Projet EFREI - C - Rubik's CUBE - 2021

Mathieu CHANTOT et Clément LE STRAT

Chapitre 1

Doxygen

— Doxygen permet de générer de la documentation automatiquement.
— Le logiciel se base sur le code et ses commentaires pour générer la doc.

1.1 Installation

1.1.1 Ubuntu

— Installer dans un terminal : `sudo apt install doxygen`
— Pour générer un PDF à partir de l'export Latex, il faut ajouter les packages : `sudo apt install texlive-latex-extra texlive-latex-recommended texlive-pictures graphviz texlive-lang-french`

1.1.2 Manjaro

— Installer dans un terminal : `pamac install doxygen`
— Pour générer un PDF à partir de l'export Latex, il faut ajouter le package texlive-latexextra : `pamac install texlive-latexextra`

1.2 Utilisation et configuration

— Doxygen se base sur un fichier de configuration (et des fichiers de types, mais que nous n'utilisons pas ici).
— Le premier fichier de configuration peut se créer via un Wizard : `doxywizard`
— Le fichier de configuration s'appelle, par défaut, Doxyfile.
— Une fois le fichier créé, on peut facilement le modifier dans un éditeur de texte quelconque, y compris dans Clion.
— Un fichier Doxyfile a été créé pour ce projet, ainsi qu'un fichier header.tex que ne nous utilisons pour changer le pied de page du document PDF.
— Pour créer un fichier vierge header.tex (pour le modifier en fonction de nos besoins), il faut faire `doxygen -w latex header.tex footer.tex doxygen.sty`. Il suffit de récupérer le fichier header.tex, le déplacer dans `Doc\doxygen\` et le modifier en conséquence.
— Enfin, on référence le fichier header.tex dans le fichier Doxyfile : `LATEX_HEADER = header.tex`

1.3 Génération de la documentation :

- Se déplacer dans le répertoire Doc/doxygen du projet.
- Et lancer la commande `doxygen`
- La commande va générer la documentation dans Doc/doxygen/output.
- La commande va sortir des logs, dont potentiellement des erreurs. Il est important de vérifier.
- Elle est, pour le moment, aux formats HTML, et Latex.
- Pour lire, par exemple, la documentation au format HTML, il suffit d'aller dans le répertoire ./Doc/doxygen/↩ Ouput/html et d'ouvrir le fichier index.html. Le résultat va s'ouvrir dans le navigateur par défaut.
- Pour générer une documentation PDF, aller dans le répertoire Latex `cd Doc/doxygen/output/latex` et lancer la commande `make`. Il suffira d'ouvrir le fichier refman.pdf généré dans ce même répertoire.

1.4 Références

- Using Doxygen
- Liens vers divers articles
- Examples de documentation
- Documentation du code avec Doxygen

Chapitre 2

Références et liens utiles au projet rubiks_efrei

2.1 Liens utiles et debuggage

- Formation C OpenClassRoom
- Gestion des conflits de variables dans fichier .h
- Nous développons sous Linux, en utilisant la librairie ncurses. De base, Clion ne peut pas debugger des programmes C utilisant ncurses :
  - https://youtrack.jetbrains.com/issue/CPP-822
  - https://youtu.be/2R3OSGmUmoU?t=2015
- Documentation pour debugger et utiliser ncurses dans Clion est dans le répertoire Doc
- Comment ajouter des librairies dans Clion pour pouvoir compiler : Lien

2.2 Documentation et aides nCurses

- Tutoriel ncurses
- Ncurses Programming Guide
- NCURSES Programming HOWTO Menus

2.3 CMake

- Dans notre cas, pour l'instant, le fichier CMakeLists.txt doit contenir, si le nom du projet est XXXX :
  cmake_minimum_required(VERSION 3.20)
  project(XXXX C)
  set(CMAKE_C_STANDARD 99)
  add_executable(XXXX main.c)
  target_link_libraries(XXXX curses)
  target_link_libraries(XXXX menu)

2.4 Librairies

- Librairie standard pour gérer des menus (basée sur ncurses)

2.5 Documentation avec Doxygen

- Comment documenter son code, avec Doxygen dans Clion
- Comment utiliser Doxygen pour générer de la documentation
- Doc officielle

Chapitre 3

Index des structures de données

3.1 Structures de données

Liste des structures de données avec une brève description :

Chapitre 4

Index des fichiers

4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Chapitre 5

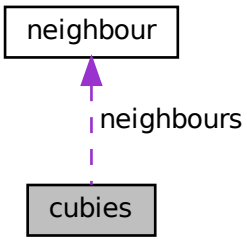
Documentation des structures de données

5.1 Référence de la structure cubies

Définition des cubies, qui sont les petits cubes de couleur rattachés à une face.

`#include <rubiks.h>`

Graphe de collaboration de cubies:



Champs de données

- int x
- int y
- int num
- T_CUBIE_TYPE type
- T_COLOR color
- T_COLOR cubie_side
- neighbour neighbours [2]

5.1.1 Description détaillée

Définition des cubies, qui sont les petits cubes de couleur rattachés à une face.

Paramètres

x	coordonnée horizontale du cubie
y	coordonnée verticale du cubie
num	Numéro du cubie dans la face : de 0 à 8
type	Type de cubie, CORNER, EDGE ou CENTER
color	Couleur du cubie
cubie_side	Face de rattachement du cubie
neighbours[2]	Définition des faces adjacentes au cubie. Il peut y en avoir 2 pour un CORNER et 1 pour un EDGE

Définition à la ligne 46 du fichier rubiks.h.

5.1.2 Documentation des champs

5.1.2.1 color

```
T_COLOR color
```

Définition à la ligne 51 du fichier rubiks.h.

5.1.2.2 cubie_side

```
T_COLOR cubie_side
```

Définition à la ligne 52 du fichier rubiks.h.

5.1.2.3 neighbours

```
neighbour neighbours[2]
```

Il peut y avoir 1 ou deux cubies adjacent.

Définition à la ligne 53 du fichier rubiks.h.

5.1.2.4 num

```
int num
```

Définition à la ligne 49 du fichier rubiks.h.

5.1.2.5 type

```
T_CUBIE_TYPE type
```

Définition à la ligne 50 du fichier rubiks.h.

5.1.2.6 x

```
int x
```

Définition à la ligne 47 du fichier rubiks.h.

5.1.2.7 y

```
int y
```

Définition à la ligne 48 du fichier rubiks.h.

La documentation de cette structure a été générée à partir du fichier suivant :
— rubiks.h

5.2 Référence de la structure neighbour

Définition des adjacents à un cubie.

```
#include <rubiks.h>
```

Champs de données

— T_COLOR num_side
— int num_cubie

5.2.1 Description détaillée

Définition des adjacents à un cubie.

Paramètres

num_side	Face de rattachement du voisin
num_cubie	Numéro du cubie dans sa face de rattachement. De 0 à 8.

Définition à la ligne 31 du fichier rubiks.h.

5.2.2 Documentation des champs

5.2.2.1 num_cubie

```
int num_cubie
```

Définition à la ligne 33 du fichier rubiks.h.

5.2.2.2 num_side

```
T_COLOR num_side
```

Définition à la ligne 32 du fichier rubiks.h.

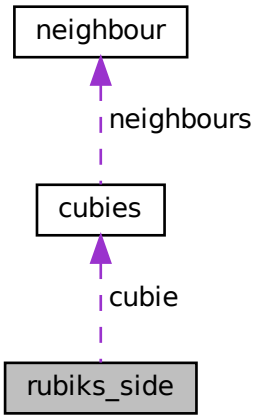
La documentation de cette structure a été générée à partir du fichier suivant :
— rubiks.h

5.3 Référence de la structure rubiks_side

Définition d'une face du Rubik's Cube.

```
#include <rubiks.h>
```

Graphe de collaboration de rubiks_side:



Champs de données

— T_COLOR neighbour_side [4]
— T_COLOR opposite_side
— T_COLOR side
— cubies cubie [9]

5.3.1 Description détaillée

Définition d'une face du Rubik's Cube.

Paramètres

neighbour_side[4]	Une face a 4 faces adjacentes, UP, DOWN, LEFT, et RIGHT.
opposite_side	Définition de la face opposée.
side	Définition du numéro et couleur de la face. Soit WHITE, ORANGE, GREEN, RED, BLUE, ou YELLOW
cubie[9]	Liaison vers les 9 cubies composants une face

Définition à la ligne 63 du fichier rubiks.h.

5.3.2 Documentation des champs

5.3.2.1 cubie

```
cubies cubie[9]
```

Définition à la ligne 67 du fichier rubiks.h.

5.3.2.2 neighbour_side

```
T_COLOR neighbour_side[4]
```

Définition à la ligne 64 du fichier rubiks.h.

5.3.2.3 opposite_side

```
T_COLOR opposite_side
```

Définition à la ligne 65 du fichier rubiks.h.

5.3.2.4 side

```
T_COLOR side
```

Définition à la ligne 66 du fichier rubiks.h.

La documentation de cette structure a été générée à partir du fichier suivant :
— rubiks.h

5.4 Référence de la structure solutions_steps

```
#include <moves_rubiks.h>
```

Graphe de collaboration de solutions_steps:

Champs de données

- char solution_step [SIZE_MOVE]
- solutions_steps * next_step

5.4.1 Description détaillée

Définition à la ligne 14 du fichier moves_rubiks.h.

5.4.2 Documentation des champs

5.4.2.1 next_step

```
solutions_steps* next_step
```

Définition à la ligne 16 du fichier moves_rubiks.h.

5.4.2.2 solution_step

```
char solution_step[SIZE_MOVE]
```

Définition à la ligne 15 du fichier moves_rubiks.h.

La documentation de cette structure a été générée à partir du fichier suivant :
- moves_rubiks.h

Chapitre 6

Documentation des fichiers

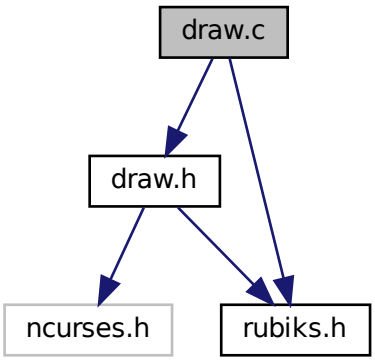
6.1 Référence du fichier Doxygen-installation-usage.md

6.2 Référence du fichier draw.c

Dans ce fichier, on gère les différents affichages du Rubik's Cube.

```
#include "draw.h"
#include "rubiks.h"
```

Graphe des dépendances par inclusion de draw.c:



Fonctions

- void detect_resize (__attribute__((unused)) int dummy)
- void set_colors (void)
- int check_and_set_term (void)
- void change_color (short line, short col, short cube_x, short cube_y, short color)
- void create_rubik_side (short line, short col, short color, char name)
- void create_board (void)
- void destroy_board (void)
- void rubiks_display (rubiks_side *rubiks)
- void draw_rubiks (rubiks_side *rubiks)

Variables

- WINDOW * BOARD [SQUARES]
- const short BLACK_ON_WHITE = 1
- const short BLACK_ON_ORANGE = 2
- const short BLACK_ON_GREEN = 3
- const short BLACK_ON_RED = 4
- const short BLACK_ON_BLUE = 5
- const short BLACK_ON_YELLOW = 6
- short NB_LINES
- short NB_COLS

6.2.1 Description détaillée

Dans ce fichier, on gère les différents affichages du Rubik's Cube.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.2.2 Documentation des fonctions

6.2.2.1 change_color()

```
void change_color (
            short line,
            short col,
            short cube_x,
            short cube_y,
            short color )
```

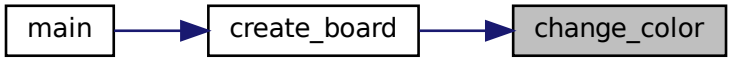
Fonction de test pour nCurses, qui permet de déplacer une couleur à partir de :
- line, col coordonnées du cube à modifier dans la matrice de l'écran
- cube_x, cube_y coordonnées x, y du cube à modifier
- color : Nouvelle couleur à appliquer

Paramètres

line	Ligne de la matrices nCurses
col	Colonne de la matrice nCurses
cube_x	Coordonnée X du cube à modifier
cube_y	Coordonnée Y du cube à modifier
color	Nouvelle couleur à appliquer

Définition à la ligne 115 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.2.2.2 check_and_set_term()

```
int check_and_set_term (
            void  )
```

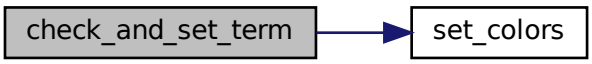
Vérification du terminal, pour voir si il pourra faire tourner le programme avec nCurses

Renvoie

0 si OK, sinon retourne une erreur différente de 0

Définition à la ligne 74 du fichier draw.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

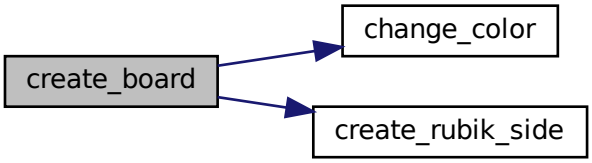
6.2.2.3 create_board()

```
void create_board (
             void  )
```

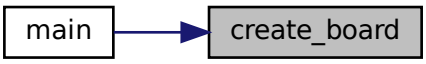
Fonction de test qui permet d'afficher diverses choses sur l'écran nCurses

Définition à la ligne 186 du fichier draw.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.2.2.4 create_rubik_side()

```
void create_rubik_side (
             short line,
             short col,
             short color,
             char name )
```

Fonction de test nCurses pour créer la face uniforme d'un cube dans une couleur donnée

Paramètres

line	Ligne de la matrice nCurses dans laquelle dessiner le cube
col	Colonne de la matrice nCurses dans laquelle dessiner le cube
color	Couleur à appliquer sur le cube
name	Nom de la face, à afficher dans le cubie central

Définition à la ligne 144 du fichier draw.c.

Voici le graphe des appelants de cette fonction :

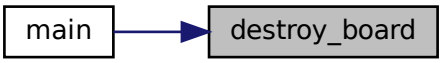
6.2.2.5 destroy_board()

```
void destroy_board (
            void  )
```

Fonction de test liée à create_board, qui permet de libérer les ressources et la mémoire.
Attention, cette fonction n'est pas à jour et ne libère pas correctement les choses

Définition à la ligne 234 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.2.2.6 detect_resize()

```
void detect_resize (
            __attribute__((unused)) int dummy )
```

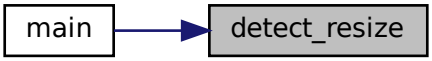
Détection de l'événement de changement de taille d'une fenêtre. L'évenement est non géré, pour le moment.

