

Projet EFREI - C - Rubik's CUBE - 2021

Mathieu CHANTOT et Clément LE STRAT



<b>1 Doxygen</b>	<b>1</b>
1.1 Installation	1
1.1.1 Ubuntu	1
1.1.2 Manjaro	1
1.2 Utilisation et configuration	1
1.3 Génération de la documentation :	2
1.4 Références	2
<b>2 Références et liens utiles au projet rubiks_efrei</b>	<b>3</b>
2.1 Liens utiles et debuggage	3
2.2 Documentation et aides nCurses	3
2.3 CMake	3
2.4 Librairies	3
2.5 Documentation avec Doxygen	4
<b>3 Index des structures de données</b>	<b>5</b>
3.1 Structures de données	5
<b>4 Index des fichiers</b>	<b>7</b>
4.1 Liste des fichiers	7
<b>5 Documentation des structures de données</b>	<b>9</b>
5.1 Référence de la structure cubies	9
5.1.1 Description détaillée	9
5.1.2 Documentation des champs	10
5.1.2.1 color	10
5.1.2.2 cubie_side	10
5.1.2.3 neighbours	10
5.1.2.4 num	10
5.1.2.5 type	11
5.1.2.6 x	11
5.1.2.7 y	11
5.2 Référence de la structure neighbour	11
5.2.1 Description détaillée	11
5.2.2 Documentation des champs	12
5.2.2.1 num_cubie	12
5.2.2.2 num_side	12
5.3 Référence de la structure rubiks_side	12
5.3.1 Description détaillée	13
5.3.2 Documentation des champs	14
5.3.2.1 cubie	14
5.3.2.2 neighbour_side	14
5.3.2.3 opposite_side	14
5.3.2.4 side	14

5.4 Référence de la structure solutions_steps . . . . .	15
5.4.1 Description détaillée . . . . .	15
5.4.2 Documentation des champs . . . . .	15
5.4.2.1 next_step . . . . .	15
5.4.2.2 solution_step . . . . .	15
<b>6 Documentation des fichiers</b>	<b>17</b>
6.1 Référence du fichier Doxygen-installation-usage.md . . . . .	17
6.2 Référence du fichier draw.c . . . . .	17
6.2.1 Description détaillée . . . . .	18
6.2.2 Documentation des fonctions . . . . .	18
6.2.2.1 change_color() . . . . .	18
6.2.2.2 check_and_set_term() . . . . .	19
6.2.2.3 create_board() . . . . .	20
6.2.2.4 create_rubik_side() . . . . .	20
6.2.2.5 destroy_board() . . . . .	21
6.2.2.6 detect_resize() . . . . .	21
6.2.2.7 draw_rubiks() . . . . .	22
6.2.2.8 rubiks_display() . . . . .	22
6.2.2.9 set_colors() . . . . .	23
6.2.3 Documentation des variables . . . . .	23
6.2.3.1 BLACK_ON_BLUE . . . . .	23
6.2.3.2 BLACK_ON_GREEN . . . . .	23
6.2.3.3 BLACK_ON_ORANGE . . . . .	24
6.2.3.4 BLACK_ON_RED . . . . .	24
6.2.3.5 BLACK_ON_WHITE . . . . .	24
6.2.3.6 BLACK_ON_YELLOW . . . . .	24
6.2.3.7 BOARD . . . . .	24
6.2.3.8 NB_COLS . . . . .	25
6.2.3.9 NB_LINES . . . . .	25
6.3 Référence du fichier draw.h . . . . .	25
6.3.1 Description détaillée . . . . .	26
6.3.2 Documentation des macros . . . . .	27
6.3.2.1 ARED . . . . .	27
6.3.2.2 BLU . . . . .	27
6.3.2.3 GRN . . . . .	27
6.3.2.4 GRY . . . . .	27
6.3.2.5 MIN_COLORS_NUMBER . . . . .	27
6.3.2.6 ORG . . . . .	28
6.3.2.7 RESET . . . . .	28
6.3.2.8 RUBIK_COLS . . . . .	28
6.3.2.9 RUBIK_LINES . . . . .	28

6.3.2.10 SQ_HEIGHT . . . . .	28
6.3.2.11 SQ_WIDTH . . . . .	29
6.3.2.12 SQUARES . . . . .	29
6.3.2.13 TERM_HAS_NO_COLORS . . . . .	29
6.3.2.14 TERM_HAS_NOT_ENOUGH_COLORS . . . . .	29
6.3.2.15 TERM_NOT_BIG_ENOUGH . . . . .	29
6.3.2.16 WHT . . . . .	30
6.3.2.17 YEL . . . . .	30
6.3.3 Documentation des fonctions . . . . .	30
6.3.3.1 change_color() . . . . .	30
6.3.3.2 check_and_set_term() . . . . .	31
6.3.3.3 create_board() . . . . .	32
6.3.3.4 create_rubik_side() . . . . .	32
6.3.3.5 destroy_board() . . . . .	33
6.3.3.6 detect_resize() . . . . .	33
6.3.3.7 draw_rubiks() . . . . .	34
6.3.3.8 rubiks_display() . . . . .	34
6.3.3.9 set_colors() . . . . .	35
6.3.4 Documentation des variables . . . . .	35
6.3.4.1 BLACK_ON_BLUE . . . . .	35
6.3.4.2 BLACK_ON_GREEN . . . . .	35
6.3.4.3 BLACK_ON_ORANGE . . . . .	36
6.3.4.4 BLACK_ON_RED . . . . .	36
6.3.4.5 BLACK_ON_WHITE . . . . .	36
6.3.4.6 BLACK_ON_YELLOW . . . . .	36
6.3.4.7 BOARD . . . . .	36
6.3.4.8 NB_COLS . . . . .	37
6.3.4.9 NB_LINES . . . . .	37
6.4 Référence du fichier main.c . . . . .	37
6.4.1 Description détaillée . . . . .	38
6.4.2 Documentation des macros . . . . .	38
6.4.2.1 PRINT_TEXT_ONLY . . . . .	38
6.4.3 Documentation des fonctions . . . . .	38
6.4.3.1 main() . . . . .	38
6.5 Référence du fichier Menu.c . . . . .	39
6.5.1 Description détaillée . . . . .	40
6.5.2 Documentation des fonctions . . . . .	40
6.5.2.1 choice_cubie() . . . . .	40
6.5.2.2 choice_menu() . . . . .	41
6.5.2.3 choose_color() . . . . .	42
6.5.2.4 clear_buffer() . . . . .	43
6.5.2.5 creation_liste_cubie() . . . . .	43

6.5.2.6 read_ints()	44
6.5.2.7 show_menu()	44
6.6 Référence du fichier Menu.h	45
6.6.1 Description détaillée	46
6.6.2 Documentation des fonctions	47
6.6.2.1 choice_cubie()	47
6.6.2.2 choice_menu()	48
6.6.2.3 choose_color()	48
6.6.2.4 clear_buffer()	49
6.6.2.5 creation_liste_cubie()	49
6.6.2.6 read_ints()	50
6.6.2.7 show_menu()	51
6.7 Référence du fichier moves_rubiks.c	51
6.7.1 Description détaillée	52
6.7.2 Documentation des fonctions	53
6.7.2.1 add_step_to_solution()	53
6.7.2.2 alternate_color()	53
6.7.2.3 free_solution()	54
6.7.2.4 init_solution()	54
6.7.2.5 left_move()	55
6.7.2.6 mix_rubiks()	56
6.7.2.7 move_corner()	57
6.7.2.8 move_edge()	58
6.7.2.9 move_side_anticlockwise()	59
6.7.2.10 move_side_clockwise()	60
6.7.2.11 print_solution()	61
6.7.2.12 right_move()	61
6.7.2.13 search_cubie()	62
6.7.2.14 solve_middle_row()	63
6.7.2.15 solve_rubiks()	64
6.7.2.16 solve_white_side()	65
6.7.2.17 solve_yellow_corner()	65
6.7.2.18 solve_yellow_cross()	66
6.7.2.19 turn_three_corner()	67
6.7.3 Documentation des variables	67
6.7.3.1 history	68
6.8 Référence du fichier moves_rubiks.h	68
6.8.1 Description détaillée	69
6.8.2 Documentation des macros	69
6.8.2.1 SIZE_MOVE	69
6.8.3 Documentation des définitions de type	70
6.8.3.1 solutions_steps	70

6.8.4 Documentation des fonctions	70
6.8.4.1 add_step_to_solution()	70
6.8.4.2 alternate_color()	71
6.8.4.3 free_solution()	71
6.8.4.4 init_solution()	72
6.8.4.5 left_move()	73
6.8.4.6 mix_rubiks()	73
6.8.4.7 move_corner()	74
6.8.4.8 move_edge()	75
6.8.4.9 move_side_anticlockwise()	76
6.8.4.10 move_side_clockwise()	77
6.8.4.11 print_solution()	78
6.8.4.12 right_move()	78
6.8.4.13 search_cubie()	79
6.8.4.14 solve_middle_row()	80
6.8.4.15 solve_rubiks()	81
6.8.4.16 solve_white_side()	82
6.8.4.17 solve_yellow_corner()	82
6.8.4.18 solve_yellow_cross()	83
6.8.4.19 turn_three_corner()	84
6.9 Référence du fichier README.md	84
6.10 Référence du fichier rubiks.c	84
6.10.1 Description détaillée	85
6.10.2 Documentation des fonctions	85
6.10.2.1 research_num()	85
6.10.2.2 research_side()	86
6.10.2.3 rubiks_creation()	87
6.10.2.4 rubiks_neighbour()	88
6.11 Référence du fichier rubiks.h	88
6.11.1 Description détaillée	89
6.11.2 Documentation du type de l'énumération	90
6.11.2.1 T_COLOR	90
6.11.2.2 T_CUBIE_TYPE	90
6.11.2.3 T_SIDE	90
6.11.3 Documentation des fonctions	91
6.11.3.1 research_num()	91
6.11.3.2 research_side()	92
6.11.3.3 rubiks_creation()	92
6.11.3.4 rubiks_neighbour()	93
<b>Index</b>	<b>95</b>





