col_start = 6;
00190                 break;
00191     }
00192
00193     for (size_t i = 0; i < 3 ; i++){
00194         for (size_t j = 0; j < 3; j++){
00195             if (SudokuGrid[row_start + i][col_start + j] == n){
00196                 return false;
00197             }
00198         }
00199     }
00200
00201     return true;
00202 }
00203
00206 void S_printGrid(void){
00207     for (size_t i = 0; i < BOARD_SIZE; i++){
00208         for (size_t j = 0; j < BOARD_SIZE; j++){
00209             printf("%d ", SudokuGrid[i][j]);
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.

Referenced by N_drawBoard(), S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_check_usergrid(), S_checkForFull(), S_generate(), S_getEmpty(), S_init(), S_printGrid(), S_solve(), and U_remove_squares().

2.9.2 Function Documentation

S_canPutInSubGrid()

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

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.

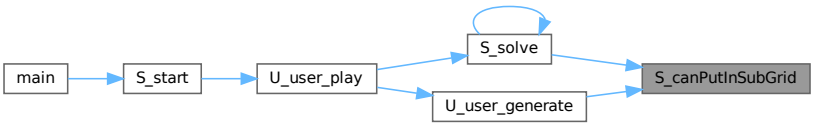
```
00154                                                   {
00155     int len = BOARD_SIZE;
00156     int index = row * len + col;
00157     int x = sqrt(len);
00158     int box_index = (index % len) / x + x * (index / (len * x));
00159 
00160     int row_start;
00161     int col_start;
00162 
00163     switch (box_index){
00164         case 0: row_start = 0;
00165                 col_start = 0;
00166                 break;
00167         case 1: row_start = 0;
00168                 col_start = 3;
00169                 break;
00170         case 2: row_start = 0;
00171                 col_start = 6;
00172                 break;
00173         case 3: row_start = 3;
00174                 col_start = 0;
00175                 break;
00176         case 4: row_start = 3;
00177                 col_start = 3;
00178                 break;
00179         case 5: row_start = 3;
00180                 col_start = 6;
00181                 break;
```

```
00182         case 6: row_start = 6;
00183                 col_start = 0;
00184                 break;
00185         case 7: row_start = 6;
00186                 col_start = 3;
00187                 break;
00188         case 8: row_start = 6;
00189                 col_start = 6;
00190                 break;
00191     }
00192
00193     for (size_t i = 0; i < 3 ; i++){
00194         for (size_t j = 0; j < 3; j++){
00195             if (SudokuGrid[row_start + i][col_start + j] == n){
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

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.

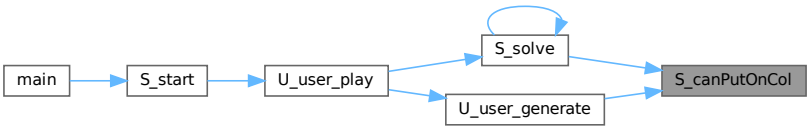
```
00142                                   {
00143     for (size_t i = 0; i < BOARD_SIZE; i++){
00144         if (SudokuGrid[i][col] == n){
00145             return false;
00146         }
00147     }
00148     return true;
```

```
00149 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_canPutOnRow()

```
bool S_canPutOnRow (
            int row,
            int n)
```

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

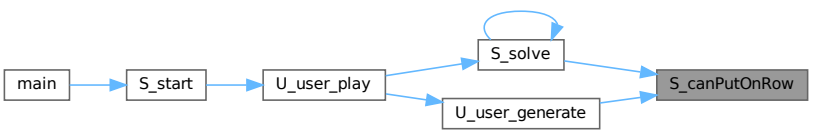
Definition at line 128 of file sudoku_solve.c.

```
00128                                   {
00129     for (size_t i = 0; i < BOARD_SIZE; i++){
00130         if (SudokuGrid[row][i] == n){
00131             return false;
00132         }
00133     }
00134     return true;
00135 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by S_solve(), and U_user_generate().

Here is the caller graph for this function:



S_check_usergrid()