Paramètres

dummy	Paramètre inutilisé. Il est passé par la fonction d'appel des évènements sur une fenêtre

Définition à la ligne 50 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.2.2.7 draw_rubiks()

```
void draw_rubiks (
            rubiks_side * rubiks )
```

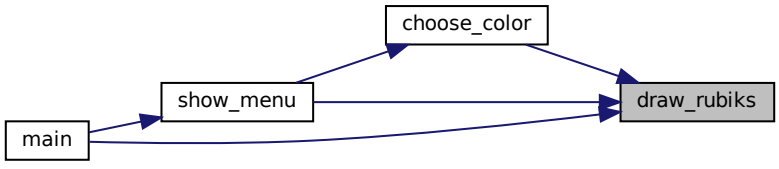
Dessine le rubik's Cube au format texte ANSI (pas au format nCurses)

Paramètres

rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 275 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.2.2.8 rubiks_display()

```
void rubiks_display (
            rubiks_side * rubiks )
```

Fonction de débuggage, permet d'afficher les positions après des mouvements.

Paramètres

rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 250 du fichier draw.c.

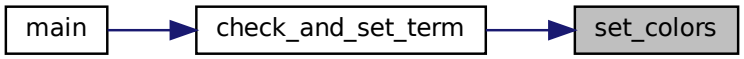
6.2.2.9 set_colors()

```
void set_colors (
             void  )
```

Définitions des couleurs pour nCurses

Définition à la ligne 58 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.2.3 Documentation des variables

6.2.3.1 BLACK_ON_BLUE

```
const short BLACK_ON_BLUE = 5
```

Écriture noire sur bleu, pour nCurses

Définition à la ligne 33 du fichier draw.c.

6.2.3.2 BLACK_ON_GREEN

```
const short BLACK_ON_GREEN = 3
```

Écriture noire sur vert, pour nCurses

Définition à la ligne 25 du fichier draw.c.

6.2.3.3 BLACK_ON_ORANGE

```
const short BLACK_ON_ORANGE = 2
```

Écriture noire sur orange, pour nCurses

Définition à la ligne 21 du fichier draw.c.

6.2.3.4 BLACK_ON_RED

```
const short BLACK_ON_RED = 4
```

Écriture noire sur rouge, pour nCurses

Définition à la ligne 29 du fichier draw.c.

6.2.3.5 BLACK_ON_WHITE

```
const short BLACK_ON_WHITE = 1
```

Écriture noire sur blanc, pour nCurses

Définition à la ligne 17 du fichier draw.c.

6.2.3.6 BLACK_ON_YELLOW

```
const short BLACK_ON_YELLOW = 6
```

Écriture noire sur jaune, pour nCurses

Définition à la ligne 37 du fichier draw.c.

6.2.3.7 BOARD

```
WINDOW* BOARD[SQUARES]
```

Définition à la ligne 12 du fichier draw.c.

6.2.3.8 NB_COLS

```
short NB_COLS
```

Définition à la ligne 43 du fichier draw.c.

6.2.3.9 NB_LINES

```
short NB_LINES
```

Taille de l'écran nCurses

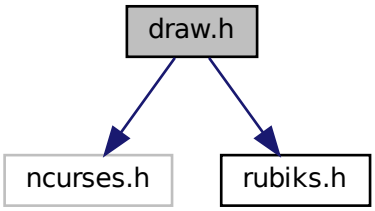
Définition à la ligne 42 du fichier draw.c.

6.3 Référence du fichier draw.h

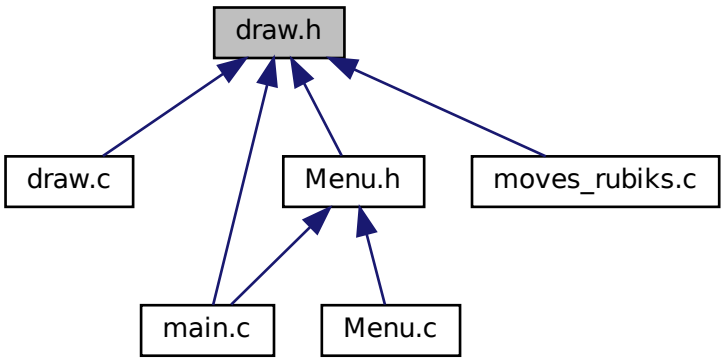
Dans ce fichier, on trouve les headers utilisés dans draw.c.

```
#include <ncurses.h>
#include "rubiks.h"
```

Graphe des dépendances par inclusion de draw.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

- #define RUBIK_LINES 3
  Permet de déterminer le nombre de lignes dans la grande matrice nCurses. Il y a trois lignes.
- #define RUBIK_COLS 4
  Permet de déterminer le nombre de colonnes dans la grande matrice nCurses. Il y a quatre colonnes.
- #define SQ_HEIGHT 4
  Hauteur en nombre de caractères d'un cubie avec nCurses.
- #define SQ_WIDTH 8
  Largeur en nombre de caractères d'un cubie avec nCurses.
- #define SQUARES ((RUBIK_LINES ∗ 3) ∗ (RUBIK_COLS ∗ 3))
  Nombre de carrés nCurses dans un Rubik's cube. Ne servira probablement pas.
- #define MIN_COLORS_NUMBER 256
  Nombre de couleurs gérés par le terminal (nCurses), nécessaire pour faire tourner le programme.
- #define TERM_NOT_BIG_ENOUGH 1
  Le terminal (nCurses) n'est pas assez grand pour afficher le Rubik's cube avec nCurses.
- #define TERM_HAS_NO_COLORS 2
  Le terminal (nCurses) ne gère pas les couleurs.
- #define TERM_HAS_NOT_ENOUGH_COLORS 3
  Le terminal (nCurses) ne gère pas assez de couleurs.
- #define ARED "\x1B[31m"
- #define GRN "\033[38;2;0;175;0m"
- #define YEL "\033[38;2;255;255;0m"
- #define BLU "\x1B[34m"
- #define WHT "\033[38;2;255;255;255m"
- #define ORG "\033[38;2;255;165;0m"
- #define GRY "\033[38;2;80;80m"
- #define RESET "\x1B[0m"

Fonctions

- int check_and_set_term (void)
- void set_colors (void)
- void change_color (short, short, short, short, short)
- void create_rubik_side (short, short, short, char)
- void create_board (void)
- void destroy_board (void)
- void detect_resize (__attribute__((unused)) int)
- void rubiks_display (rubiks_side ∗rubiks)
- void draw_rubiks (rubiks_side ∗rubiks)

Variables

- WINDOW ∗ BOARD [SQUARES]
- const short BLACK_ON_WHITE
- const short BLACK_ON_ORANGE
- const short BLACK_ON_GREEN
- const short BLACK_ON_RED
- const short BLACK_ON_BLUE
- const short BLACK_ON_YELLOW
- short NB_LINES
- short NB_COLS

6.3.1 Description détaillée

Dans ce fichier, on trouve les headers utilisés dans draw.c.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.3.2 Documentation des macros

6.3.2.1 ARED

```
#define ARED "\x1B[31m"
```

Les couleurs ANSI pour l'affichage du cube au format texte (non nCurses) Références : https://en.wikipedia.org/wiki/ANSI_escape_code

Définition à la ligne 81 du fichier draw.h.

6.3.2.2 BLU

```
#define BLU "\x1B[34m"
```

Définition à la ligne 86 du fichier draw.h.

6.3.2.3 GRN

```
#define GRN "\033[38;2;0;175;0m"
```

Définition à la ligne 83 du fichier draw.h.

6.3.2.4 GRY

```
#define GRY "\033[38;2;80;80m"
```

Définition à la ligne 90 du fichier draw.h.

6.3.2.5 MIN_COLORS_NUMBER

```
#define MIN_COLORS_NUMBER 256
```

Nombre de couleurs gérés par le terminal (nCurses), nécessaire pour faire tourner le programme.

Définition à la ligne 45 du fichier draw.h.

6.3.2.6 ORG

```
#define ORG "\033[38;2;255;165;0m"
```

Définition à la ligne 89 du fichier draw.h.

6.3.2.7 RESET

```
#define RESET "\x1B[0m"
```

Définition à la ligne 91 du fichier draw.h.

6.3.2.8 RUBIK_COLS

```
#define RUBIK_COLS 4
```

Permet de déterminer le nombre de colonnes dans la grande matrice nCurses. Il y a quatre colonnes.

Définition à la ligne 25 du fichier draw.h.

6.3.2.9 RUBIK_LINES

```
#define RUBIK_LINES 3
```

Permet de déterminer le nombre de lignes dans la grande matrice nCurses. Il y a trois lignes.

Définition à la ligne 19 du fichier draw.h.

6.3.2.10 SQ_HEIGHT

```
#define SQ_HEIGHT 4
```

Hauteur en nombre de caractères d'un cubie avec nCurses.

Définition à la ligne 30 du fichier draw.h.

6.3.2.11 SQ_WIDTH

```
#define SQ_WIDTH 8
```

Largeur en nombre de caractères d'un cubie avec nCurses.

Définition à la ligne 34 du fichier draw.h.

6.3.2.12 SQUARES

```
#define SQUARES ((RUBIK_LINES * 3) * (RUBIK_COLS * 3))
```

Nombre de carrés nCurses dans un Rubik's cube. Ne servira probablement pas.

Définition à la ligne 39 du fichier draw.h.

6.3.2.13 TERM_HAS_NO_COLORS

```
#define TERM_HAS_NO_COLORS 2
```

Le terminal (nCurses) ne gère pas les couleurs.

Définition à la ligne 55 du fichier draw.h.

6.3.2.14 TERM_HAS_NOT_ENOUGH_COLORS

```
#define TERM_HAS_NOT_ENOUGH_COLORS 3
```

Le terminal (nCurses) ne gère pas assez de couleurs.

Définition à la ligne 59 du fichier draw.h.

6.3.2.15 TERM_NOT_BIG_ENOUGH

```
#define TERM_NOT_BIG_ENOUGH 1
```

Le terminal (nCurses) n'est pas assez grand pour afficher le Rubik's cube avec nCurses.

Définition à la ligne 51 du fichier draw.h.

6.3.2.16 WHT

```
#define WHT "\033[38;2;255;255;255m"
```

Définition à la ligne 88 du fichier draw.h.

6.3.2.17 YEL

```
#define YEL "\033[38;2;255;255;0m"
```

Définition à la ligne 85 du fichier draw.h.

6.3.3 Documentation des fonctions

6.3.3.1 change_color()

```
void change_color (
            short line,
            short col,
            short cube_x,
            short cube_y,
            short color )
```

Fonction de test pour nCurses, qui permet de déplacer une couleur à partir de :

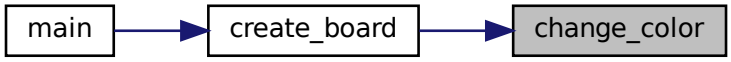
- line, col coordonnées du cube à modifier dans la matrice de l'écran
- cube_x, cube_y coordonnées x, y du cube à modifier
- color : Nouvelle couleur à appliquer

Paramètres

line	Ligne de la matrices nCurses
col	Colonne de la matrice nCurses
cube_x	Coordonnée X du cube à modifier
cube_y	Coordonnée Y du cube à modifier
color	Nouvelle couleur à appliquer

Définition à la ligne 115 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.3.3.2 check_and_set_term()

```
int check_and_set_term (
            void  )
```

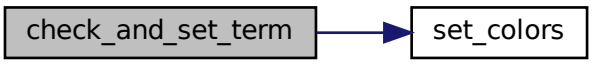
Vérification du terminal, pour voir si il pourra faire tourner le programme avec nCurses

Renvoie

0 si OK, sinon retourne une erreur différente de 0

Définition à la ligne 74 du fichier draw.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

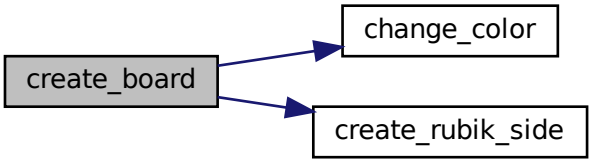
6.3.3.3 create_board()

```
void create_board (
             void  )
```

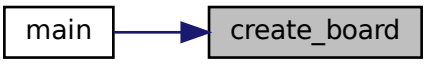
Fonction de test qui permet d'afficher diverses choses sur l'écran nCurses

Définition à la ligne 186 du fichier draw.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.3.3.4 create_rubik_side()

```
void create_rubik_side (
             short line,
             short col,
             short color,
             char name )
```

Fonction de test nCurses pour créer la face uniforme d'un cube dans une couleur donnée

Paramètres

line	Ligne de la matrice nCurses dans laquelle dessiner le cube
col	Colonne de la matrice nCurses dans laquelle dessiner le cube
color	Couleur à appliquer sur le cube
name	Nom de la face, à afficher dans le cubie central

Définition à la ligne 144 du fichier draw.c.

Voici le graphe des appelants de cette fonction :

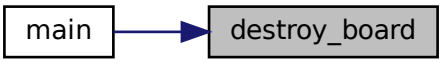
6.3.3.5 destroy_board()

```
void destroy_board (
            void  )
```

Fonction de test liée à create_board, qui permet de libérer les ressources et la mémoire.
Attention, cette fonction n'est pas à jour et ne libère pas correctement les choses

Définition à la ligne 234 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.3.3.6 detect_resize()

```
void detect_resize (
            __attribute__((unused)) int dummy )
```

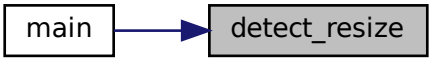
Détection de l'événement de changement de taille d'une fenêtre. L'évenement est non géré, pour le moment.