# 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/output/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](#)
- [Exemples 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 :

<a href="#">cubies</a>	Définition des cubies, qui sont les petits cubes de couleur rattachés à une face . . . . .	9
<a href="#">neighbour</a>	Définition des adjacents à un cubie . . . . .	11
<a href="#">rubiks_side</a>	Définition d'une face du Rubik's Cube . . . . .	12
<a href="#">solutions_steps</a>	. . . . .	15



# Chapitre 4

## Index des fichiers

### 4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

<a href="#">draw.c</a>	Dans ce fichier, on gère les différents affichages du Rubik's Cube . . . . .	17
<a href="#">draw.h</a>	Dans ce fichier, on trouve les headers utilisés dans <a href="#">draw.c</a> . . . . .	25
<a href="#">main.c</a>	Programme principal qui gère le projet EFREI Rubik's Cube . . . . .	37
<a href="#">Menu.c</a>	Dans ce fichier, on gère tous les menus nécessaires aux différentes fonctions utilisables dans le projet. Ainsi que la gestion des saisies utilisateur . . . . .	39
<a href="#">Menu.h</a>	Dans ce fichier, on trouve les headers des fonction définies dans <a href="#">Menu.c</a> . . . . .	45
<a href="#">moves_rubiks.c</a>	Dans ce fichier, on gère les déplacements possible du Rubik's Cube, ainsi que le mélange automatique et la résolution d'un cube . . . . .	51
<a href="#">moves_rubiks.h</a>	Dans ce fichier, on trouve les headers utilisés partout dans le code. Définitions dans <a href="#">moves_rubiks.c</a> . . . . .	68
<a href="#">rubiks.c</a>	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) . .	84
<a href="#">rubiks.h</a>	Dans ce fichier, on trouve les headers et structures utilisées partout dans le code. Définitions dans <a href="#">rubiks.c</a> . . . . .	88





## Chapitre 5

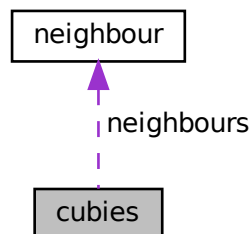
# Documentation des structures de données

### 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**

<i>x</i>	coordonnée horizontale du cubie
<i>y</i>	coordonnée verticale du cubie
<i>num</i>	Numéro du cubie dans la face : de 0 à 8
<i>type</i>	Type de cubie, CORNER, EDGE ou CENTER
<i>color</i>	Couleur du cubie
<i>cubie_side</i>	Face de rattachement du cubie
<i>neighbours[2]</i>	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

<code>num_side</code>	Face de rattachement du voisin
<code>num_cubie</code>	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 :

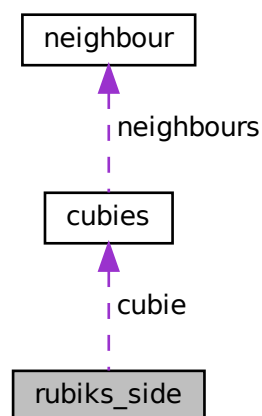
— [rubiks.h](#)

## 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**

<i>neighbour_side[4]</i>	Une face a 4 faces adjacentes, UP, DOWN, LEFT, et RIGHT.
<i>opposite_side</i>	Définition de la face opposée.
<i>side</i>	Définition du numéro et couleur de la face. Soit WHITE, ORANGE, GREEN, RED, BLUE, ou YELLOW
<i>cubie[9]</i>	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.

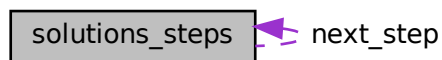
La documentation de cette structure a été générée à partir du fichier suivant :

— [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

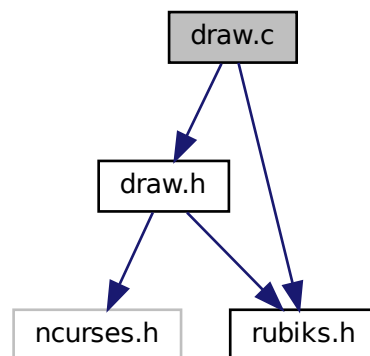
### 6.1 Référence du fichier Doxygen-installation-usage.md

### 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 :

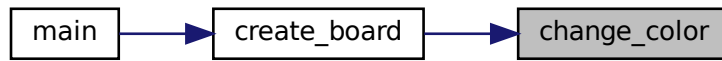
- line, col coordonnées du cube à modifier dans la matrice de l'écran
- cube\_x, cube\_y coordonnées x, y du cube à modifier
- color : Nouvelle couleur à appliquer

#### Paramètres

<i>line</i>	Ligne de la matrices nCurses
<i>col</i>	Colonne de la matrice nCurses
<i>cube_x</i>	Coordonnée X du cube à modifier
<i>cube_y</i>	Coordonnée Y du cube à modifier
<i>color</i>	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 )
```

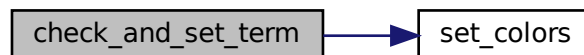
Vérification du terminal, pour voir si il pourra faire tourner le programme avec nCurses

##### Renvoie

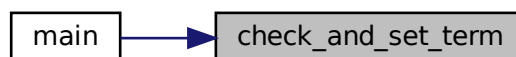
0 si OK, sinon retourne une erreur différente de 0

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 :



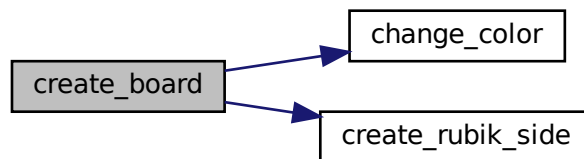
### 6.2.2.3 create\_board()

```
void create_board (
    void )
```

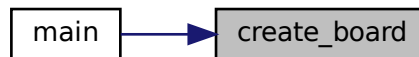
Fonction de test qui permet d'afficher diverses choses sur l'écran nCurses

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 )
```

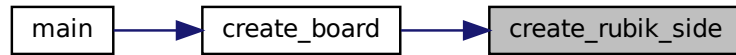
Fonction de test nCurses pour créer la face uniforme d'un cube dans une couleur donnée

#### Paramètres

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

Définition à la ligne 144 du fichier draw.c.

Voici le graphe des appelants de cette fonction :



#### 6.2.2.5 destroy\_board()

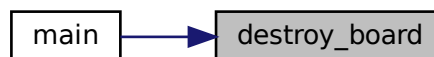
```
void destroy_board (  
    void )
```

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 )
```

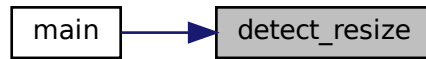
Détection de l'événement de changement de taille d'une fenêtre. L'événement est non géré, pour le moment.

##### Paramètres

<i>dummy</i>	Paramètre inutilisé. Il est passé par la fonction d'appel des événements 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 )
```

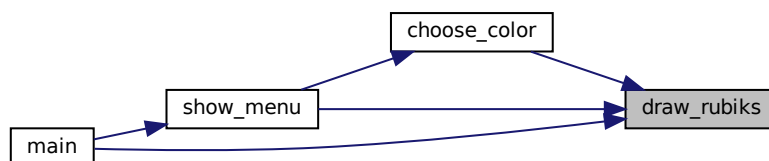
Dessine le rubik's Cube au format texte ANSI (pas au format nCurses)

##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 débogage, permet d'afficher les positions après des mouvements.

## Paramètres

<code>rubiks</code>	Un pointeur vers une structure <a href="#">rubiks_side</a>
---------------------	--

Définition à la ligne 250 du fichier draw.c.

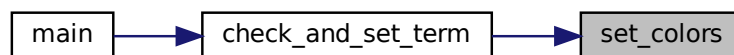
### 6.2.2.9 set\_colors()

```
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

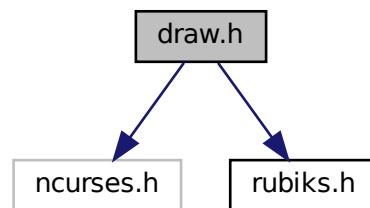
Définition à la ligne 42 du fichier draw.c.

## 6.3 Référence du fichier draw.h

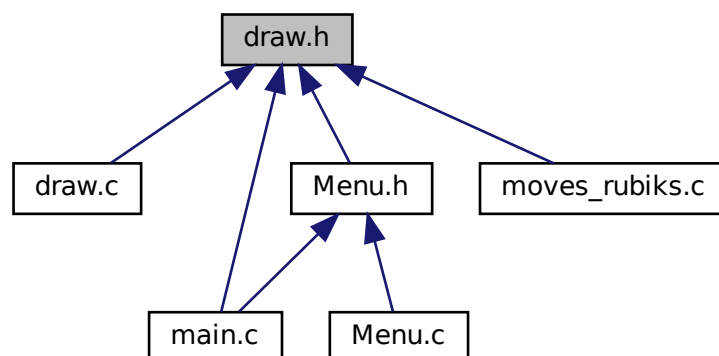
Dans ce fichier, on trouve les headers utilisés dans [draw.c](#).