```
bool S_check_usergrid (
            int grid[][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.

```
00020                                                {
00021     int count;
00022
00023     for(size_t row = 0; row < BOARD_SIZE; row++){
00024         count = 0;
00025         for (size_t col = 0; col < BOARD_SIZE; col++){
00026             count += grid[row][col];
00027         }
00028         if (count == 45){
00029             continue;
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()

```
bool S_checkForFull (
            int grid[][BOARD_SIZE])
```

Check if grid is full.

Parameters

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

Definition at line 110 of file sudoku_solve.c.

```
00110                                           {
00111     for (size_t i = 0; i < BOARD_SIZE; i++){
00112         for (size_t j = 0; j < BOARD_SIZE; j++){
00113             if (grid[i][j] == 0){
00114                 return false;
00115             }
00116         }
00117     }
00118     return true;
00119 }
```

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	*r	Pointer to integer representing a col
in,out	*r	Pointer to integer representing a number

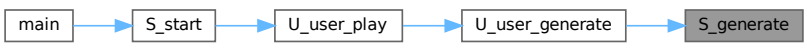
Definition at line 55 of file sudoku_solve.c.

```
00055                                           {
00056     *r = rand() % BOARD_SIZE;
00057     *c = rand() % BOARD_SIZE;
00058     *n = rand() % (9 - 1 + 1) + 1;
00059 }
```

References BOARD_SIZE.

Referenced by U_user_generate().

Here is the caller graph for this function:



S_getEmpty()

```
bool S_getEmpty (
            int * r,
            int * c)
```

find an empty grid square

Parameters

in,out	*r	pointer to an empty row
in,out	*c	pointer to an empty col

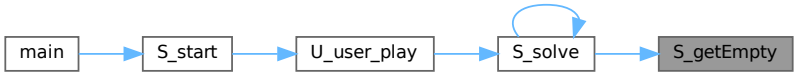
Definition at line 95 of file sudoku_solve.c.

```
00095                                   {
00096     for (size_t i = 0; i < BOARD_SIZE; i++){
00097         for (size_t j = 0; j < BOARD_SIZE; j++){
00098             if (SudokuGrid[i][j] == 0){
00099                 *r = i;
00100                 *c = j;
00101                 return true;
00102             }
00103         }
00104     }
00105     return false;
00106 }
```

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

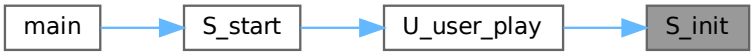
Definition at line 42 of file sudoku_solve.c.

```
00042                                   {
00043     for(size_t row = 0; row < BOARD_SIZE; row++){
00044         for (size_t col = 0; col < BOARD_SIZE; col++){
00045             grid[row][col] = 0;
00046         }
00047     }
00048 }
```

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.

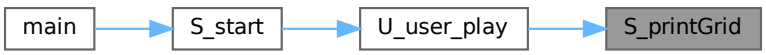
Definition at line 206 of file sudoku_solve.c.

```
00206                       {
00207     for (size_t i = 0; i < BOARD_SIZE; i++){
00208         for (size_t j = 0; j < BOARD_SIZE; j++){
00209             printf("%d ", SudokuGrid[i][j]);
00210         }
00211         printf("\n");
00212     }
00213 }
```

References BOARD_SIZE, and SudokuGrid.

Referenced by U_user_play().

Here is the caller graph for this function:



S_solve()

```
bool S_solve (
             int a[][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