Paramètres

dummy	Paramètre inutilisé. Il est passé par la fonction d'appel des évènements sur une fenêtre

Définition à la ligne 50 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.3.3.7 draw_rubiks()

```
void draw_rubiks (
            rubiks_side * rubiks )
```

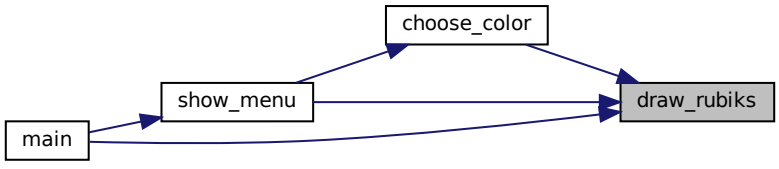
Dessine le rubik's Cube au format texte ANSI (pas au format nCurses)

Paramètres

rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 275 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.3.3.8 rubiks_display()

```
void rubiks_display (
            rubiks_side * rubiks )
```

Fonction de débuggage, permet d'afficher les positions après des mouvements.

Paramètres

rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 250 du fichier draw.c.

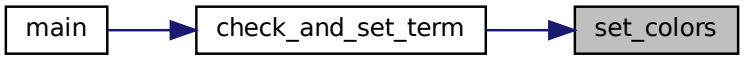
6.3.3.9 set_colors()

```
void set_colors (
             void  )
```

Définitions des couleurs pour nCurses

Définition à la ligne 58 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



6.3.4 Documentation des variables

6.3.4.1 BLACK_ON_BLUE

```
const short BLACK_ON_BLUE
```

Écriture noire sur bleu, pour nCurses

Définition à la ligne 33 du fichier draw.c.

6.3.4.2 BLACK_ON_GREEN

```
const short BLACK_ON_GREEN
```

Écriture noire sur vert, pour nCurses

Définition à la ligne 25 du fichier draw.c.

6.3.4.3 BLACK_ON_ORANGE

```
const short BLACK_ON_ORANGE
```

Écriture noire sur orange, pour nCurses

Définition à la ligne 21 du fichier draw.c.

6.3.4.4 BLACK_ON_RED

```
const short BLACK_ON_RED
```

Écriture noire sur rouge, pour nCurses

Définition à la ligne 29 du fichier draw.c.

6.3.4.5 BLACK_ON_WHITE

```
const short BLACK_ON_WHITE
```

Écriture noire sur blanc, pour nCurses

Définition à la ligne 17 du fichier draw.c.

6.3.4.6 BLACK_ON_YELLOW

```
const short BLACK_ON_YELLOW
```

Écriture noire sur jaune, pour nCurses

Définition à la ligne 37 du fichier draw.c.

6.3.4.7 BOARD

```
WINDOW* BOARD[SQUARES]
```

Définition à la ligne 12 du fichier draw.c.

6.3.4.8 NB_COLS

```
short NB_COLS
```

Définition à la ligne 43 du fichier draw.c.

6.3.4.9 NB_LINES

```
short NB_LINES
```

Taille de l'écran nCurses

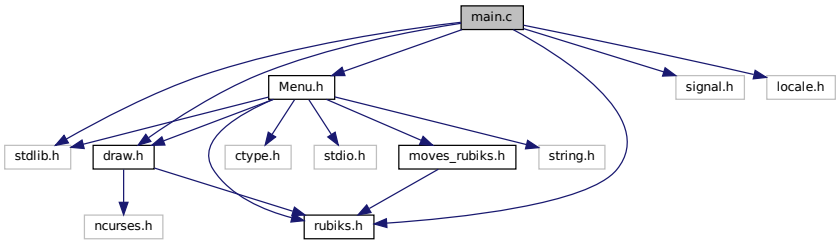
Définition à la ligne 42 du fichier draw.c.

6.4 Référence du fichier main.c

Programme principal qui gère le projet EFREI Rubik's Cube.

```
#include <stdlib.h>
#include <signal.h>
#include <locale.h>
#include "draw.h"
#include "rubiks.h"
#include "Menu.h"
```

Graphe des dépendances par inclusion de main.c:



Macros

— #define PRINT_TEXT_ONLY true

Fonctions

— int main ()

6.4.1 Description détaillée

Programme principal qui gère le projet EFREI Rubik's Cube.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.4.2 Documentation des macros

6.4.2.1 PRINT_TEXT_ONLY

```
#define PRINT_TEXT_ONLY true
```

Définie si nous réalisons les affichages en mode texte (true) ou en mode nCurses (false)

Définition à la ligne 19 du fichier main.c.

6.4.3 Documentation des fonctions

6.4.3.1 main()

```
int main ( )
```

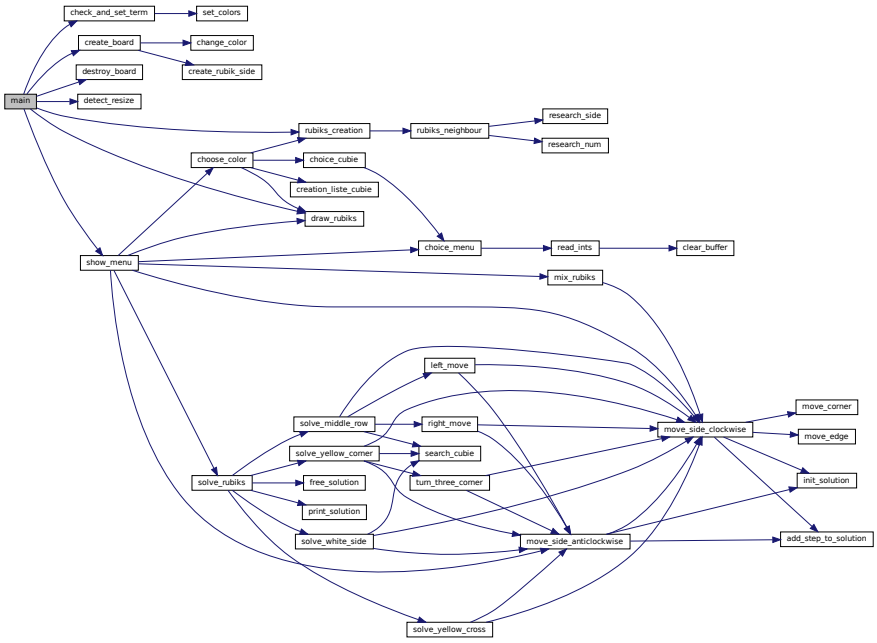
Fonction principale du programme. Elle se contente d'afficher les menus. Soit en mode nCurses (non implémenté pour le moment) soit en mode "texte" avec des couleurs ANSI.

Renvoie

0

Définition à la ligne 26 du fichier main.c.

Voici le graphe d'appel pour cette fonction :

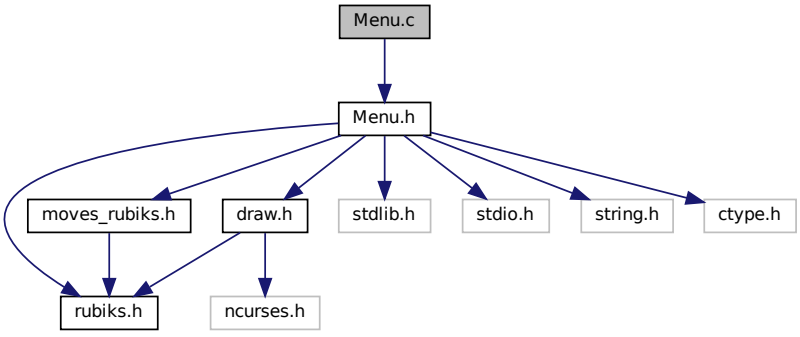


6.5 Référence du fichier Menu.c

Dans ce fichier, on gère tous les menus nécessaires aux différentes fonctions utilisables dans le projet. Ainsi que la gestion des saisies utilisateur.

```
#include "Menu.h"
```

Graphe des dépendances par inclusion de Menu.c:



Fonctions

- void show_menu (rubiks_side *rubiks)
- void choose_color (rubiks_side *rubiks)
- void choice_cubie (rubiks_side *reference_rubiks, rubiks_side *rubiks, T_CUBIE_TYPE type, cubies *liste_cubie, int len_liste, int face, int num_cubie)
- int creation_liste_cubie (rubiks_side *rubiks, cubies *liste, T_CUBIE_TYPE type)
- void clear_buffer (void)
- int read_ints (void)
- int choice_menu (int num_entries)

6.5.1 Description détaillée

Dans ce fichier, on gère tous les menus nécessaires aux différentes fonctions utilisables dans le projet. Ainsi que la gestion des saisies utilisateur.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.5.2 Documentation des fonctions

6.5.2.1 choice_cubie()

```
void choice_cubie (
            rubiks_side * reference_rubiks,
            rubiks_side * rubiks,
            T_CUBIE_TYPE type,
            cubies * liste_cubie,
            int len_liste,
            int face,
            int num_cubie )
```

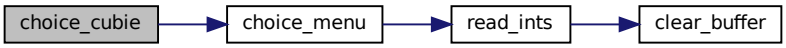
Cette fonction permet de récupérer le choix de l'utilisateur en fonction des cubies disponibles.

Paramètres

reference_rubiks	Pointeur sur un rubik's de référence
rubiks	Pointeur sur le rubik's de travail
type	Type de cubie à placer
liste_cubie	Liste des cubie disponibles
len_liste	Longueur de la liste précédente
face	Face du cubie à modifier
num_cubie	Numéro du cubie à modifier

Définition à la ligne 174 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

6.5.2.2 choice_menu()

```
int choice_menu (
            int num_entries )
```

Cette fonction valide les saisies de l'utilisateur en s'assurant que celles-ci sont bien celles qui sont acceptable. Les entrées disponibles vont de 1 à num_entries.

Paramètres

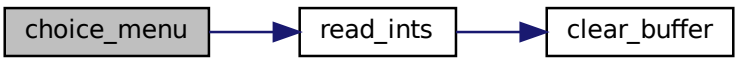
num_entries	Nombre d'entrées dans le menu.

Renvoie

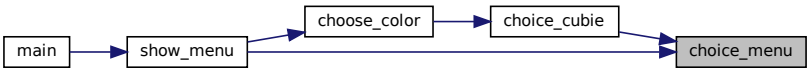
L'entrée choisit par l'utilisateur

Définition à la ligne 282 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.5.2.3 choose_color()

```
void choose_color (
            rubiks_side * rubiks )
```

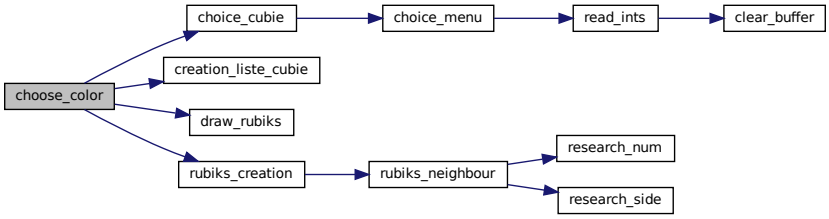
Cette fonction permet à l'utilisateur de remplir un cube

Paramètres

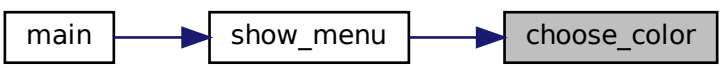
rubiks	Un pointeur vers le rubik's en cours de traitement.

Définition à la ligne 114 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



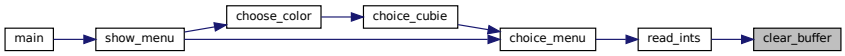
6.5.2.4 clear_buffer()

```
void clear_buffer (
            void  )
```

Purge des buffers de getchar(), car elle conserve les return du clavier. On les supprime donc quand ça nous arrange.

Définition à la ligne 241 du fichier Menu.c.

Voici le graphe des appelants de cette fonction :



6.5.2.5 creation_liste_cubie()

```
int creation_liste_cubie (
            rubiks_side * rubiks,
            cubies * liste,
            T_CUBIE_TYPE type )
```

Cette fonction va créer la liste des cubies disponibles à la saisie utilisateur

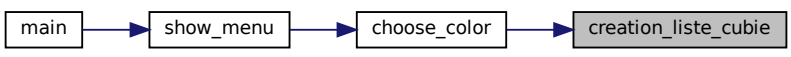
Paramètres

rubiks	Pointeur vers le rubik's de travail
liste	Liste des cubies disponibles en sortie
type	Type de cubie à travailler, soit EDGE, soit CORNER dans notre cas, puisque les centres sont placés d'avance.

Renvoie

Définition à la ligne 222 du fichier Menu.c.

Voici le graphe des appelants de cette fonction :



6.5.2.6 read_ints()

```
int read_ints (
            void  )
```

Cette fonction permet de lire plusieurs caractères, qui seront des entiers. Elle retourne l'entier saisi.

Renvoie

L'entier saisie par l'utilisateur

Définition à la ligne 255 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :

6.5.2.7 show_menu()

```
void show_menu (
            rubiks_side * rubiks )
```

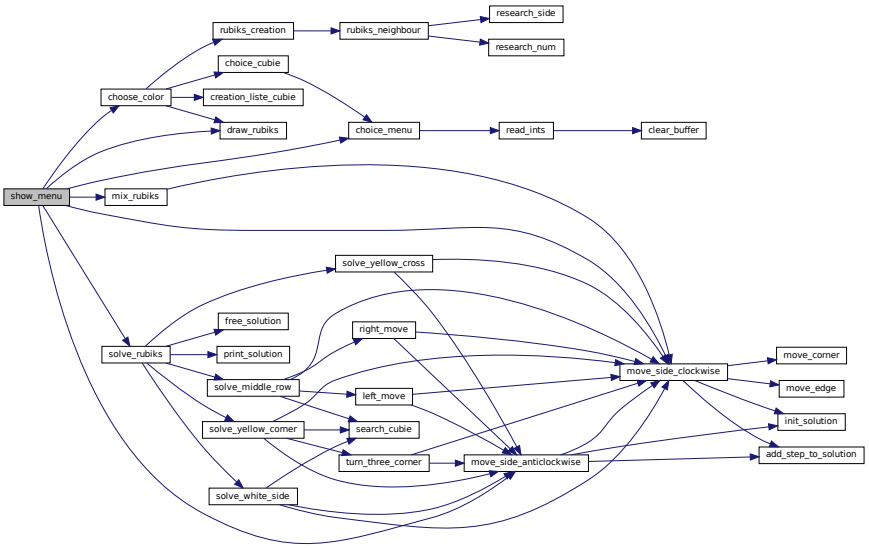
Cette fonction affiche le menu principal et tous les sous-menus.