```
#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 RED "\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](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 :

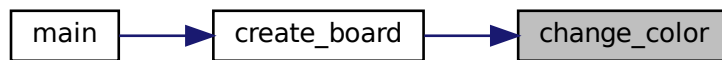
- line, col coordonnées du cube à modifier dans la matrice de l'écran
- cube\_x, cube\_y coordonnées x, y du cube à modifier
- color : Nouvelle couleur à appliquer

#### Paramètres

<i>line</i>	Ligne de la matrices nCurses
<i>col</i>	Colonne de la matrice nCurses
<i>cube</i> ↔ <i>_x</i>	Coordonnée X du cube à modifier
<i>cube</i> ↔ <i>_y</i>	Coordonnée Y du cube à modifier
<i>color</i>	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 )
```

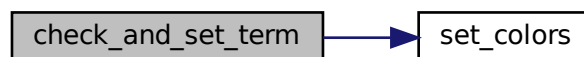
Vérification du terminal, pour voir si il pourra faire tourner le programme avec nCurses

#### Renvoie

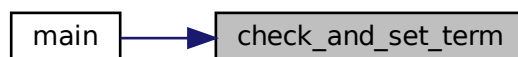
0 si OK, sinon retourne une erreur différente de 0

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 :



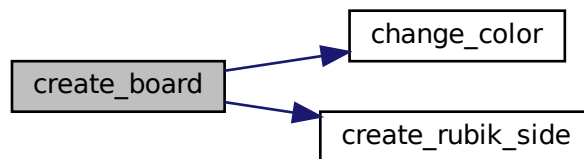
### 6.3.3.3 create\_board()

```
void create_board (
    void )
```

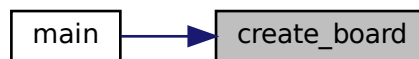
Fonction de test qui permet d'afficher diverses choses sur l'écran nCurses

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

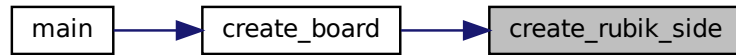
#### Paramètres

<i>line</i>	Ligne de la matrice nCurses dans laquelle dessiner le cube
<i>col</i>	Colonne de la matrice nCurses dans laquelle dessiner le cube
<i>color</i>	Couleur à appliquer sur le cube
<i>name</i>	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()

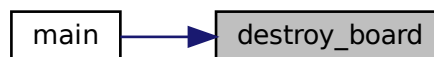
```
void destroy_board (  
    void )
```

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 )
```

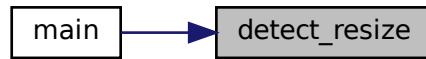
Détection de l'événement de changement de taille d'une fenêtre. L'événement est non géré, pour le moment.

##### Paramètres

<i>dummy</i>	Paramètre inutilisé. Il est passé par la fonction d'appel des événements 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 )
```

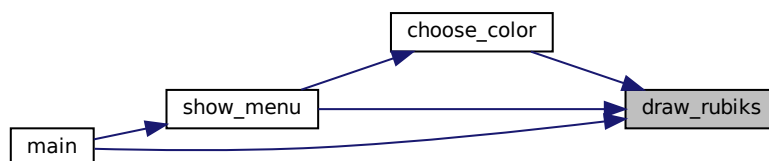
Dessine le rubik's Cube au format texte ANSI (pas au format nCurses)

##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 débogage, permet d'afficher les positions après des mouvements.

## Paramètres

<code>rubiks</code>	Un pointeur vers une structure <a href="#">rubiks_side</a>
---------------------	--

Définition à la ligne 250 du fichier draw.c.

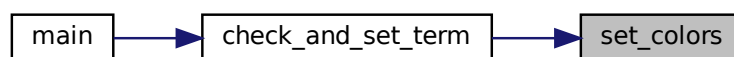
### 6.3.3.9 set\_colors()

```
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

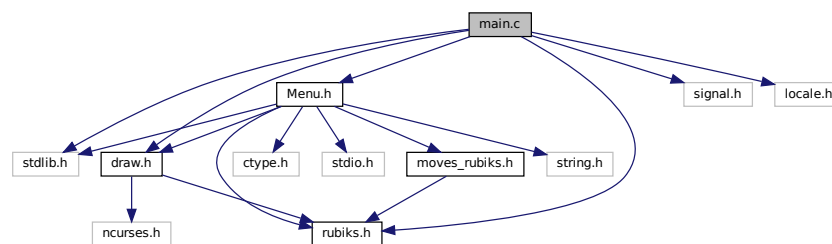
Définition à la ligne 42 du fichier draw.c.

## 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 ( )
```

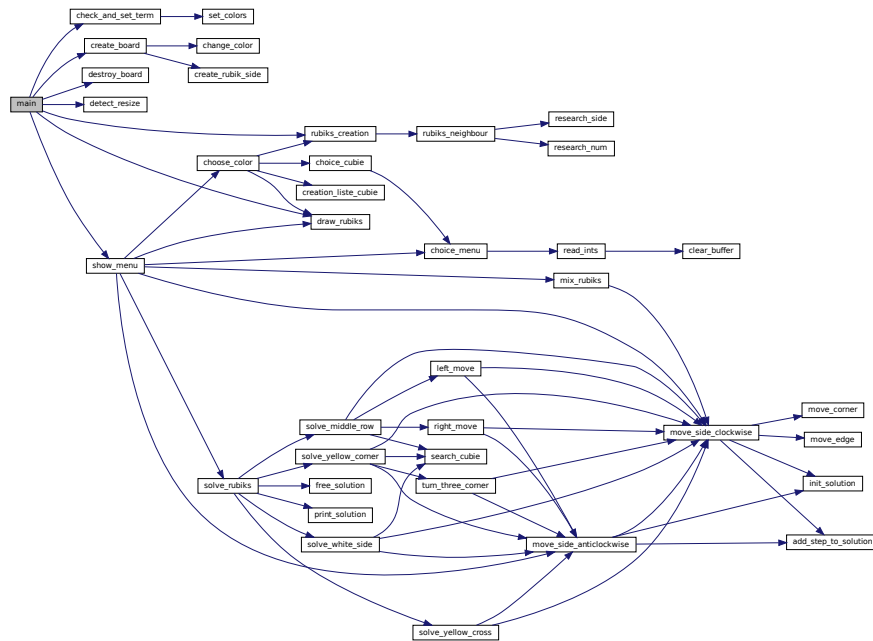
Fonction principale du programme. Elle se contente d'afficher les menus. Soit en mode nCurses (non implémenté pour le moment) soit en mode "texte" avec des couleurs ANSI.

Renvoie

0

Définition à la ligne 26 du fichier main.c.

Voici le graphe d'appel pour cette fonction :

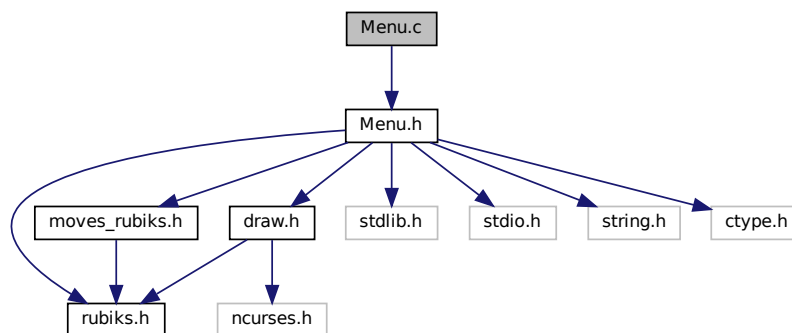


## 6.5 Référence du fichier Menu.c

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

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

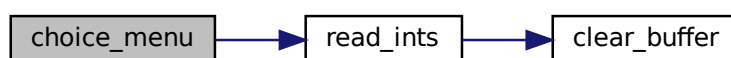
<code>num_entries</code>	Nombre d'entrées dans le menu.
--------------------------	--------------------------------

#### Renvoie

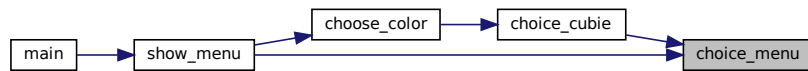
L'entrée choisit par l'utilisateur

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 )
```

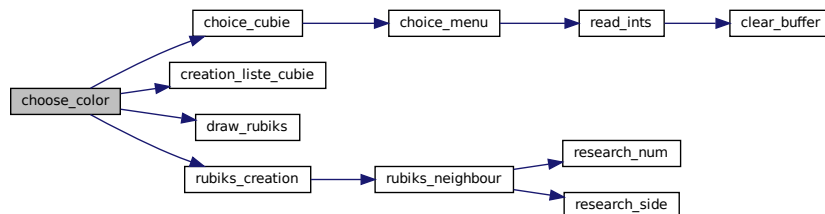
Cette fonction permet à l'utilisateur de remplir un cube

#### Paramètres

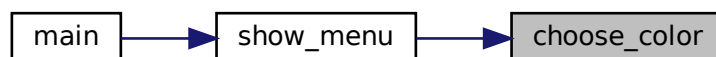
<i>rubiks</i>	Un pointeur vers le rubik's en cours de traitement.
---------------	---

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 :



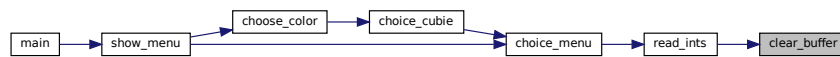
#### 6.5.2.4 clear\_buffer()

```
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

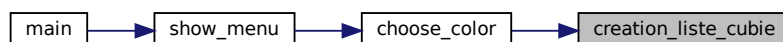
##### Paramètres

<i>rubiks</i>	Pointeur vers le rubik's de travail
<i>liste</i>	Liste des cubies disponibles en sortie
<i>type</i>	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

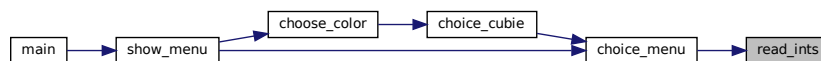
L'entier saisi par l'utilisateur

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 )
```

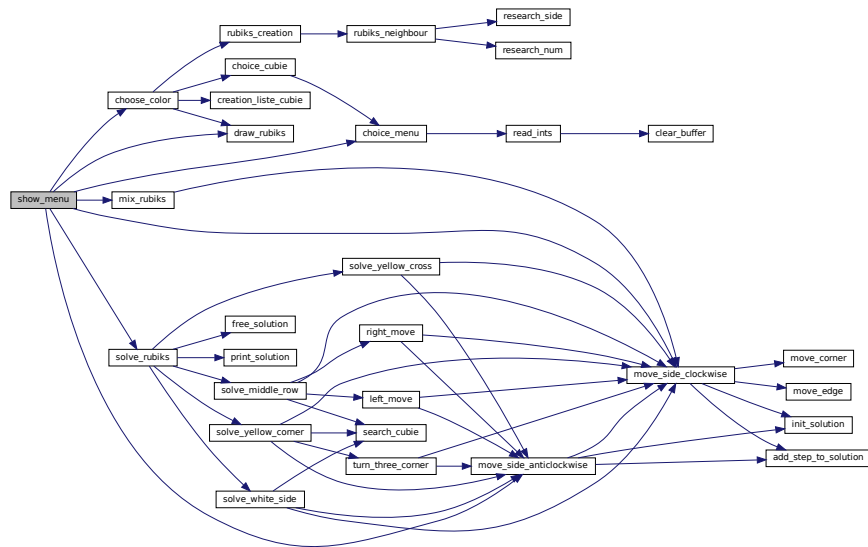
Cette fonction affiche le menu principal et tous les sous-menus.