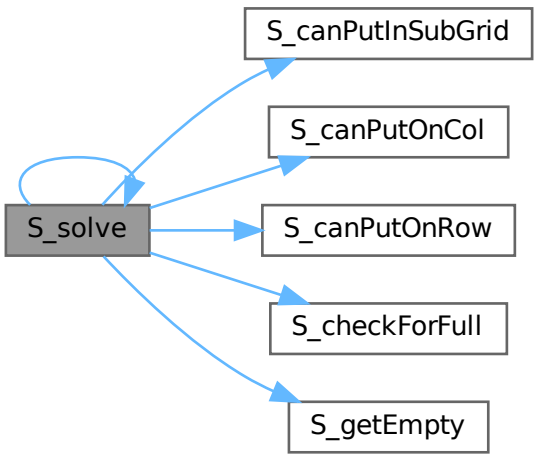
Definition at line 68 of file sudoku_solve.c.

```
00068                                   {
00069     int row, col;
00070
00071     S_getEmpty(&row, &col); // this needs to be counted
00072                             // doesn't matter which ones are removed at the end
00073
00074     if (S_checkForFull(a)){
00075         return true;
00076     }
00077
00078     for (size_t i = 1; i <= 9; i++){
00079         if (S_canPutOnRow(row,i) &&
00080                 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 }
```

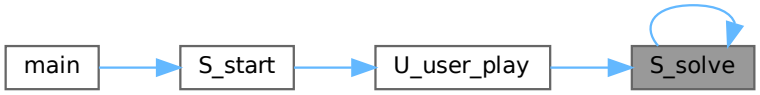
References BOARD_SIZE, S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_checkForFull(), S_getEmpty(), S_solve(), and SudokuGrid.

Referenced by S_solve(), and U_user_play().

Here is the call graph for this function:



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

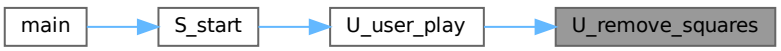
Definition at line 79 of file sudoku_user.c.

```
00079                                {
00080     int rand_row, rand_col;
00081     for (size_t i = 0; i <= n; i++){
00082         rand_row = rand() % BOARD_SIZE; rand_col = rand() % BOARD_SIZE;
00083         SudokuGrid[rand_row][rand_col] = 0;
00084     }
00085 }
```

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	*n*	Number of squares to fill.

Definition at line 52 of file sudoku_user.c.

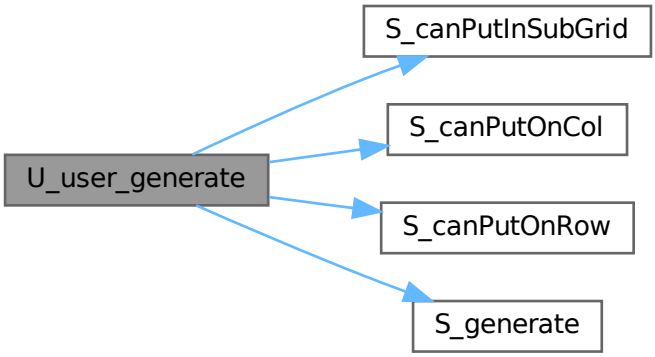
```
00052                            {
00053     int count, row, col, number;
00054     count = 0;
00055
00056     while (count < n){
00057         S_generate(&row, &col, &number);
00058         if (SudokuGrid[row][col] != 0){
00059             continue;
00060         } else if(
00061                     S_canPutOnRow(row,number) &&
00062                     S_canPutOnCol(col,number) &&
00063                     S_canPutInSubGrid(row, col, number)){
00064             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 {
00070             continue;
```

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

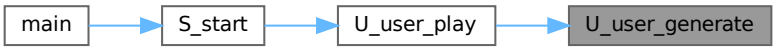
References S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_generate(), and SudokuGrid.