Paramètres

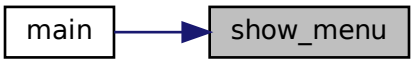
rubiks	Un pointeur vers le rubik's cube actuellement en cours d'utilisation

Définition à la ligne 15 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

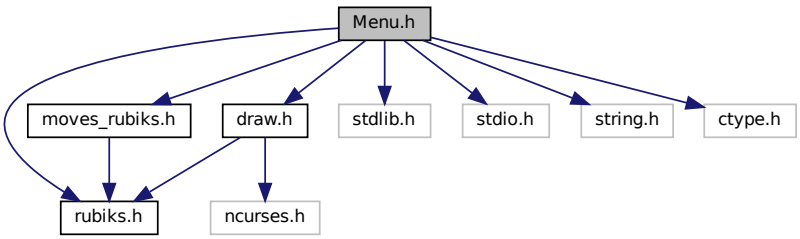


6.6 Référence du fichier Menu.h

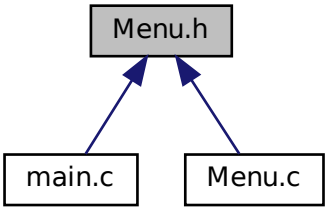
Dans ce fichier, on trouve les headers des fonction définies dans Menu.c.

```
#include "rubiks.h"
#include "moves_rubiks.h"
#include "draw.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
```

Graphe des dépendances par inclusion de Menu.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Fonctions

- void show_menu (rubiks_side *)
- void choose_color (rubiks_side *)
- int choice_menu (int)
- void clear_buffer (void)
- int read_ints (void)
- void choice_cubie (rubiks_side *, rubiks_side *, T_CUBIE_TYPE, cubies *, int, int, int)
- int creation_liste_cubie (rubiks_side *, cubies *, T_CUBIE_TYPE)

6.6.1 Description détaillée

Dans ce fichier, on trouve les headers des fonction définies dans Menu.c.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.6.2 Documentation des fonctions

6.6.2.1 choice_cubie()

```
void choice_cubie (
            rubiks_side * reference_rubiks,
            rubiks_side * rubiks,
            T_CUBIE_TYPE type,
            cubies * liste_cubie,
            int len_liste,
            int face,
            int num_cubie )
```

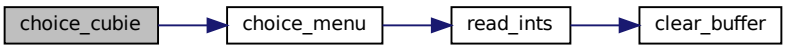
Cette fonction permet de récupérer le choix de l'utilisateur en fonction des cubies disponibles.

Paramètres

reference_rubiks	Pointeur sur un rubik's de référence
rubiks	Pointeur sur le rubik's de travail
type	Type de cubie à placer
liste_cubie	Liste des cubie disponibles
len_liste	Longueur de la liste précédente
face	Face du cubie à modifier
num_cubie	Numéro du cubie à modifier

Définition à la ligne 174 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

6.6.2.2 choice_menu()

```
int choice_menu (
            int num_entries )
```

Cette fonction valide les saisies de l'utilisateur en s'assurant que celles-ci sont bien celles qui sont acceptable. Les entrées disponibles vont de 1 à num_entries.

Paramètres

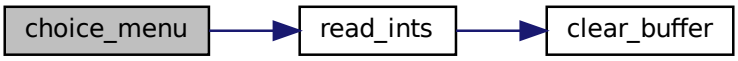
num_entries	Nombre d'entrées dans le menu.

Renvoie

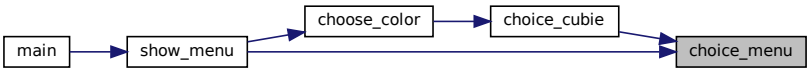
L'entrée choisit par l'utilisateur

Définition à la ligne 282 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.6.2.3 choose_color()

```
void choose_color (
            rubiks_side * rubiks )
```

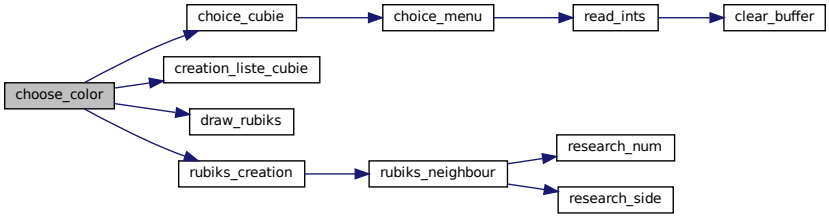
Cette fonction permet à l'utilisateur de remplir un cube

Paramètres

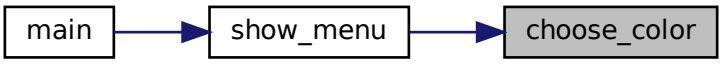
rubiks	Un pointeur vers le rubik's en cours de traitement.

Définition à la ligne 114 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



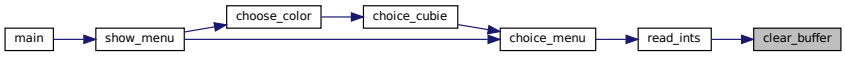
6.6.2.4 clear_buffer()

```
void clear_buffer (
            void  )
```

Purge des buffers de getchar(), car elle conserve les return du clavier. On les supprime donc quand ça nous arrange.

Définition à la ligne 241 du fichier Menu.c.

Voici le graphe des appelants de cette fonction :



6.6.2.5 creation_liste_cubie()

```
int creation_liste_cubie (
            rubiks_side * rubiks,
            cubies * liste,
            T_CUBIE_TYPE type )
```

Cette fonction va créer la liste des cubies disponibles à la saisie utilisateur

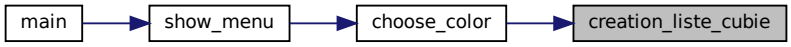
Paramètres

rubiks	Pointeur vers le rubik's de travail
liste	Liste des cubies disponibles en sortie
type	Type de cubie à travailler, soit EDGE, soit CORNER dans notre cas, puisque les centres sont placés d'avance.

Renvoie

Définition à la ligne 222 du fichier Menu.c.

Voici le graphe des appelants de cette fonction :



6.6.2.6 read_ints()

```
int read_ints (
            void  )
```

Cette fonction permet de lire plusieurs caractères, qui seront des entiers. Elle retourne l'entier saisi.

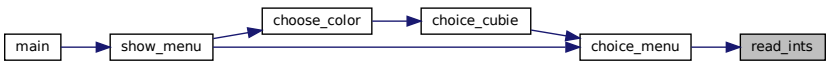
Renvoie

L'entier saisie par l'utilisateur

Définition à la ligne 255 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



6.6.2.7 show_menu()

```
void show_menu (
            rubiks_side * rubiks )
```

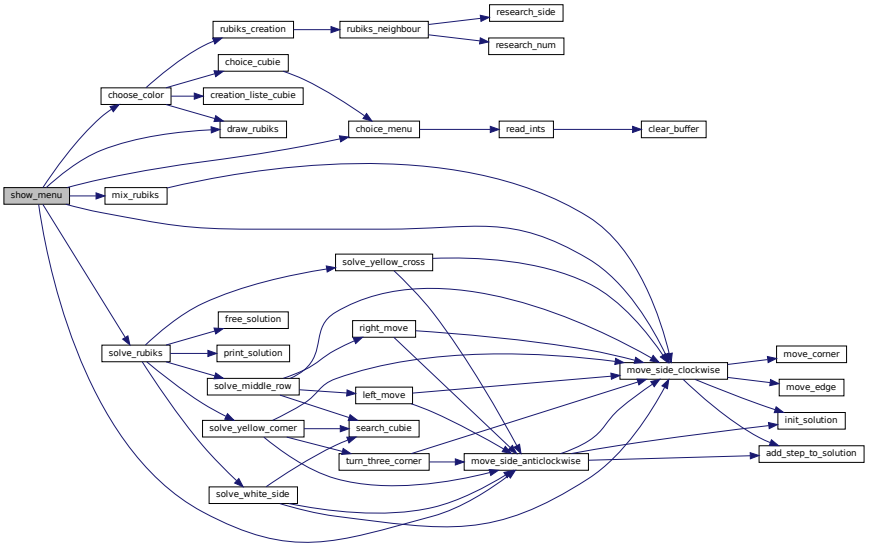
Cette fonction affiche le menu principal et tous les sous-menus.

Paramètres

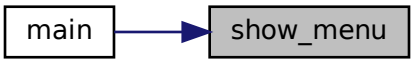
rubiks	Un pointeur vers le rubik's cube actuellement en cours d'utilisation

Définition à la ligne 15 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

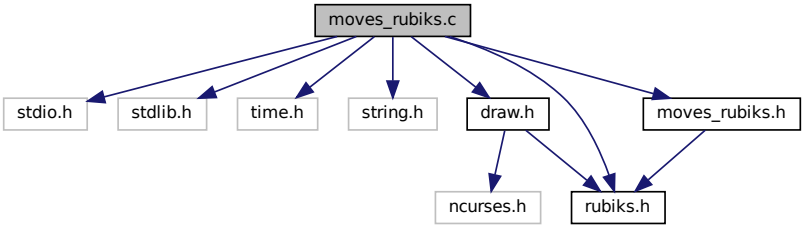


6.7 Référence du fichier moves_rubiks.c

Dans ce fichier, on gère les déplacements possible du Rubik's Cube, ainsi que le mélange automatique et la résolution d'un cube.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "draw.h"
#include "moves_rubiks.h"
#include "rubiks.h"
```

Graphe des dépendances par inclusion de moves_rubiks.c:



Fonctions

- void move_corner (rubiks_side *rubiks, int side, int from, int to)
- void move_edge (rubiks_side *rubiks, int side, int from, int to)
- void move_side_clockwise (rubiks_side *rubiks, int side, int add_to_history)
- void move_side_anticlockwise (rubiks_side *rubiks, int side, int add_to_history)
- void mix_rubiks (rubiks_side *rubiks)
- void alternate_color (rubiks_side *rubiks)
- void solve_rubiks (rubiks_side *rubiks)
- void solve_yellow_corner (rubiks_side *rubiks)
- void turn_three_corner (rubiks_side *rubiks)
- void solve_yellow_cross (rubiks_side *rubiks)
- void solve_middle_row (rubiks_side *rubiks)
- void right_move (rubiks_side *rubiks, cubies cubie)
- void left_move (rubiks_side *rubiks, cubies cubie)
- void solve_white_side (rubiks_side *rubiks)
- cubies search_cubie (rubiks_side *rubiks, T_COLOR cubie_color, T_COLOR neighbour1, T_CUBIE_TYPE cubie_type)
- solutions_steps * init_solution (char new_step[SIZE_MOVE])
- void add_step_to_solution (solutions_steps *first_step, char new_step[SIZE_MOVE])
- void print_solution (solutions_steps *first_step)
- void free_solution (solutions_steps *first_step)

Variables

- solutions_steps * history = NULL

6.7.1 Description détaillée

Dans ce fichier, on gère les déplacements possible du Rubik's Cube, ainsi que le mélange automatique et la résolution d'un cube.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.7.2 Documentation des fonctions

6.7.2.1 add_step_to_solution()

```
void add_step_to_solution (
            solutions_steps * first_step,
            char new_step[SIZE_MOVE] )
```

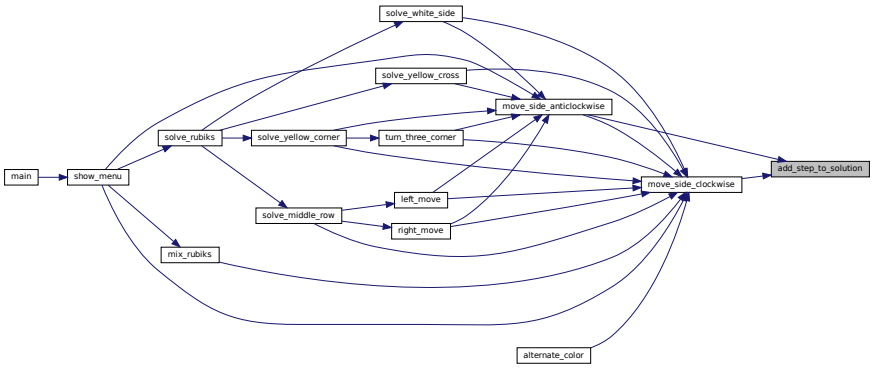
Cette fonction ajoute une nouvelle étape de résolution à un historique déjà existant.

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution
new_step	Texte à stocker pour cette nouvelle étape

Définition à la ligne 687 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.7.2.2 alternate_color()

```
void alternate_color (
            rubiks_side * rubiks )
```

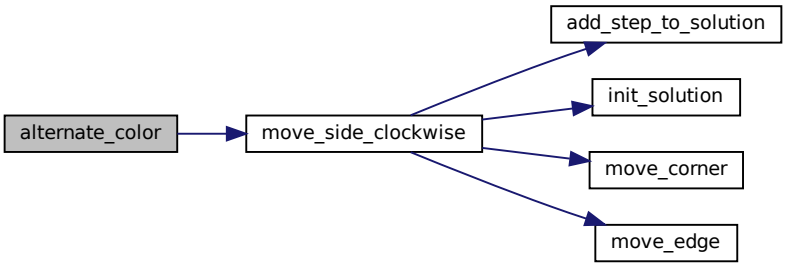
Cette fonction permet de créer un damier de couleurs

Paramètres

rubiks	un pointeur vers une structure rubiks_side

Définition à la ligne 150 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



6.7.2.3 free_solution()

```
void free_solution (
            solutions_steps * first_step )
```

Cette fonction libère totalement la mémoire allouée par la liste chainée des étapes d'une solution.

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution à supprimer.

Définition à la ligne 746 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :

6.7.2.4 init_solution()