#### Paramètres

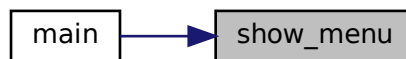
<i>rubiks</i>	Un pointeur vers le rubik's cube actuellement en cours d'utilisation
---------------	--

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

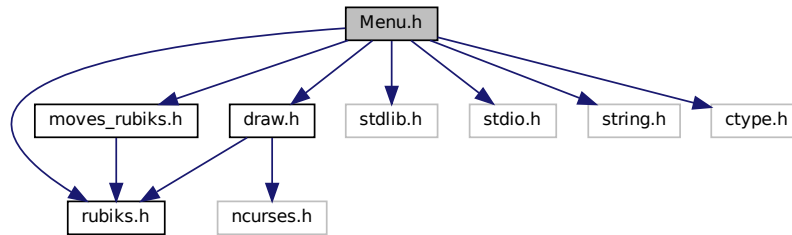
Dans ce fichier, on trouve les headers des fonction définies dans [Menu.c](#).

```

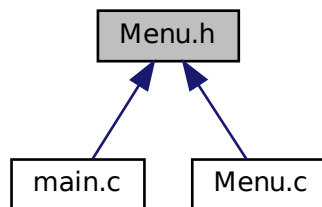
#include "rubiks.h"
#include "moves_rubiks.h"
#include "draw.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

```

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 )
```

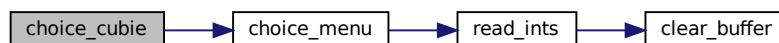
Cette fonction permet de récupérer le choix de l'utilisateur en fonction des cubies disponibles.

#### Paramètres

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

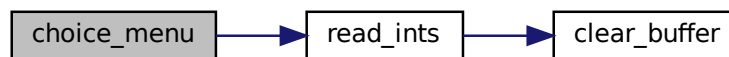
<i>num_entries</i>	Nombre d'entrées dans le menu.
--------------------	--------------------------------

#### Renvoie

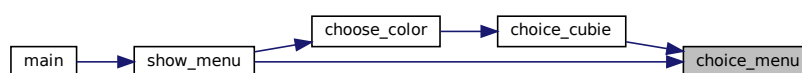
L'entrée choisit par l'utilisateur

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

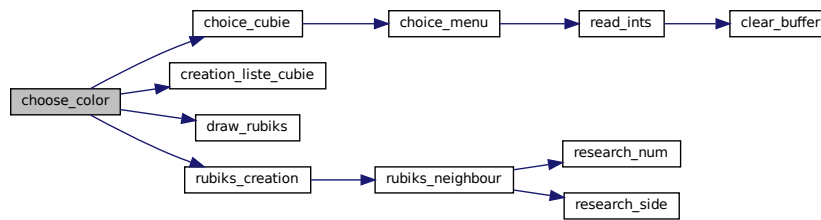
#### Paramètres

<i>rubiks</i>	Un pointeur vers le rubik's en cours de traitement.
---------------	---

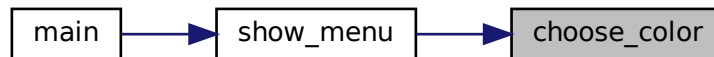


Définition à la ligne 114 du fichier Menu.c.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



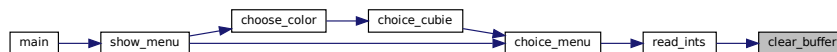
#### 6.6.2.4 clear\_buffer()

```
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

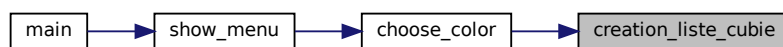
## Paramètres

<i>rubiks</i>	Pointeur vers le rubik's de travail
<i>liste</i>	Liste des cubies disponibles en sortie
<i>type</i>	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

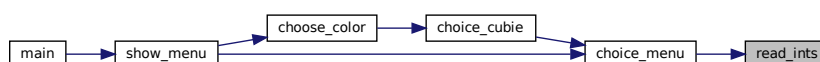
L'entier saisi par l'utilisateur

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 )
```

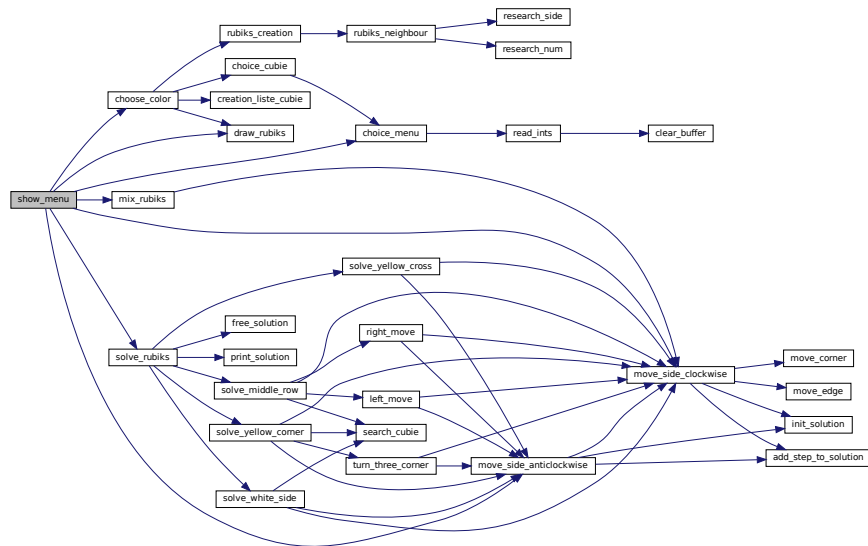
Cette fonction affiche le menu principal et tous les sous-menus.

#### Paramètres

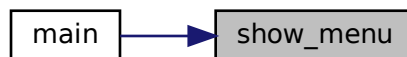
<i>rubiks</i>	Un pointeur vers le rubik's cube actuellement en cours d'utilisation
---------------	--

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 :

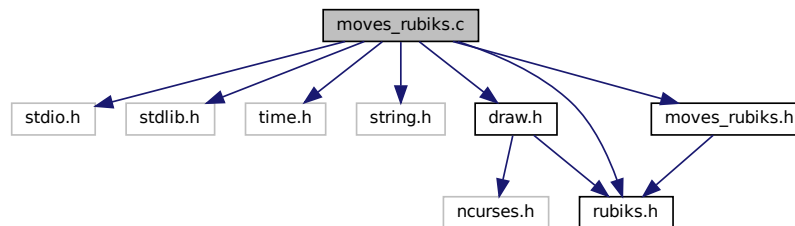


## 6.7 Référence du fichier moves\_rubiks.c

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] )
```

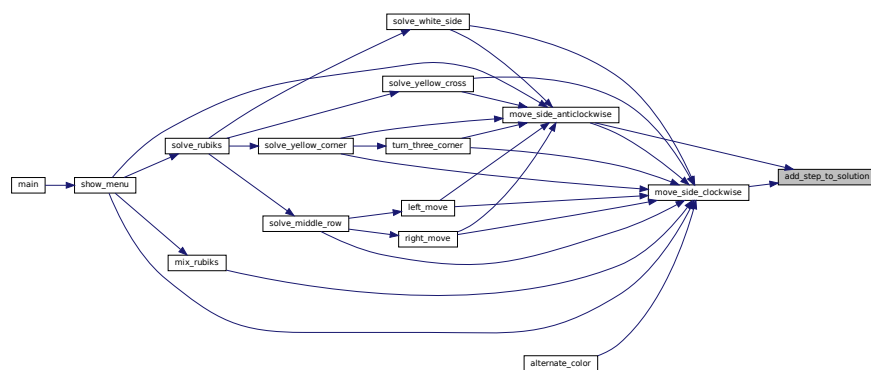
Cette fonction ajoute une nouvelle étape de résolution à un historique déjà existant.

#### Paramètres

<i>first_step</i>	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution
<i>new_step</i>	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 )
```

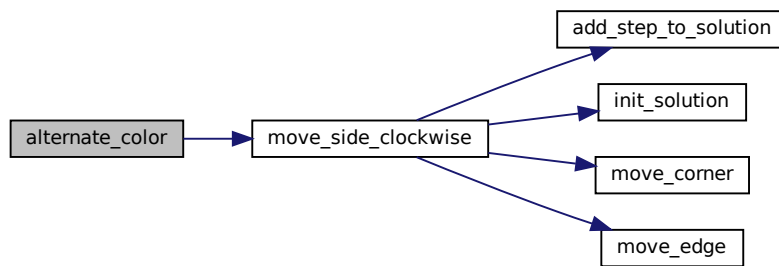
Cette fonction permet de créer un damier de couleurs

#### Paramètres

<i>rubiks</i>	un pointeur vers une structure <code>rubiks_side</code>
---------------	---

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 chaînée des étapes d'une solution.

#### Paramètres

<i>first_step</i>	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

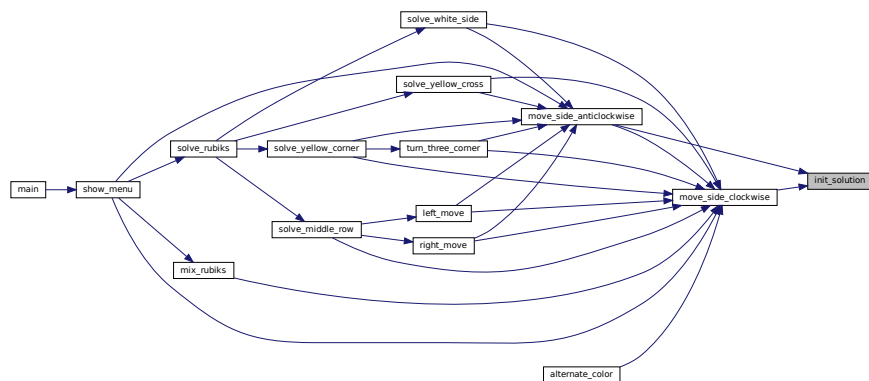
<i>new_step</i>	Description du premier mouvement, au format texte
-----------------	---

## 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 )

```

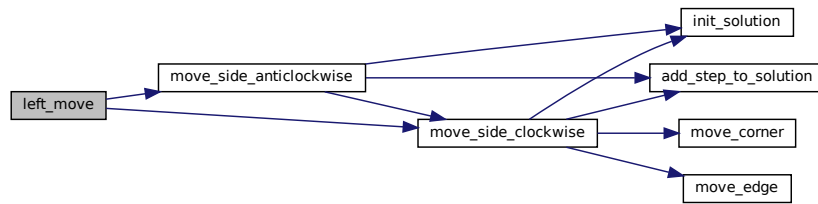
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne. Déplacement vers le côté gauche. Utilisée lors de la résolution de la 2ème couronne.