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 (
            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

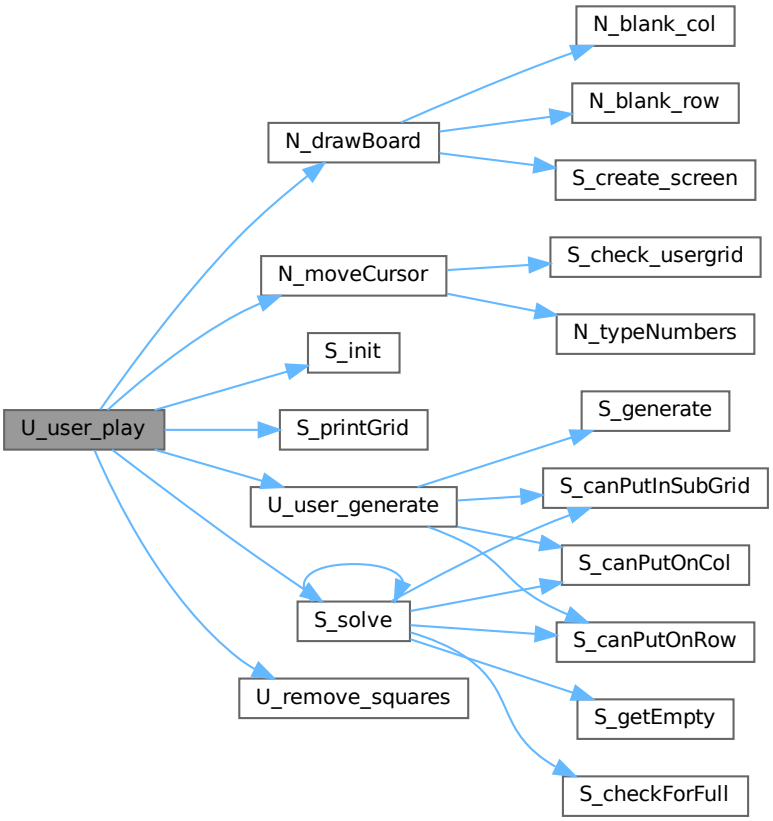
Definition at line 16 of file sudoku_user.c.

```
00016                       {
00017     int squaresToRemove = 0;
00018
00019     switch(n){
00020         case 0: squaresToRemove = EASY;
00021                 break;
00022         case 1: squaresToRemove = NORMAL;
00023                 break;
00024         case 2: squaresToRemove = HARD;
00025                 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 {
00041         puts("Not solvable");
00042     }
00043
00044     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:

2.12 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){
00017     int squaresToRemove = 0;
00018
00019     switch(n){
00020         case 0: squaresToRemove = EASY;
00021                 break;
00022         case 1: squaresToRemove = NORMAL;
00023                 break;
00024         case 2: squaresToRemove = HARD;
00025                 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 {
00041         puts("Not solvable");
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){
00057         S_generate(&row, &col, &number);
00058         if (SudokuGrid[row][col] != 0){
00059             continue;
00060         } else if(
00061                     S_canPutOnRow(row,number) &&
00062                     S_canPutOnCol(col,number) &&
00063                     S_canPutInSubGrid(row, col, number)){
00064             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 {
00070             continue;
00071         }
00072     }
00073 }
00074
00079 void U_remove_squares(int n){
00080     int rand_row, rand_col;
00081     for (size_t i = 0; i <= n; i++){
00082         rand_row = rand() % BOARD_SIZE; rand_col = rand() % BOARD_SIZE;
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	*n*	Number of squares to remove

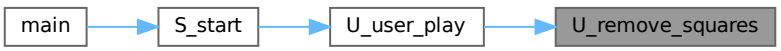
Definition at line 79 of file sudoku_user.c.