```
solutions_steps* init_solution (
            char new_step[SIZE_MOVE] )
```

Cette fonction initialise une liste chaînée avec le premier mouvement de la solution. Elle renvoie un pointeur vers le premier élément (rubiks_solution.solution) de la liste chaînée (solution_steps).

Paramètres

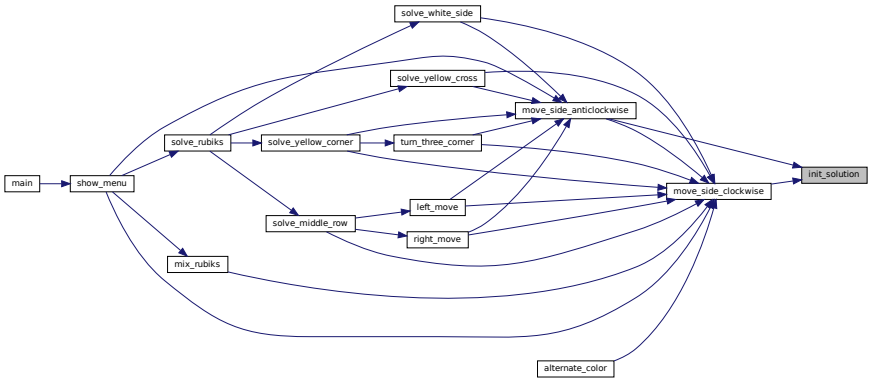
new_step	Description du premier mouvement, au format texte

Renvoie

Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution.

Définition à la ligne 662 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.7.2.5 left_move()

```
void left_move (
            rubiks_side * rubiks,
            cubies cubie )
```

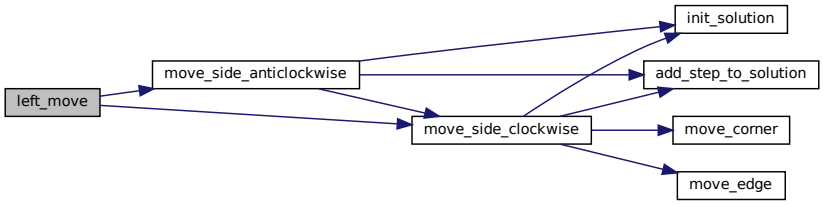
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne Déplacement vers le côté gauche. Utilisée lors de la résolution de la 2ème couronne.

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
cubie	cubie situé au dessus de l'arrête à déplacer

Définition à la ligne 451 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

6.7.2.6 mix_rubiks()

```
void mix_rubiks (
            rubiks_side * rubiks )
```

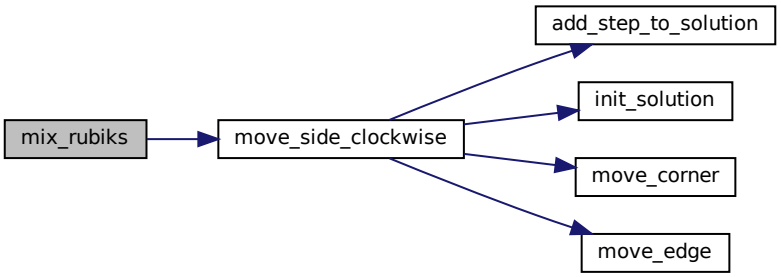
Cette fonction permet de mélanger le cube de manière aléatoire. Elle choisit entre 20 et 30 mouvements à réaliser tout en choisissant une face à déplacer, au hasard.

Paramètres

rubiks	un pointeur vers une structure rubiks_side

Définition à la ligne 133 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

6.7.2.7 move_corner()

```
void move_corner (
            rubiks_side * rubiks,
            int side,
            int from,
            int to )
```

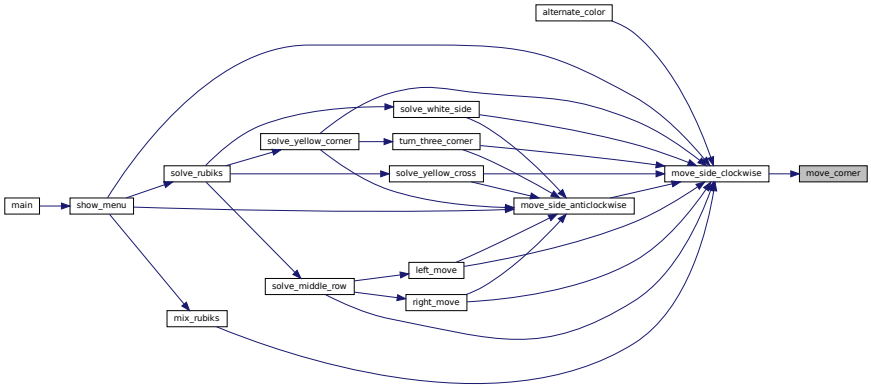
Cette fonction permet de déplacer un coin du cube sur la face "side", depuis "from" vers "to".

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
side	La face du rubiks à traiter
from	Déplacer depuis ce cubie
to	Déplacer vers ce cubie

Définition à la ligne 27 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.7.2.8 move_edge()

```
void move_edge (
            rubiks_side * rubiks,
            int side,
            int from,
            int to )
```

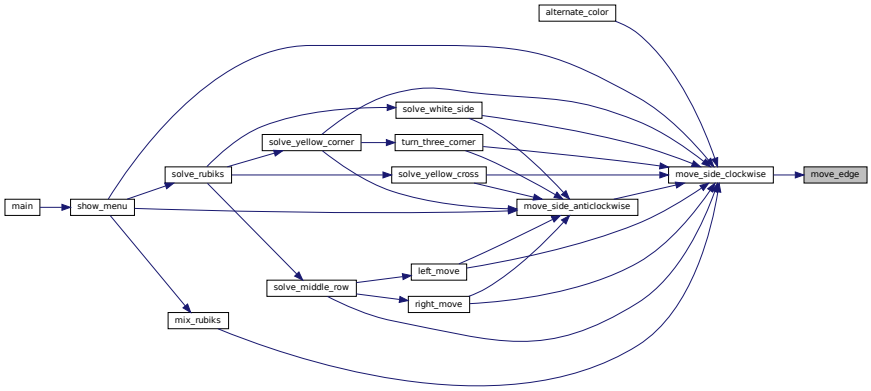
Cette fonction permet de déplacer une arête sur la face "side" depuis "from" vers "to".

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
side	La face du rubiks à traiter
from	Déplacer depuis ce cubie
to	Déplacer vers ce cubie

Définition à la ligne 45 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.7.2.9 move_side_anticlockwise()

```
void move_side_anticlockwise (
            rubiks_side * rubiks,
            int side,
            int add_to_history )
```

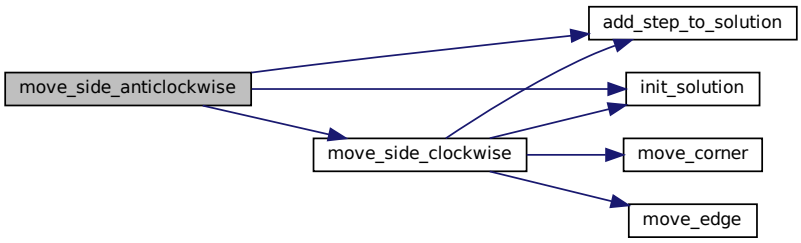
Cette fonction fait tourner la face side dans le sens anti-horaire. En réalité, cela revient à tourner trois fois dans le sens horaire.

Paramètres

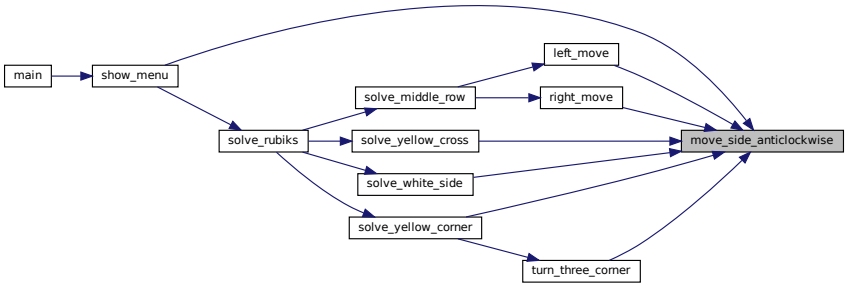
rubiks	Un pointeur vers une structure rubiks_side
side	La face à faire tourner dans le sens horaire
add_to_history	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

Définition à la ligne 108 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.10 move_side_clockwise()

```
void move_side_clockwise (
            rubiks_side * rubiks,
            int side,
            int add_to_history )
```

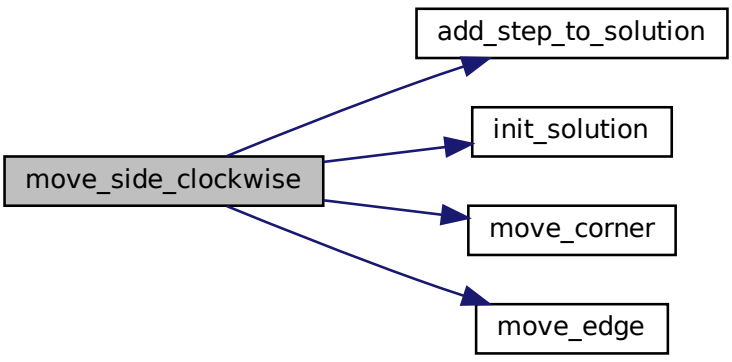
cette fonction permet de faire tourner la face side du cube dans le sens horaire

Paramètres

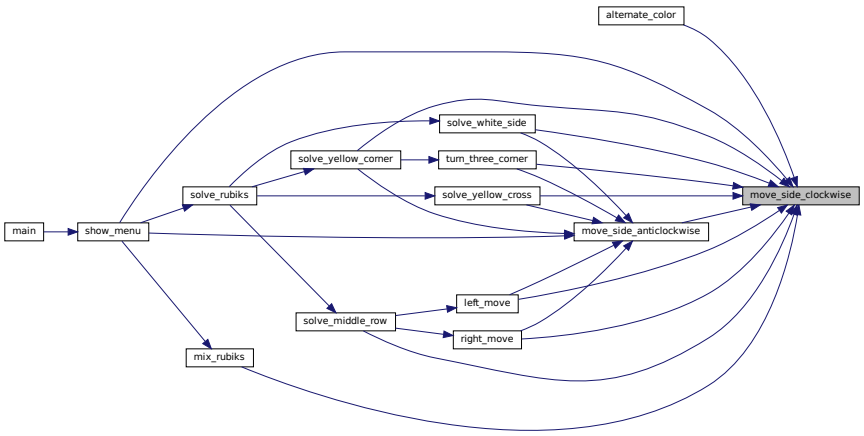
rubiks	Un pointeur vers une structure rubiks_side
side	La face à faire tourner dans le sens horaire
add_to_history	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

Définition à la ligne 58 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.11 print_solution()

```
void print_solution (
            solutions_steps * first_step )
```

Cette fonction imprime à l'écran, toutes les étapes d'une solution

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution à imprimer.

Définition à la ligne 718 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :

6.7.2.12 right_move()

```
void right_move (
            rubiks_side * rubiks,
            cubies cubie )
```

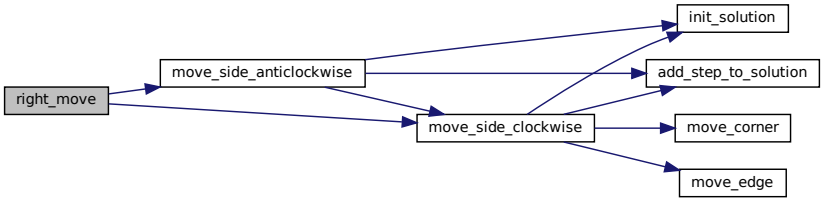
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne Déplacement vers le côté droit. Utilisée lors de la résolution de la 2ème couronne.

Paramètres

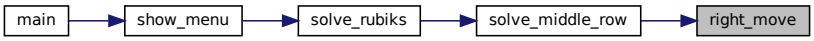
rubiks	Un pointeur vers une structure rubiks_side
cubie	cubie situé au dessus de l'arrête à déplacer

Définition à la ligne 431 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.13 search_cubie()

```
cubies search_cubie (
            rubiks_side * rubiks,
            T_COLOR cubie_color,
            T_COLOR neighbour1,
            T_CUBIE_TYPE cubie_type )
```

Cette fonction permet de trouver un cubie en fonction de sa couleur et celle de ses voisins.

Paramètres

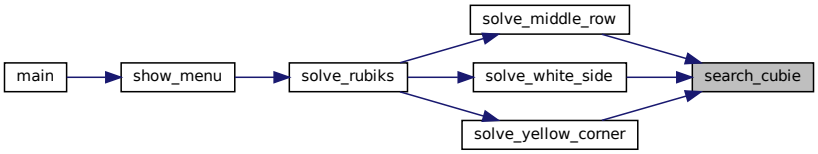
rubiks	Un pointeur vers une structure rubiks_side
cubie_color	couleur du cubie que l'on cherche
neighbour1	couleur du premier voisin du cubie recherché
cubie_type	type du cubie recherché

Renvoie

toutes les information sur un cubie une fois qu'il a été trouvé

Définition à la ligne 641 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.7.2.14 solve_middle_row()

```
void solve_middle_row (
            rubiks_side * rubiks )
```

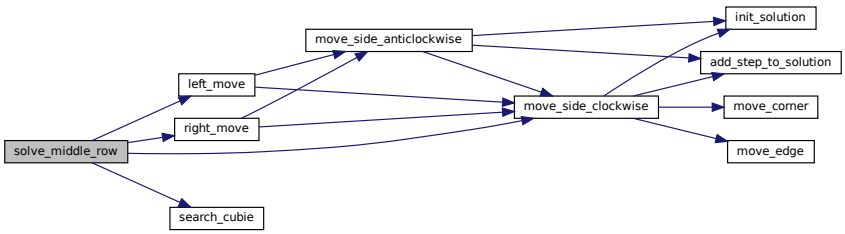
Cette fonction cherche à résoudre la 2ème couronne du rubik's cube Le choix est fait de ne pas "retourner" le rubiks cube et d'adapter les algorithmes en conséquence.

Paramètres

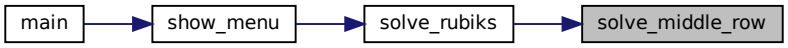
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 386 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.15 solve_rubiks()