## Paramètres

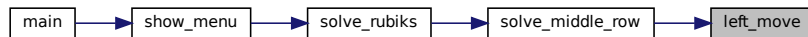
<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>cubie</i>	cubie situé au dessus de l'arête à déplacer

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.

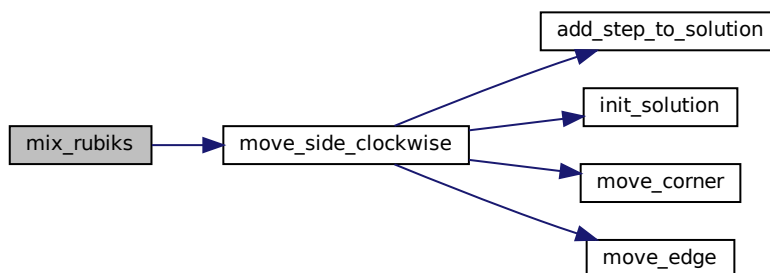
##### Paramètres

<i>rubiks</i>	un pointeur vers une structure <code>rubiks_side</code>
---------------	---

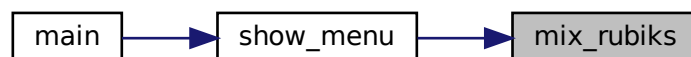
Définition à la ligne 133 du fichier `moves_rubiks.c`.



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 )
  
```

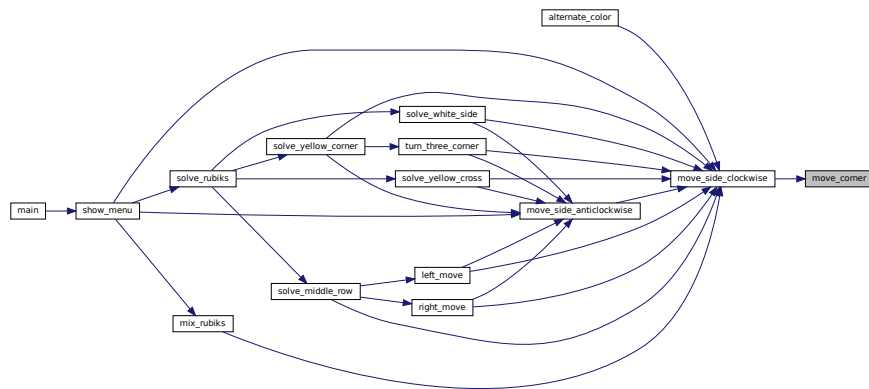
Cette fonction permet de déplacer un coin du cube sur la face "side", depuis "from" vers "to".

##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>side</i>	La face du rubiks à traiter
<i>from</i>	Déplacer depuis ce cubie
<i>to</i>	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".

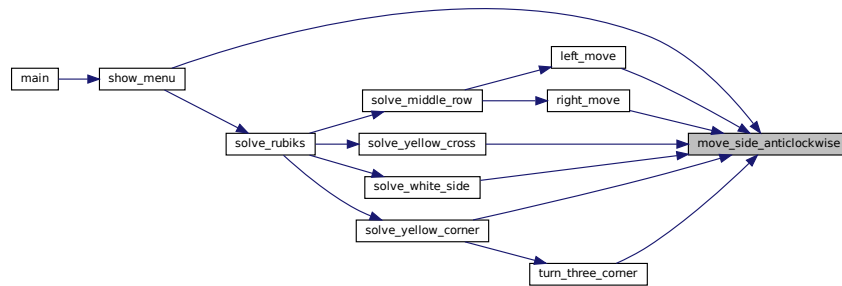
##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>side</i>	La face du rubiks à traiter
<i>from</i>	Déplacer depuis ce cubie
<i>to</i>	Déplacer vers ce cubie

Définition à la ligne 45 du fichier moves\_rubiks.c.



Voici le graphe des appelants de cette fonction :



#### 6.7.2.10 move\_side\_clockwise()

```

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

```

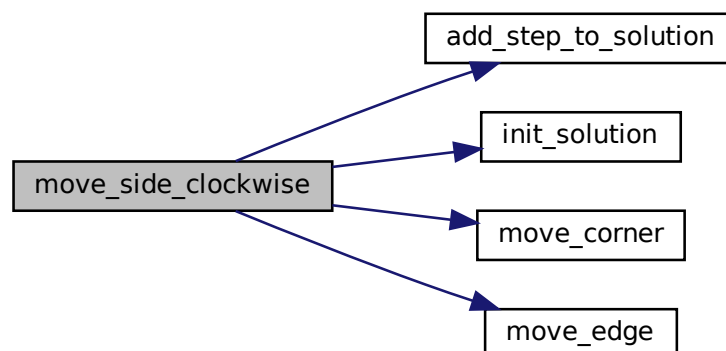
cette fonction permet de faire tourner la face side du cube dans le sens horaire

##### Paramètres

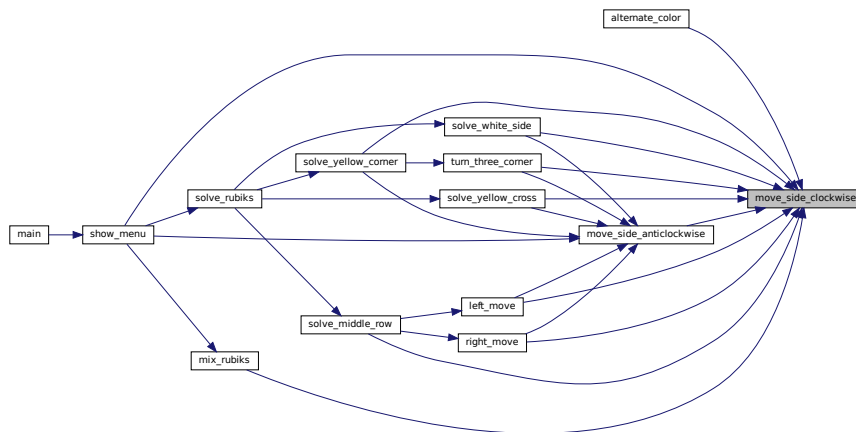
<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>side</i>	La face à faire tourner dans le sens horaire
<i>add_to_history</i>	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

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

<i>first_step</i>	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 )
```

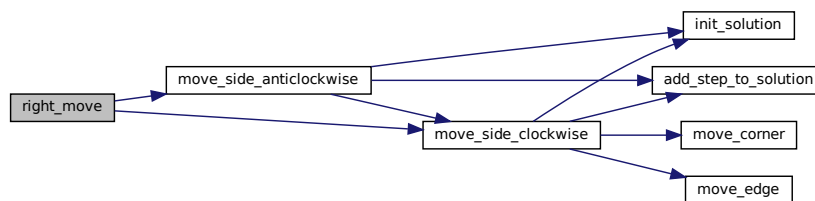
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne. Déplacement vers le côté droit. Utilisée lors de la résolution de la 2ème couronne.

#### Paramètres

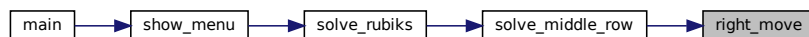
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>cubie</i>	cubie situé au dessus de l'arrête à déplacer

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

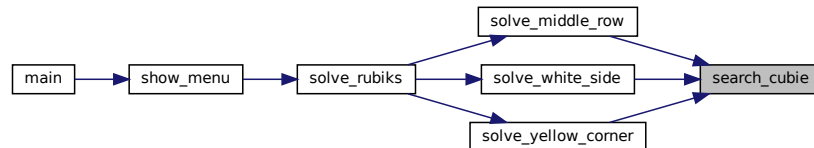
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>cubie_color</i>	couleur du cubie que l'on cherche
<i>neighbour1</i>	couleur du premier voisin du cubie recherché
<i>cubie_type</i>	type du cubie recherché