```
00079                                {
00080     int rand_row, rand_col;
00081     for (size_t i = 0; i <= n; i++){
00082         rand_row = rand() % BOARD_SIZE; rand_col = rand() % BOARD_SIZE;
00083         SudokuGrid[rand_row][rand_col] = 0;
00084     }
00085 }
```

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	*n*	Number of squares to fill.

Definition at line 52 of file sudoku_user.c.

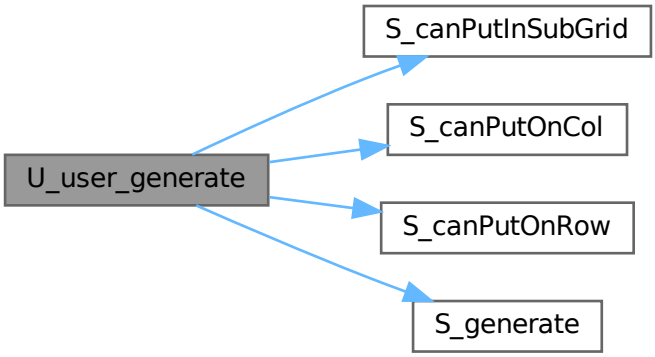
```
00052                            {
00053     int count, row, col, number;
00054     count = 0;
00055
00056     while (count < n){
00057         S_generate(&row, &col, &number);
00058         if (SudokuGrid[row][col] != 0){
00059             continue;
00060         } else if(
00061                     S_canPutOnRow(row,number) &&
00062                     S_canPutOnCol(col,number) &&
00063                     S_canPutInSubGrid(row, col, number)){
00064             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 {
00070             continue;
```

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

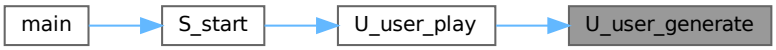
References S_canPutInSubGrid(), S_canPutOnCol(), S_canPutOnRow(), S_generate(), and SudokuGrid.

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 (
            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

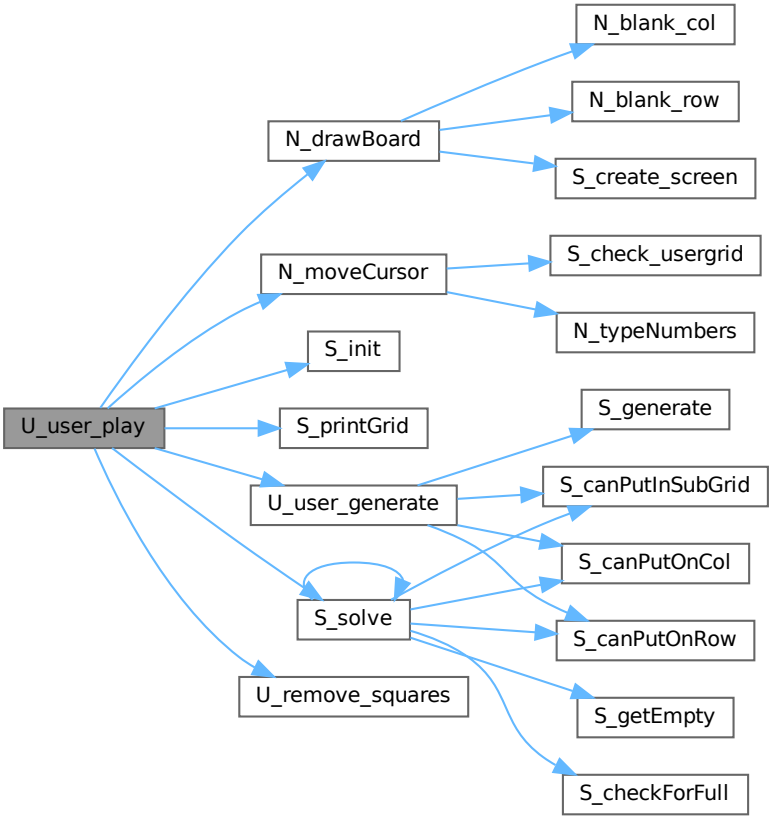
Definition at line 16 of file sudoku_user.c.

```
00016                       {
00017     int squaresToRemove = 0;
00018
00019     switch(n){
00020         case 0: squaresToRemove = EASY;
00021                 break;
00022         case 1: squaresToRemove = NORMAL;
00023                 break;
00024         case 2: squaresToRemove = HARD;
00025                 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 {
00041         puts("Not solvable");
00042     }
00043
00044     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

Go to the documentation of this file.

```
00001
00002 #ifndef SUDOOKU_USER_H
00003 #define SUDOOKU_USER_H
00004
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);
00036
00037 #endif
```

Index