```
void solve_rubiks (
            rubiks_side * rubiks )
```

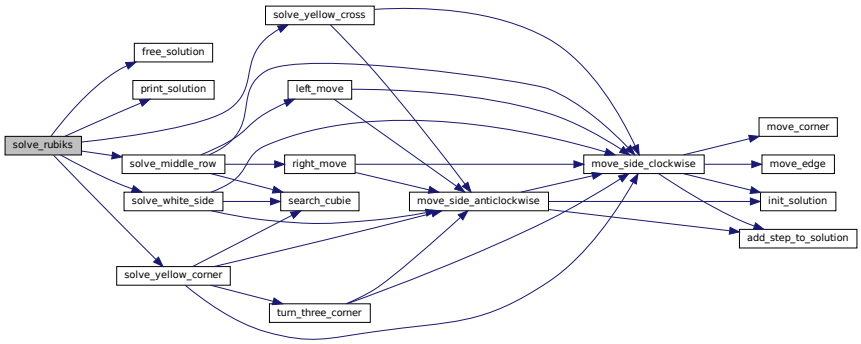
Cette fonction permet de résoudre le cube.

Paramètres

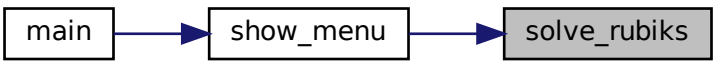
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 170 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.16 solve_white_side()

```
void solve_white_side (
            rubiks_side * rubiks )
```

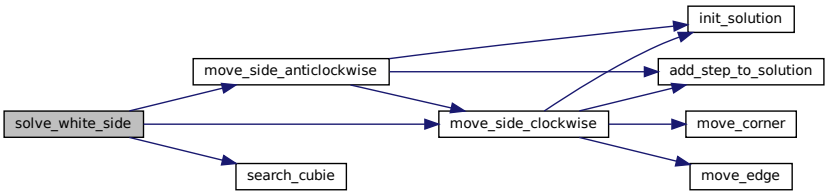
Cette fonction cherche à résoudre la face blanche du Rubik's Cube.

Paramètres

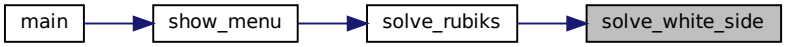
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 468 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.17 solve_yellow_corner()

```
void solve_yellow_corner (
            rubiks_side * rubiks )
```

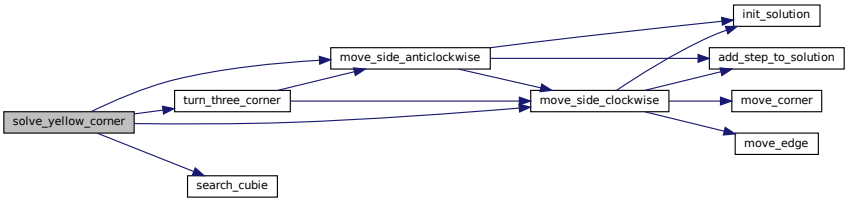
Cette fonction permet de résoudre les coins jaunes

Paramètres

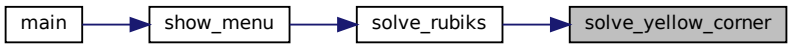
rubiks	Un pointeur vers une structure rubiks

Définition à la ligne 188 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.18 solve_yellow_cross()

```
void solve_yellow_cross (
            rubiks_side * rubiks )
```

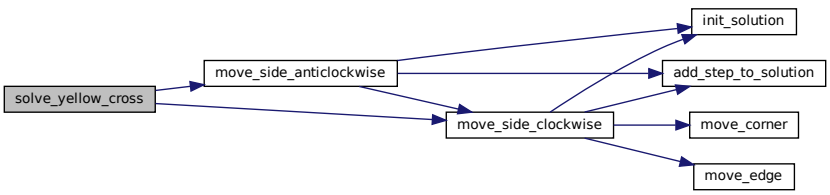
Cette fonction résoud la croix jaune.

Paramètres

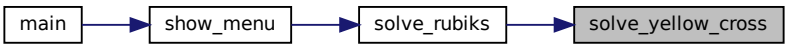
rubiks	Un pointeur vers une structure rubiks

Définition à la ligne 257 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.2.19 turn_three_corner()

```
void turn_three_corner (
            rubiks_side * rubiks )
```

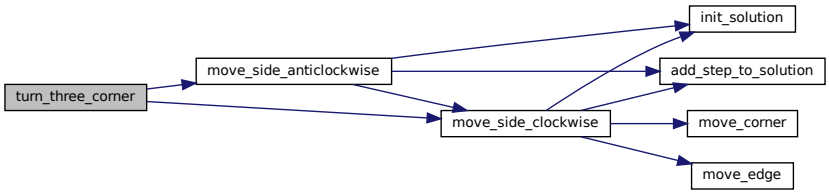
Cette fonction permet de faire tourner trois côtés d'une même face sans bouger le 4em et le reste du cube.

Paramètres

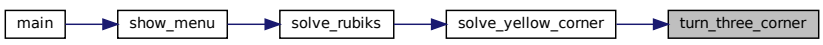
Rubiks	est un pointeur vers une strucure rubiks_sid

Définition à la ligne 241 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.7.3 Documentation des variables

6.7.3.1 history

```
solutions_steps* history = NULL
```

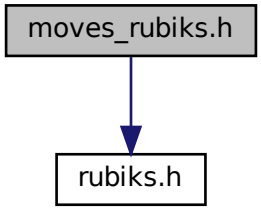
Définition à la ligne 17 du fichier moves_rubiks.c.

6.8 Référence du fichier moves_rubiks.h

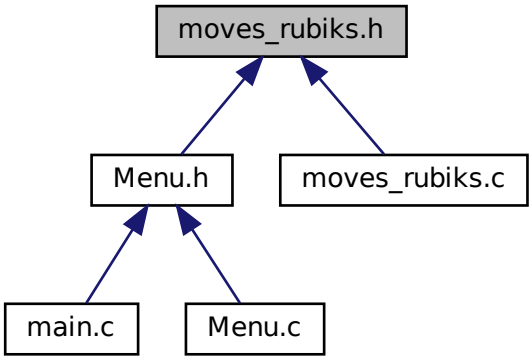
Dans ce fichier, on trouve les headers utilisés partout dans le code. Définitions dans moves_rubiks.c.

```
#include "rubiks.h"
```

Graphe des dépendances par inclusion de moves_rubiks.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

— struct solutions_steps

Macros

— #define SIZE_MOVE 20

Définitions de type

— typedef struct solutions_steps solutions_steps

Fonctions

— void move_side_clockwise (rubiks_side *, int, int)
— void move_side_anticlockwise (rubiks_side *, int, int)
— void move_corner (rubiks_side *, int, int, int)
— void move_edge (rubiks_side *, int, int, int)
— void mix_rubiks (rubiks_side *)
— void alternate_color (rubiks_side *)
— void solve_rubiks (rubiks_side *)
— void solve_white_side (rubiks_side *)
— void solve_middle_row (rubiks_side *)
— void solve_yellow_cross (rubiks_side *)
— void solve_yellow_corner (rubiks_side *)
— void turn_three_corner (rubiks_side *)
— cubies search_cubie (rubiks_side *, T_COLOR, T_COLOR, T_CUBIE_TYPE)
— void right_move (rubiks_side *, cubies)
— void left_move (rubiks_side *, cubies)
— solutions_steps * init_solution (char[SIZE_MOVE])
— void add_step_to_solution (solutions_steps *, char[SIZE_MOVE])
— void print_solution (solutions_steps *)
— void free_solution (solutions_steps *)

6.8.1 Description détaillée

Dans ce fichier, on trouve les headers utilisés partout dans le code. Définitions dans moves_rubiks.c.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.8.2 Documentation des macros

6.8.2.1 SIZE_MOVE

```
#define SIZE_MOVE 20
```

Définition à la ligne 12 du fichier moves_rubiks.h.

6.8.3 Documentation des définitions de type

6.8.3.1 solutions_steps

```
typedef struct solutions_steps solutions_steps
```

Définition à la ligne 13 du fichier moves_rubiks.h.

6.8.4 Documentation des fonctions

6.8.4.1 add_step_to_solution()

```
void add_step_to_solution (
            solutions_steps * first_step,
            char new_step[SIZE_MOVE] )
```

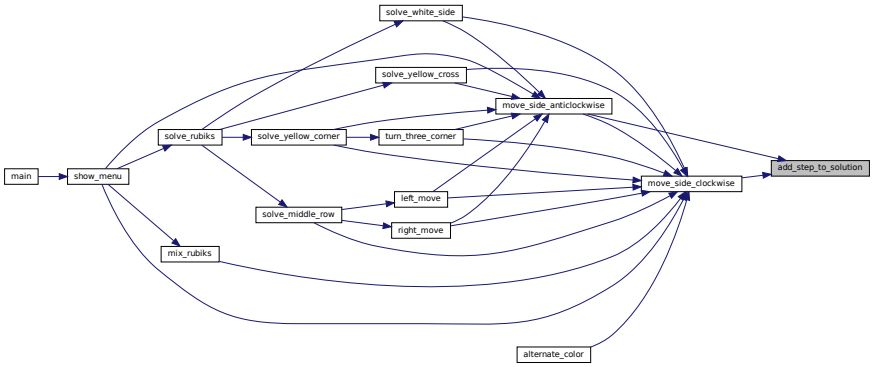
Cette fonction ajoute une nouvelle étape de résolution à un historique déjà existant.

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution
new_step	Texte à stocker pour cette nouvelle étape

Définition à la ligne 687 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.8.4.2 alternate_color()

```
void alternate_color (
            rubiks_side * rubiks )
```

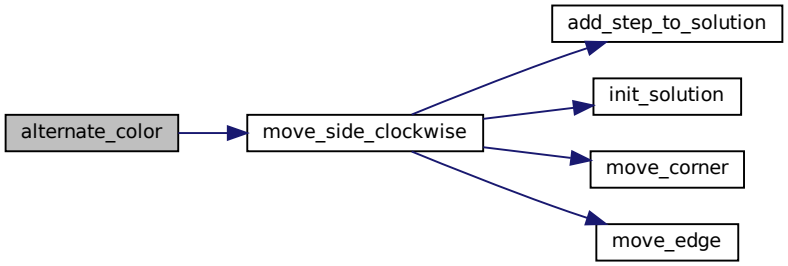
Cette fonction permet de créer un damier de couleurs

Paramètres

rubiks	un pointeur vers une structure rubiks_side

Définition à la ligne 150 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



6.8.4.3 free_solution()

```
void free_solution (
            solutions_steps * first_step )
```

Cette fonction libère totalement la mémoire allouée par la liste chainée des étapes d'une solution.

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution à supprimer.

Définition à la ligne 746 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :

6.8.4.4 init_solution()

```
solutions_steps* init_solution (
            char new_step[SIZE_MOVE] )
```

Cette fonction initialise une liste chaînée avec le premier mouvement de la solution. Elle renvoie un pointeur vers le premier élément (rubiks_solution.solution) de la liste chaînée (solution_steps).

Paramètres

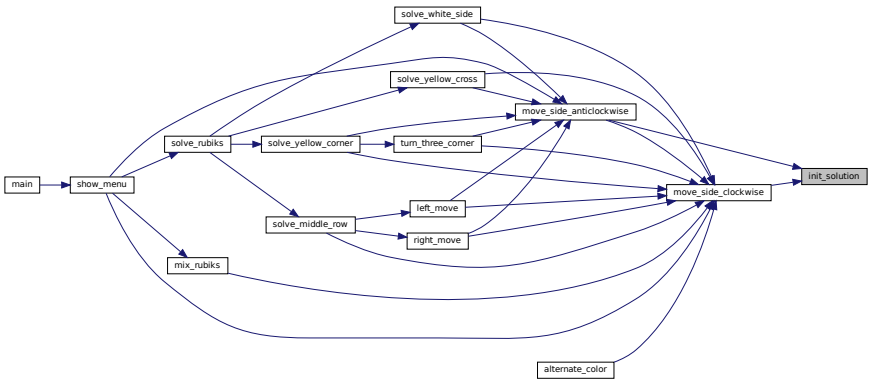
new_step	Description du premier mouvement, au format texte

Renvoie

Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution.

Définition à la ligne 662 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.8.4.5 left_move()

```
void left_move (
            rubiks_side * rubiks,
            cubies cubie )
```

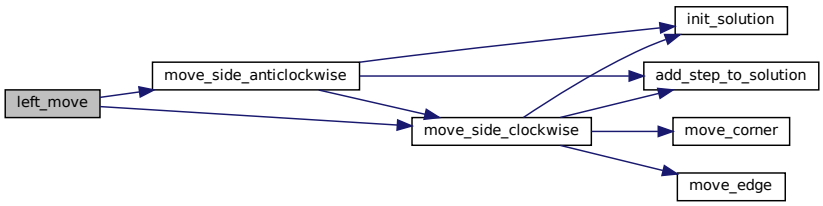
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne Déplacement vers le côté gauche. Utilisée lors de la résolution de la 2ème couronne.

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
cubie	cubie situé au dessus de l'arrête à déplacer

Définition à la ligne 451 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

6.8.4.6 mix_rubiks()

```
void mix_rubiks (
            rubiks_side * rubiks )
```

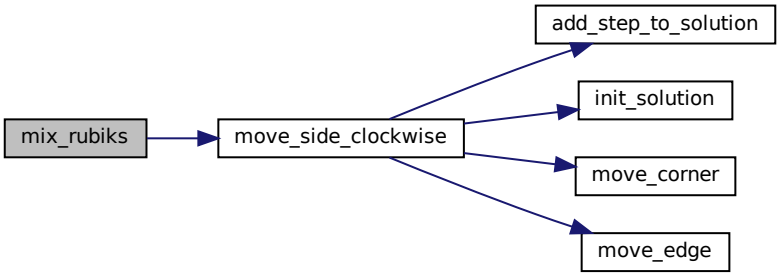
Cette fonction permet de mélanger le cube de manière aléatoire. Elle choisit entre 20 et 30 mouvements à réaliser tout en choisissant une face à déplacer, au hasard.

Paramètres

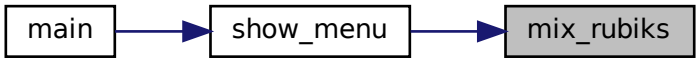
rubiks	un pointeur vers une structure rubiks_side

Définition à la ligne 133 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.7 move_corner()