**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 )
```

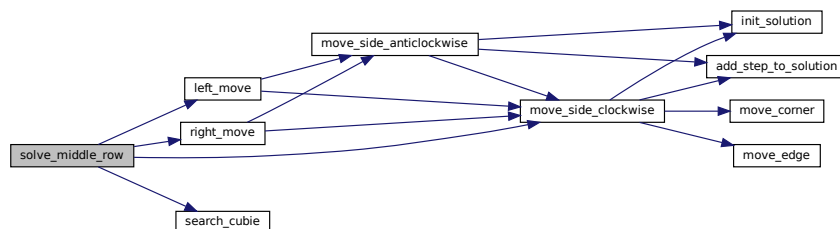
Cette fonction cherche à résoudre la 2ème couronne du rubik's cube Le choix est fait de ne pas "retourner" le rubiks cube et d'adapter les algorithmes en conséquence.

**Paramètres**

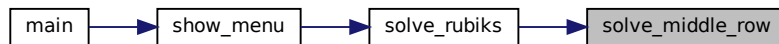
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

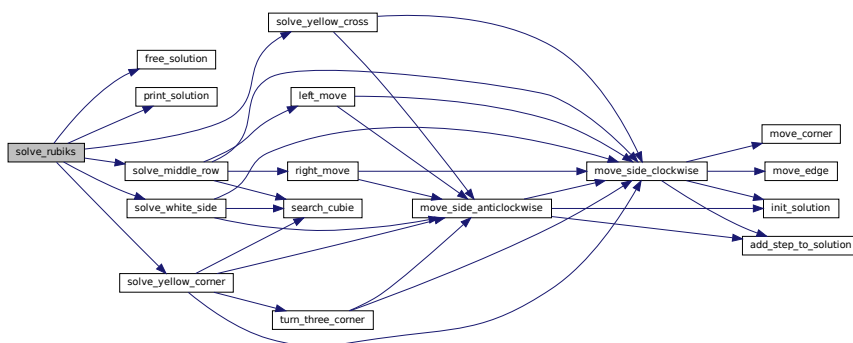
Cette fonction permet de résoudre le cube.

#### Paramètres

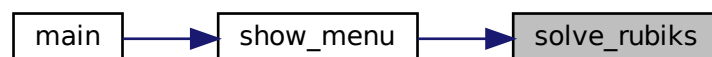
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

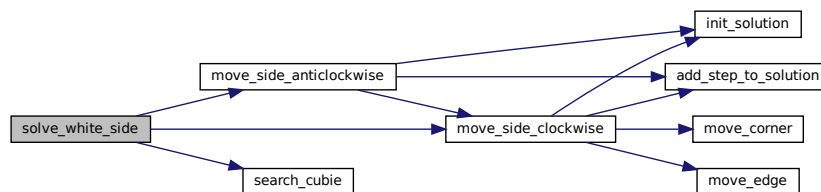
Cette fonction cherche à résoudre la face blanche du Rubik's Cube.

#### Paramètres

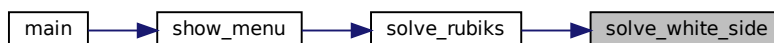
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

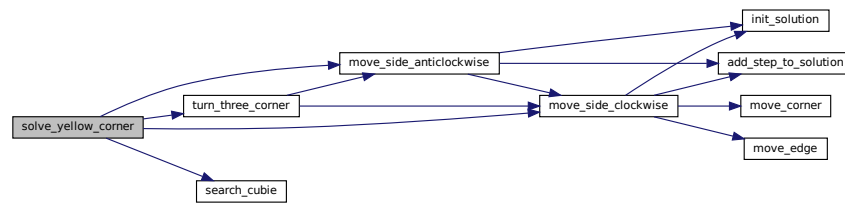
Cette fonction permet de résoudre les coins jaunes

#### Paramètres

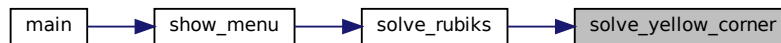
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks</i>
---------------	--

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 )
```

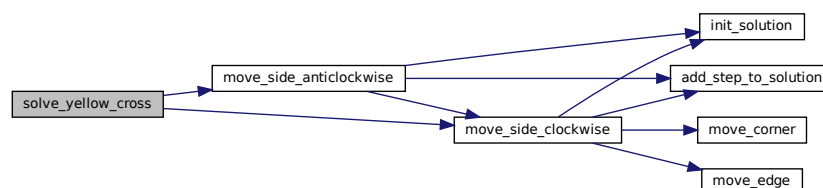
Cette fonction résoud la croix jaune.

##### Paramètres

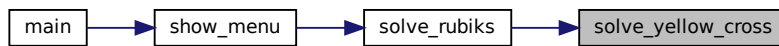
<i>rubiks</i>	Un pointeur vers une structure rubiks
---------------	---------------------------------------

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 )
```

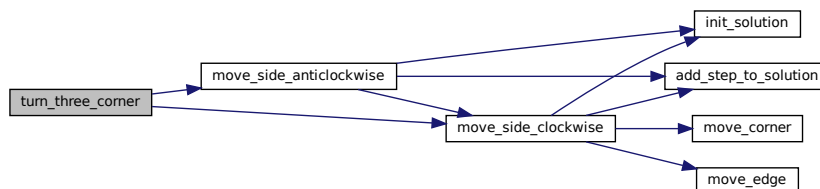
Cette fonction permet de faire tourner trois côtés d'une même face sans bouger le 4em et le reste du cube.

#### Paramètres

<i>Rubiks</i>	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
```

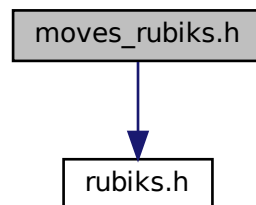
Définition à la ligne 17 du fichier moves\_rubiks.c.

## 6.8 Référence du fichier moves\_rubiks.h

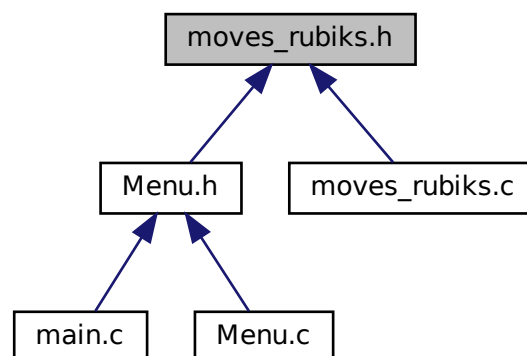
Dans ce fichier, on trouve les headers utilisés partout dans le code. Définitions dans [moves\\_rubiks.c](#).

```
#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] )
```

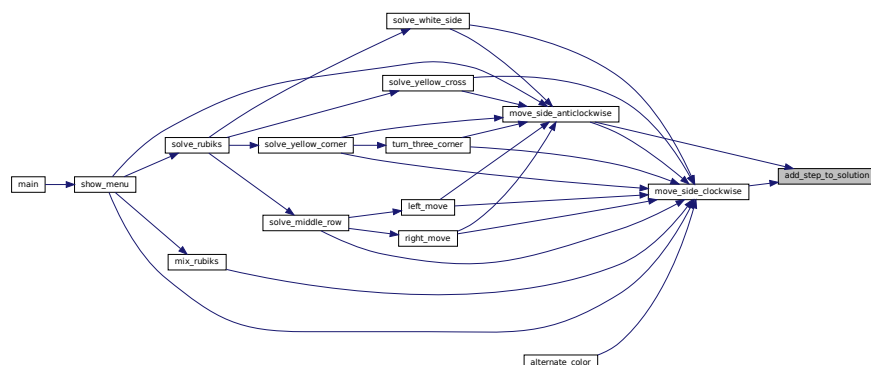
Cette fonction ajoute une nouvelle étape de résolution à un historique déjà existant.

##### Paramètres

<i>first_step</i>	Pointeur vers le premier élément de la liste chaînée de l'historique des étapes de résolution
<i>new_step</i>	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 )
```

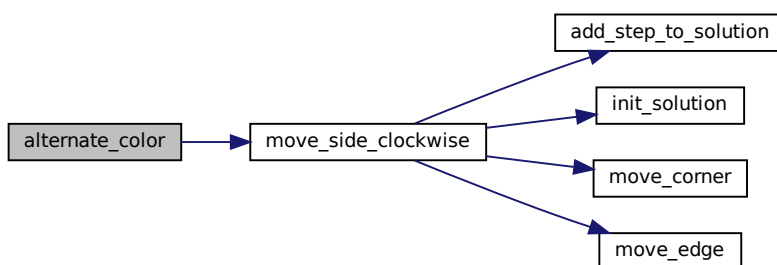
Cette fonction permet de créer un damier de couleurs

#### Paramètres

<i>rubiks</i>	un pointeur vers une structure <code>rubiks_side</code>
---------------	---

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 chaînée des étapes d'une solution.

#### Paramètres

<i>first_step</i>	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

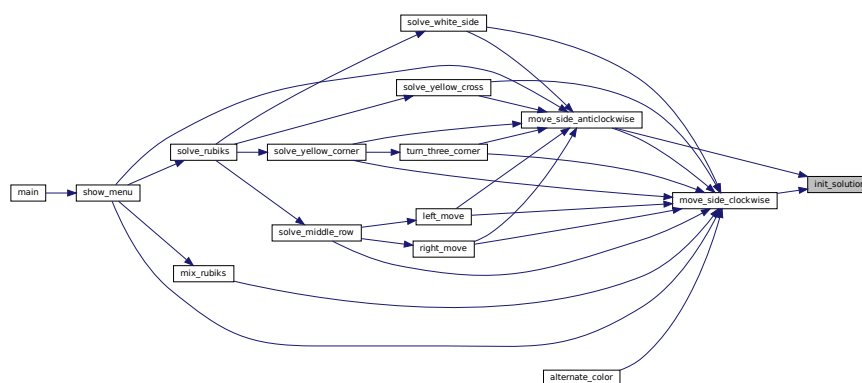
<i>new_step</i>	Description du premier mouvement, au format texte
-----------------	---

##### 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 )
```

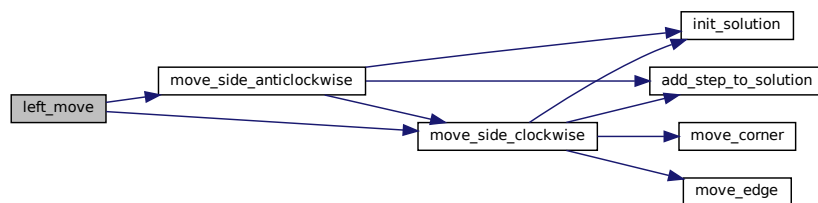
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne. Déplacement vers le côté gauche. Utilisée lors de la résolution de la 2ème couronne.