```
void move_corner (
            rubiks_side * rubiks,
            int side,
            int from,
            int to )
```

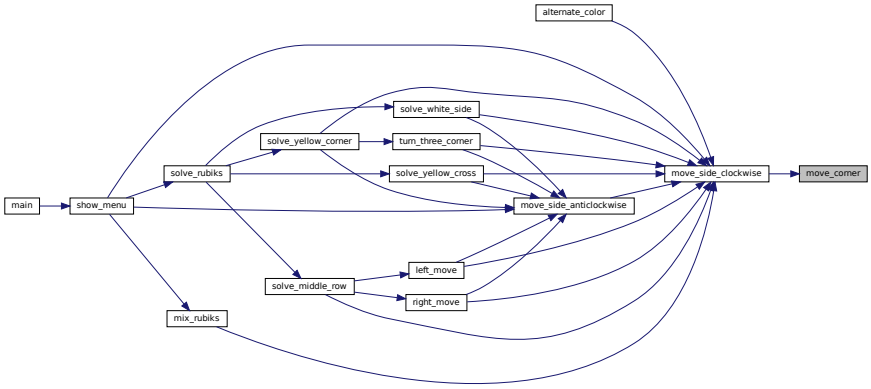
Cette fonction permet de déplacer un coin du cube sur la face "side", depuis "from" vers "to".

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
side	La face du rubiks à traiter
from	Déplacer depuis ce cubie
to	Déplacer vers ce cubie

Définition à la ligne 27 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.8.4.8 move_edge()

```
void move_edge (
            rubiks_side * rubiks,
            int side,
            int from,
            int to )
```

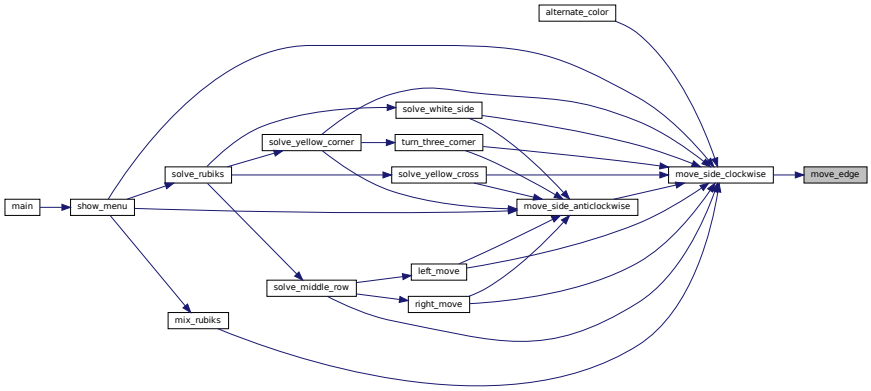
Cette fonction permet de déplacer une arête sur la face "side" depuis "from" vers "to".

Paramètres

rubiks	Un pointeur vers une structure rubiks_side
side	La face du rubiks à traiter
from	Déplacer depuis ce cubie
to	Déplacer vers ce cubie

Définition à la ligne 45 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.8.4.9 move_side_anticlockwise()

```
void move_side_anticlockwise (
            rubiks_side * rubiks,
            int side,
            int add_to_history )
```

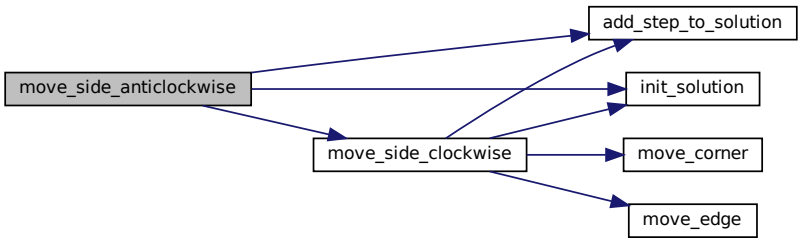
Cette fonction fait tourner la face side dans le sens anti-horaire. En réalité, cela revient à tourner trois fois dans le sens horaire.

Paramètres

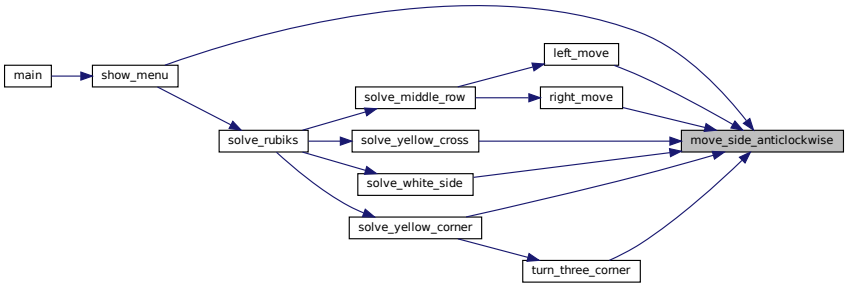
rubiks	Un pointeur vers une structure rubiks_side
side	La face à faire tourner dans le sens horaire
add_to_history	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

Définition à la ligne 108 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.10 move_side_clockwise()

```
void move_side_clockwise (
            rubiks_side * rubiks,
            int side,
            int add_to_history )
```

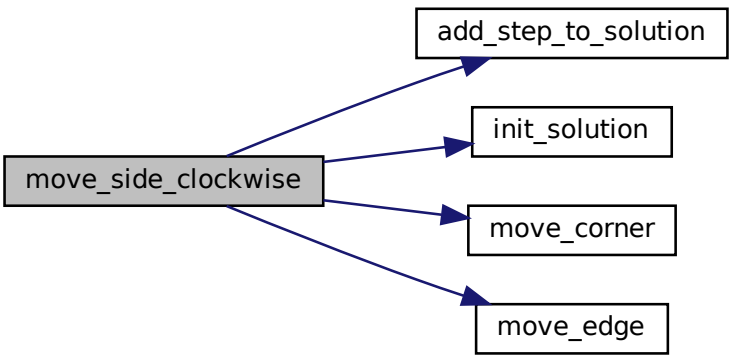
cette fonction permet de faire tourner la face side du cube dans le sens horaire

Paramètres

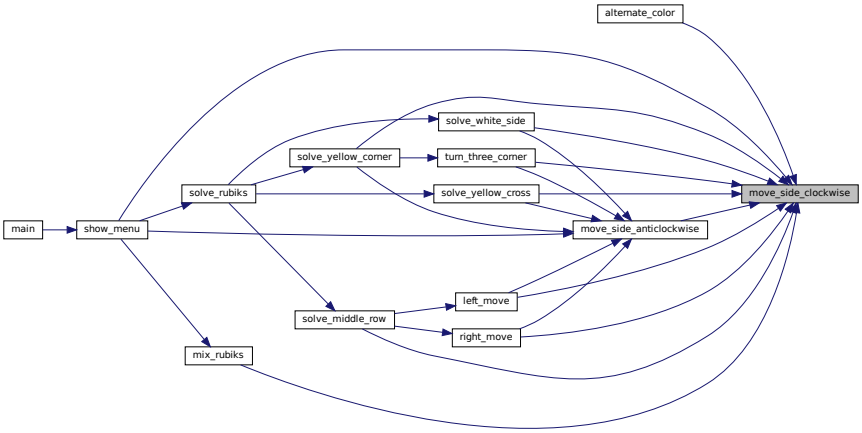
rubiks	Un pointeur vers une structure rubiks_side
side	La face à faire tourner dans le sens horaire
add_to_history	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

Définition à la ligne 58 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.11 print_solution()

```
void print_solution (
            solutions_steps * first_step )
```

Cette fonction imprime à l'écran, toutes les étapes d'une solution

Paramètres

first_step	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution à imprimer.

Définition à la ligne 718 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :

6.8.4.12 right_move()

```
void right_move (
            rubiks_side * rubiks,
            cubies cubie )
```

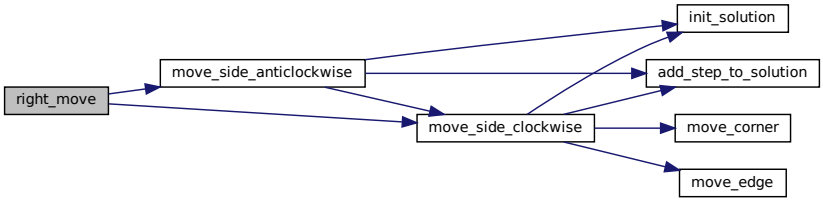
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne Déplacement vers le côté droit. Utilisée lors de la résolution de la 2ème couronne.

Paramètres

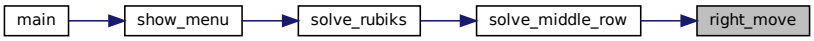
rubiks	Un pointeur vers une structure rubiks_side
cubie	cubie situé au dessus de l'arrête à déplacer

Définition à la ligne 431 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.13 search_cubie()

```
cubies search_cubie (
            rubiks_side * rubiks,
            T_COLOR cubie_color,
            T_COLOR neighbour1,
            T_CUBIE_TYPE cubie_type )
```

Cette fonction permet de trouver un cubie en fonction de sa couleur et celle de ses voisins.

Paramètres

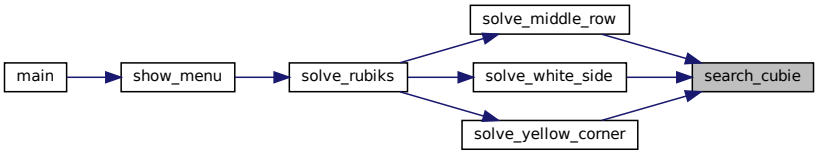
rubiks	Un pointeur vers une structure rubiks_side
cubie_color	couleur du cubie que l'on cherche
neighbour1	couleur du premier voisin du cubie recherché
cubie_type	type du cubie recherché

Renvoie

toutes les information sur un cubie une fois qu'il a été trouvé

Définition à la ligne 641 du fichier moves_rubiks.c.

Voici le graphe des appelants de cette fonction :



6.8.4.14 solve_middle_row()

```
void solve_middle_row (
            rubiks_side * rubiks )
```

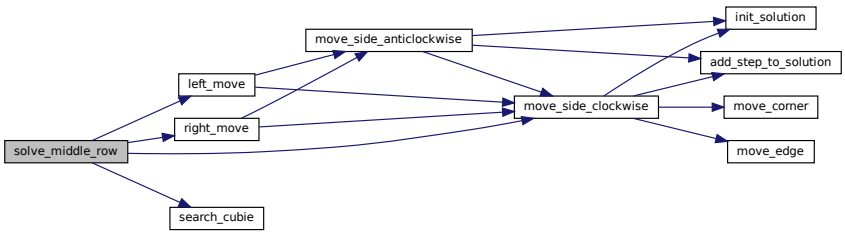
Cette fonction cherche à résoudre la 2ème couronne du rubik's cube Le choix est fait de ne pas "retourner" le rubiks cube et d'adapter les algorithmes en conséquence.

Paramètres

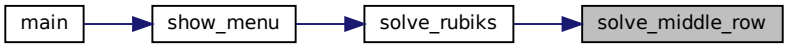
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 386 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.15 solve_rubiks()

```
void solve_rubiks (
            rubiks_side * rubiks )
```

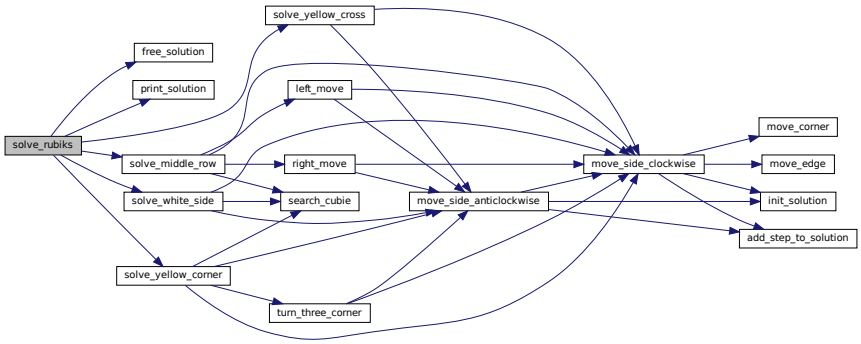
Cette fonction permet de résoudre le cube.

Paramètres

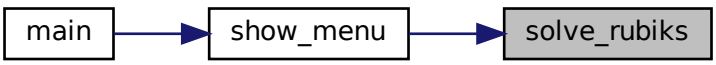
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 170 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.16 solve_white_side()

```
void solve_white_side (
            rubiks_side * rubiks )
```

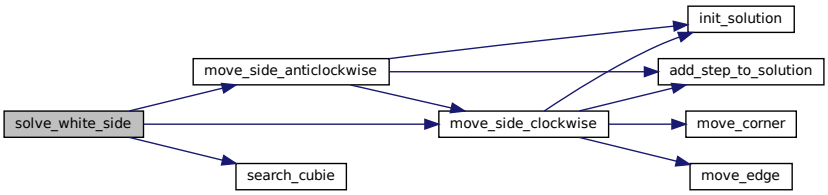
Cette fonction cherche à résoudre la face blanche du Rubik's Cube.

Paramètres

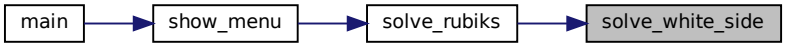
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 468 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.17 solve_yellow_corner()

```
void solve_yellow_corner (
            rubiks_side * rubiks )
```

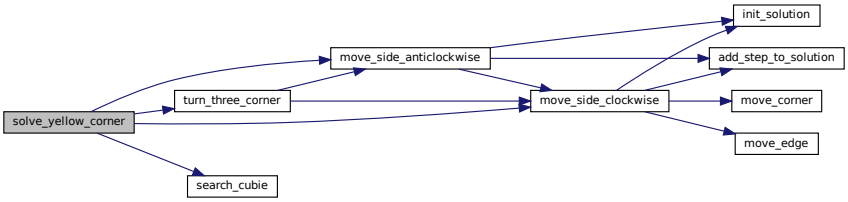
Cette fonction permet de résoudre les coins jaunes

Paramètres

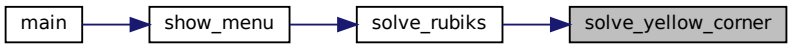
rubiks	Un pointeur vers une structure rubiks

Définition à la ligne 188 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.18 solve_yellow_cross()

```
void solve_yellow_cross (
            rubiks_side * rubiks )
```

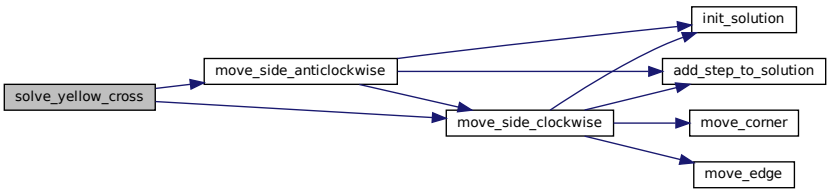
Cette fonction résoud la croix jaune.

Paramètres

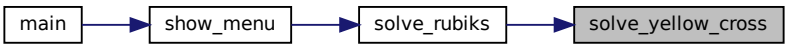
rubiks	Un pointeur vers une structure rubiks

Définition à la ligne 257 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.4.19 turn_three_corner()

```
void turn_three_corner (
            rubiks_side * rubiks )
```

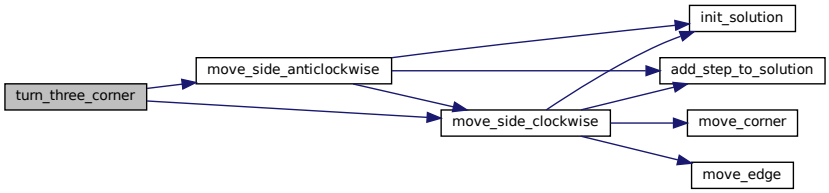
Cette fonction permet de faire tourner trois côtés d'une même face sans bouger le 4em et le reste du cube.

Paramètres

Rubiks	est un pointeur vers une strucure rubiks_sid

Définition à la ligne 241 du fichier moves_rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

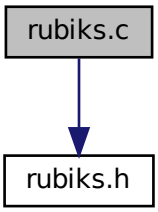
6.9 Référence du fichier README.md

6.10 Référence du fichier rubiks.c

Dans ce fichier, on gère toutes les propriétés du Rubik's Cube, depuis sa création, la gestion des faces adjacentes, et les recherches associées (recherche par couleur, ou depuis la couleur).

```
#include "rubiks.h"
```

Graphe des dépendances par inclusion de rubiks.c:



Fonctions

- void rubiks_creation (rubiks_side *rubiks)
- void rubiks_neighbour (rubiks_side *rubiks)
- int research_side (rubiks_side *rubiks, int side, int neighbour_face)
- int research_num (int side, int type, int side2)

6.10.1 Description détaillée

Dans ce fichier, on gère toutes les propriétés du Rubik's Cube, depuis sa création, la gestion des faces adjacentes, et les recherches associées (recherche par couleur, ou depuis la couleur).

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.10.2 Documentation des fonctions

6.10.2.1 research_num()

```
int research_num (
            int side,
            int type,
            int side2 )
```

Cette fonction retourne le numéro du cubie selon la position relative de la face de son ou de ses voisins.

Paramètres

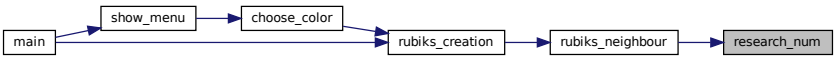
side	position relative de la face voisine au cubie dont on cherche le numéro.
type	Le type recherché permet d'orienter la recherche
side2	Dans le cas où l'on cherche la position du'un coin alors il est nécessaire de connaitre la position de la face de son deuxième voisin

Renvoie

Renvoi l'indice de lu cubie trouvé, -1 en cas d'erreur, ou 0 en cas d'échec de la recherche

Définition à la ligne 239 du fichier rubiks.c.

Voici le graphe des appelants de cette fonction :



6.10.2.2 research_side()

```
int research_side (
            rubiks_side * rubiks,
            int side,
            int neighbour_face )
```

Cette fonction permet de trouver à partir de la position absolu d'une face (neighbour_side) la position relative quelle occupe. exemple : si je souhaite retrouver la position de GREEN par rapport à ORANGE la fonction renvoie LEFT.

Paramètres

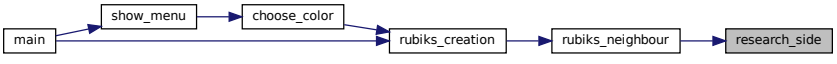
rubiks	Un pointeur vers une structure rubiks_side
side	La face principale qui nous sert de référence pour la recherche
neighbour_face	la face voisine, celle dont on cherche position

Renvoie

Renvoi l'indice de la face trouvée ou 0 en cas d'échec de la recherche

Définition à la ligne 219 du fichier rubiks.c.

Voici le graphe des appelants de cette fonction :



6.10.2.3 rubiks_creation()

```
void rubiks_creation (
            rubiks_side * rubiks )
```

Initialisation du rubik's cube. Au départ, il est résolu.

Dans cette fonction, et dans toutes celles qui en découlent, on considère que la face en cours de traitement est face à nous. Puis on se réfère au fichier ./Doc/reference_rubiks_excel.png pour déterminer les adjacents.
Si on prend la face blanche en exemple :

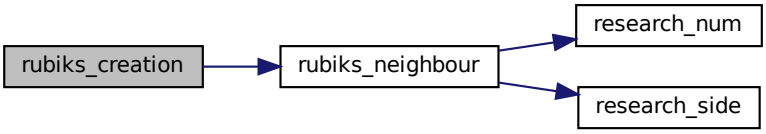
- La face au dessus est la face bleue
- La face à sa droite est la face rouge
- La face à sa gauche est la face orange
- La face sous-elle est la face verte
- La face opposée ne change jamais, dans le cas de la blanche, il s'agit de la face jaune.

Paramètres

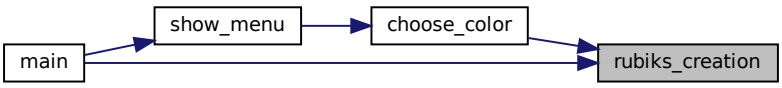
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 24 du fichier rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.10.2.4 rubiks_neighbour()

```
void rubiks_neighbour (
            rubiks_side * rubiks )
```

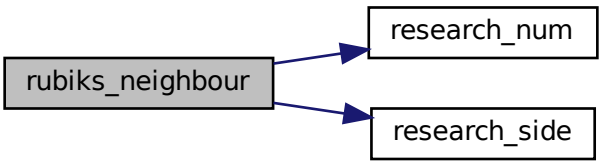
Cette fonction permet d'attribuer à chaque cubie ses voisin. num_side indique sur quelle face il se trouve num_cubie correspond au numéro du cubie voisin sur sa face les voisins ne changent pas avec la réalisation de mouvements ils sont donc attribués définitivement.

Paramètres

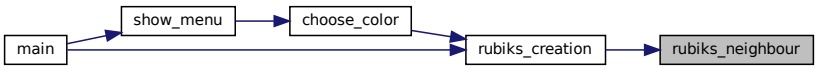
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 118 du fichier rubiks.c.

Voici le graphe d'appel pour cette fonction :



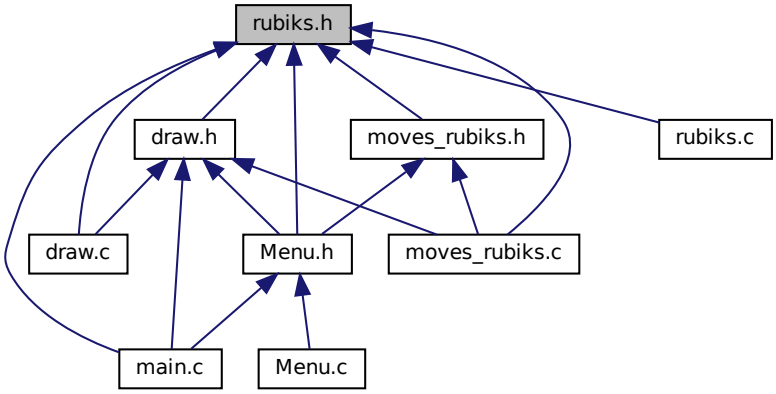
Voici le graphe des appelants de cette fonction :



6.11 Référence du fichier rubiks.h

Dans ce fichier, on trouve les headers et structures utilisées partout dans le code. Définitions dans rubiks.c.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

- struct neighbour
  Définition des adjacents à un cubie.
- struct cubies
  Définition des cubies, qui sont les petits cubes de couleur rattachés à une face.
- struct rubiks_side
  Définition d'une face du Rubik's Cube.

Énumérations

- enum T_COLOR {
  WHITE =0, ORANGE =1, GREEN =2, RED =3,
  BLUE =4, YELLOW =5, GREY = 6, NO_COLOR = 7 }
  Définition des couleurs des faces et des cubies.
- enum T_SIDE { UP =0, RIGHT =1, DOWN =2, LEFT =3 }
  Définition des faces adjacentes à une face du Rubik's Cube.
- enum T_CUBIE_TYPE { CORNER =0, EDGE =1, CENTER =2 }
  Définition des types de chaque cubie.

Fonctions

- void rubiks_creation (rubiks_side ∗rubiks)
- int research_side (rubiks_side ∗, int, int)
- int research_num (int, int, int)
- void rubiks_neighbour (rubiks_side ∗)

6.11.1 Description détaillée

Dans ce fichier, on trouve les headers et structures utilisées partout dans le code. Définitions dans rubiks.c.

Auteur

Mathieu CHANTOT et Clément LE STRAT

Date

7 Mai 2021

6.11.2 Documentation du type de l'énumération

6.11.2.1 T_COLOR

`enum T_COLOR`

Définition des couleurs des faces et des cubies.

Valeurs énumérées

WHITE	
ORANGE	
GREEN	
RED	
BLUE	
YELLOW	
GREY	
NO_COLOR	

Définition à la ligne 14 du fichier rubiks.h.

6.11.2.2 T_CUBIE_TYPE

`enum T_CUBIE_TYPE`

Définition des types de chaque cubie.

Valeurs énumérées

CORNER	
EDGE	
CENTER	

Définition à la ligne 24 du fichier rubiks.h.

6.11.2.3 T_SIDE

`enum T_SIDE`

Définition des faces adjacentes à une face du Rubik's Cube.

Valeurs énumérées

UP	
RIGHT	
DOWN	
LEFT	

Définition à la ligne 19 du fichier rubiks.h.

6.11.3 Documentation des fonctions

6.11.3.1 research_num()

```
int research_num (
            int side,
            int type,
            int side2 )
```

Cette fonction retourne le numéro du cubie selon la position relative de la face de son ou de ses voisins.

Paramètres

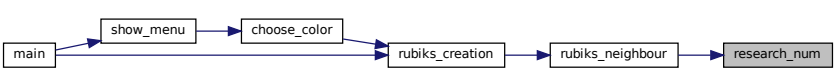
side	position relative de la face voisine au cubie dont on cherche le numéro.
type	Le type recherché permet d'orienter la recherche
side2	Dans le cas où l'on cherche la position du'un coin alors il est nécessaire de connaitre la position de la face de son deuxième voisin

Renvoie

Renvoi l'indice de lu cubie trouvé, -1 en cas d'erreur, ou 0 en cas d'échec de la recherche

Définition à la ligne 239 du fichier rubiks.c.

Voici le graphe des appelants de cette fonction :



6.11.3.2 research_side()

```
int research_side (
            rubiks_side * rubiks,
            int side,
            int neighbour_face )
```

Cette fonction permet de trouver à partir de la position absolu d'une face (neighbour_side) la position relative quelle occupe. exemple : si je souhaite retrouver la position de GREEN par rapport à ORANGE la fonction renvoie LEFT.

Paramètres

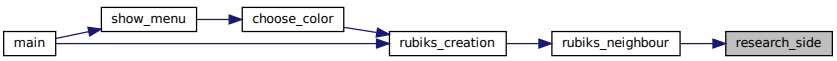
rubiks	Un pointeur vers une structure rubiks_side
side	La face principale qui nous sert de référence pour la recherche
neighbour_face	la face voisine, celle dont on cherche position

Renvoie

Renvoi l'indice de la face trouvée ou 0 en cas d'échec de la recherche

Définition à la ligne 219 du fichier rubiks.c.

Voici le graphe des appelants de cette fonction :



6.11.3.3 rubiks_creation()

```
void rubiks_creation (
            rubiks_side * rubiks )
```

Initialisation du rubik's cube. Au départ, il est résolu.

Dans cette fonction, et dans toutes celles qui en découlent, on considère que la face en cours de traitement est face à nous. Puis on se réfère au fichier ./Doc/reference_rubiks_excel.png pour déterminer les adjacents.
Si on prend la face blanche en exemple :

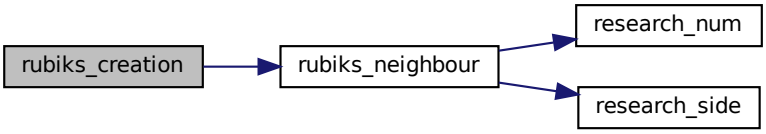
- La face au dessus est la face bleue
- La face à sa droite est la face rouge
- La face à sa gauche est la face orange
- La face sous-elle est la face verte
- La face opposée ne change jamais, dans le cas de la blanche, il s'agit de la face jaune.

Paramètres

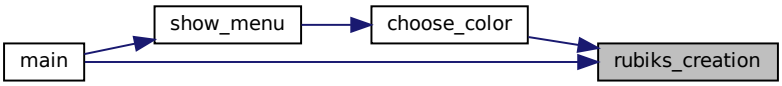
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 24 du fichier rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.11.3.4 rubiks_neighbour()

```
void rubiks_neighbour (
            rubiks_side * rubiks )
```

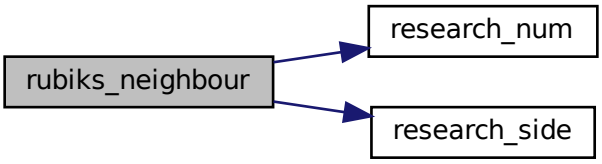
Cette fonction permet d'attribuer à chaque cubie ses voisin. num_side indique sur quelle face il se trouve num_cubie correspond au numéro du cubie voisin sur sa face les voisins ne changent pas avec la réalisation de mouvements ils sont donc attribués définitivement.

Paramètres

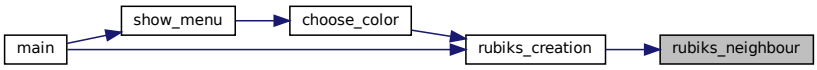
rubiks	Un pointeur vers une structure rubiks_side

Définition à la ligne 118 du fichier rubiks.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



Index