##### Paramètres

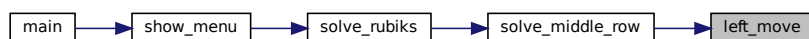
<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>cubie</i>	cubie situé au dessus de l'arête à déplacer

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 )
```

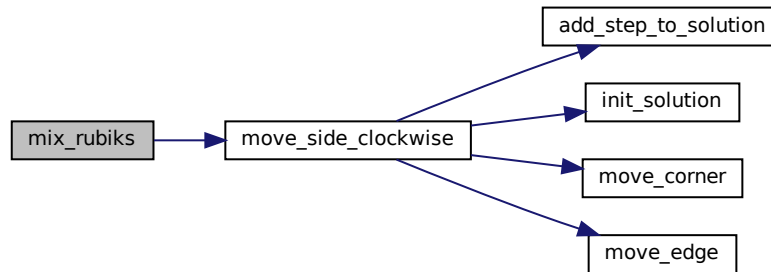
Cette fonction permet de mélanger le cube de manière aléatoire. Elle choisit entre 20 et 30 mouvements à réaliser tout en choisissant une face à déplacer, au hasard.

##### Paramètres

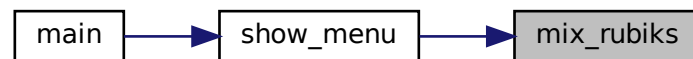
<i>rubiks</i>	un pointeur vers une structure <code>rubiks_side</code>
---------------	---

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 )
  
```

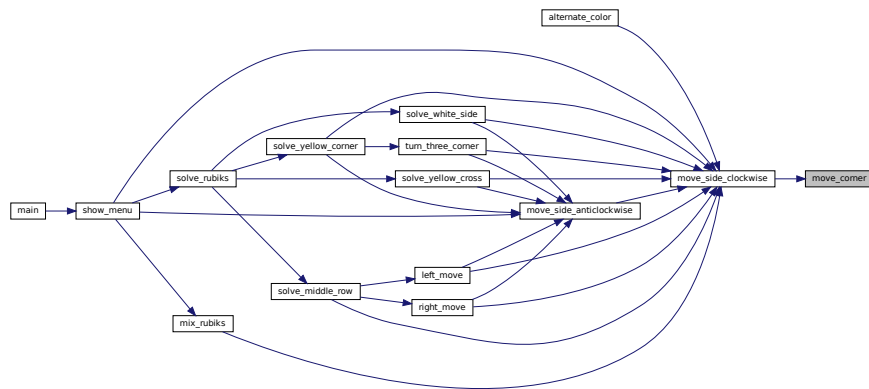
Cette fonction permet de déplacer un coin du cube sur la face "side", depuis "from" vers "to".

##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>side</i>	La face du rubiks à traiter
<i>from</i>	Déplacer depuis ce cubie
<i>to</i>	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 )
  
```

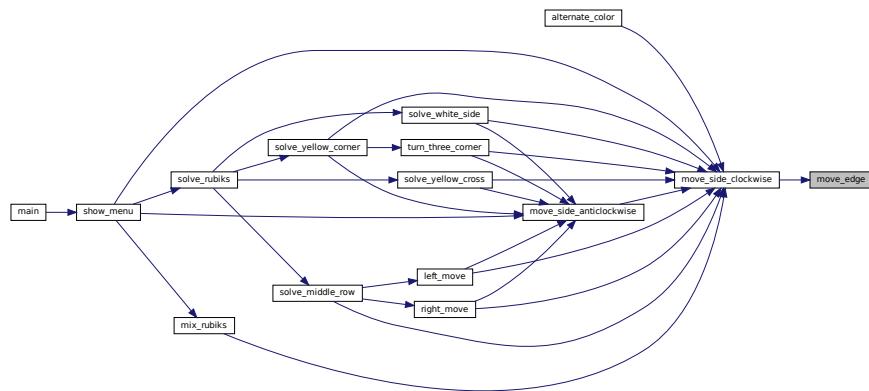
Cette fonction permet de déplacer une arête sur la face "side" depuis "from" vers "to".

##### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <code>rubiks_side</code>
<i>side</i>	La face du rubiks à traiter
<i>from</i>	Déplacer depuis ce cubie
<i>to</i>	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 )
  
```

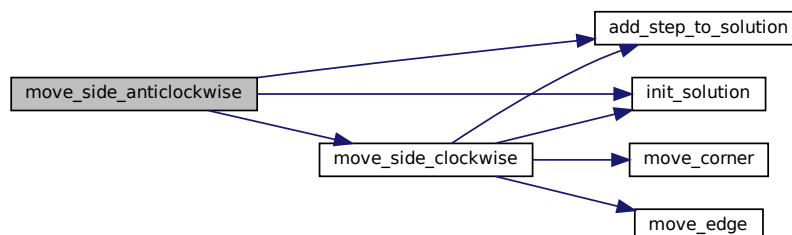
Cette fonction fait tourner la face side dans le sens anti-horaire. En réalité, cela revient à tourner trois fois dans le sens horaire.

##### Paramètres

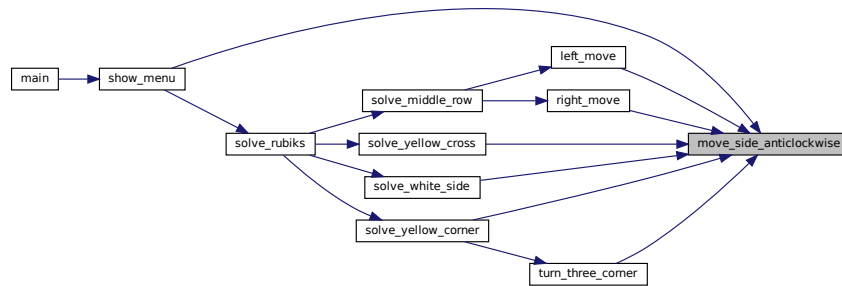
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>side</i>	La face à faire tourner dans le sens horaire
<i>add_to_history</i>	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

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 )

```

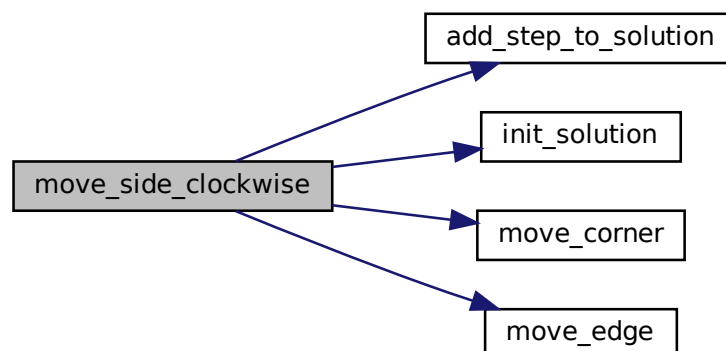
cette fonction permet de faire tourner la face side du cube dans le sens horaire

##### Paramètres

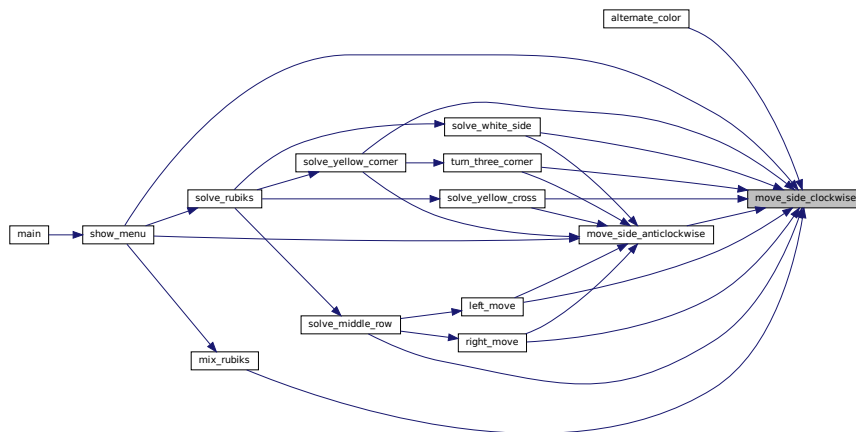
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>side</i>	La face à faire tourner dans le sens horaire
<i>add_to_history</i>	Si true, on ajoute le mouvement à l'historique des étapes de la résolution du cube.

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

<i>first_step</i>	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 )
```

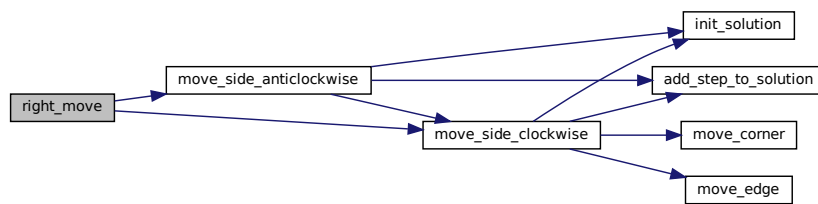
Cette fonction permet de déplacer une arête correctement positionné de la 3ème couronne à la 2ème couronne  
Déplacement vers le côté droit. Utilisée lors de la résolution de la 2ème couronne.

#### Paramètres

<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>cubie</i>	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

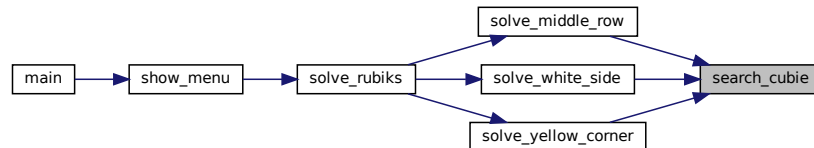
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
<i>cubie_color</i>	couleur du cubie que l'on cherche
<i>neighbour1</i>	couleur du premier voisin du cubie recherché
<i>cubie_type</i>	type du cubie recherché

**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 )
```

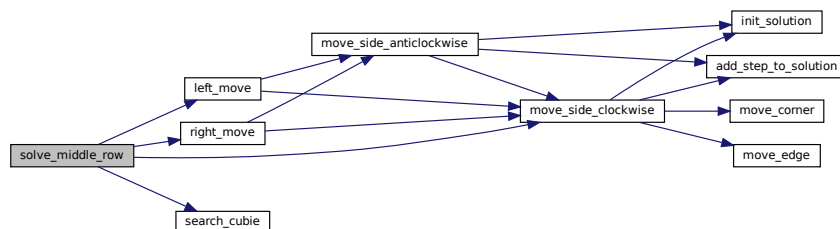
Cette fonction cherche à résoudre la 2ème couronne du rubik's cube Le choix est fait de ne pas "retourner" le rubiks cube et d'adapter les algorithmes en conséquence.

**Paramètres**

<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

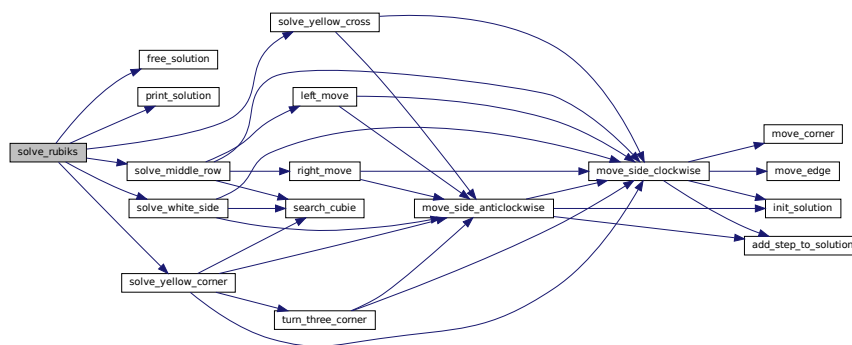
Cette fonction permet de résoudre le cube.

##### Paramètres

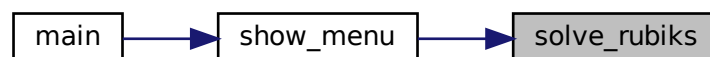
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

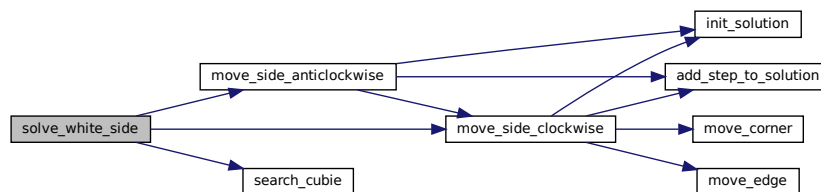
Cette fonction cherche à résoudre la face blanche du Rubik's Cube.

##### Paramètres

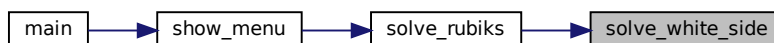
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

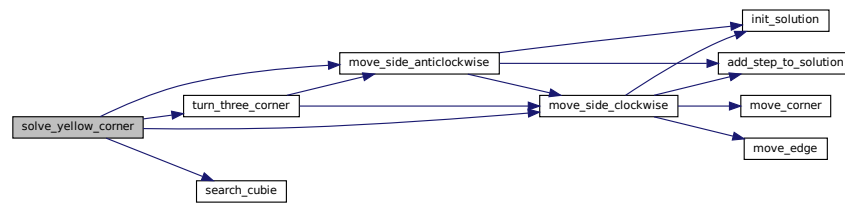
Cette fonction permet de résoudre les coins jaunes

##### Paramètres

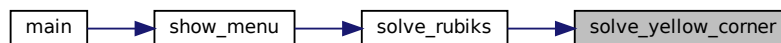
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks</i>
---------------	--

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 )
```

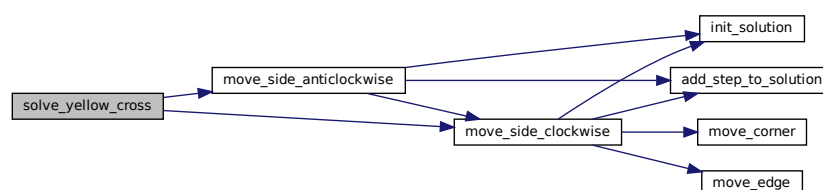
Cette fonction résoud la croix jaune.

##### Paramètres

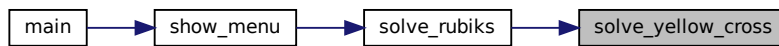
<i>rubiks</i>	Un pointeur vers une structure rubiks
---------------	---------------------------------------

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 )
```

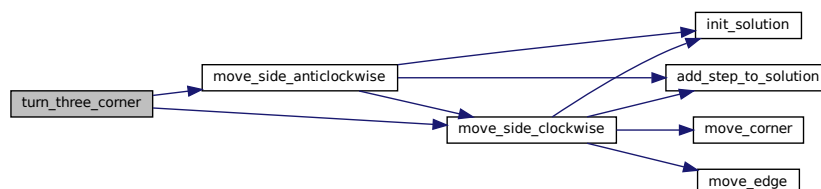
Cette fonction permet de faire tourner trois côtés d'une même face sans bouger le 4em et le reste du cube.

##### Paramètres

<i>Rubiks</i>	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 :



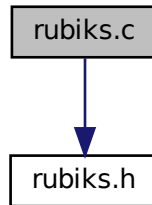
## 6.9 Référence du fichier README.md

## 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

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

## Renvoi

Renvoie l'indice de la face trouvée, -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 absolue d'une face (neighbour\_side) la position relative qu'elle occupe. exemple : si je souhaite retrouver la position de GREEN par rapport à ORANGE la fonction renvoie LEFT.

## Paramètres

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

## Renvoi

Renvoie 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 :

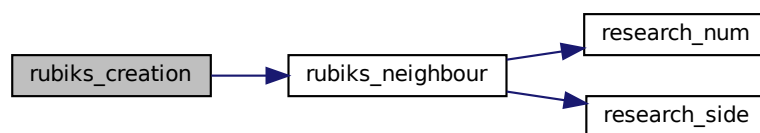
- La face au dessus est la face bleue
- La face à sa droite est la face rouge
- La face à sa gauche est la face orange
- La face sous-elle est la face verte
- La face opposée ne change jamais, dans le cas de la blanche, il s'agit de la face jaune.

#### Paramètres

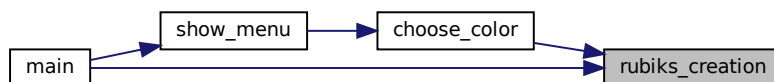
<code>rubiks</code>	Un pointeur vers une structure <code>rubiks_side</code>
---------------------	---

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 )
```

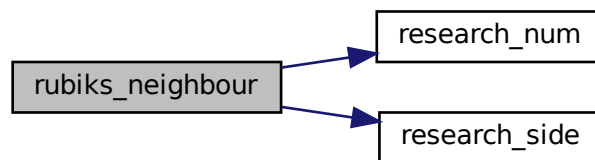
Cette fonction permet d'attribuer à chaque cubie ses voisins. num\_side indique sur quelle face il se trouve num\_cubie correspond au numéro du cubie voisin sur sa face les voisins ne changent pas avec la réalisation de mouvements ils sont donc attribués définitivement.

##### Paramètres

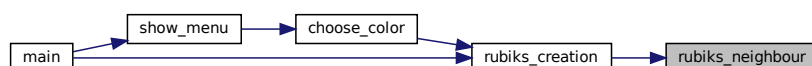
<i>rubiks</i>	Un pointeur vers une structure <a href="#">rubiks_side</a>
---------------	--

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 :

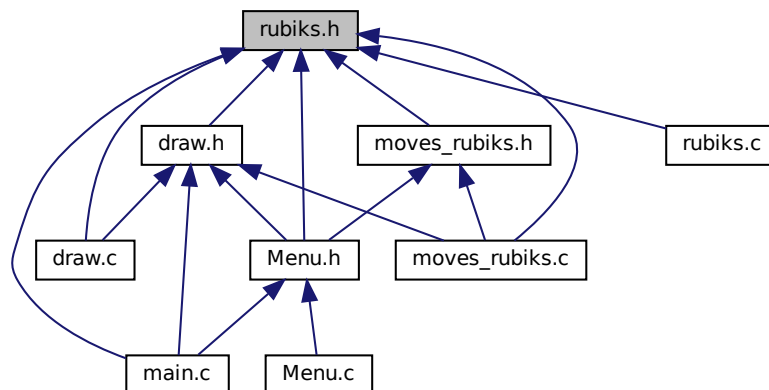


## 6.11 Référence du fichier rubiks.h

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

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

## Renvoie

Renvoie l'indice de la cubie trouvée, -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

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

#### Renvoie

Renvoie 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 :

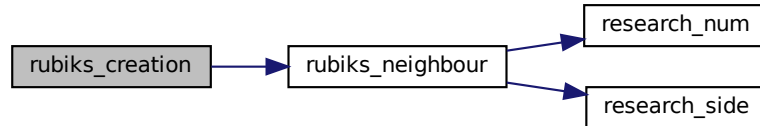
- La face au dessus est la face bleue
- La face à sa droite est la face rouge
- La face à sa gauche est la face orange
- La face sous-elle est la face verte
- La face opposée ne change jamais, dans le cas de la blanche, il s'agit de la face jaune.

#### Paramètres

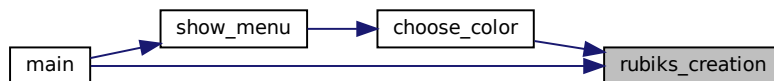
<i>rubiks</i>	Un pointeur vers une structure <i>rubiks_side</i>
---------------	---

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 )
```

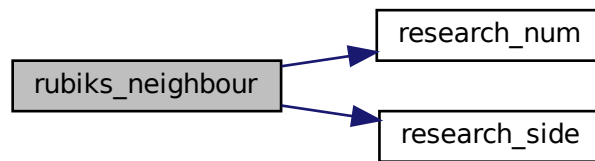
Cette fonction permet d'attribuer à chaque cubie ses voisins. `num_side` indique sur quelle face il se trouve `num_cubie` correspond au numéro du cubie voisin sur sa face les voisins ne changent pas avec la réalisation de mouvements ils sont donc attribués définitivement.

##### Paramètres

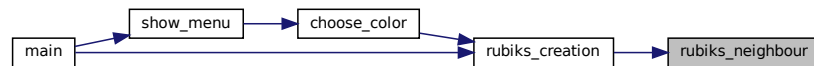
<code>rubiks</code>	Un pointeur vers une structure <code>rubiks_side</code>
---------------------	---

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

add\_step\_to\_solution  
    moves\_rubiks.c, [53](#)  
    moves\_rubiks.h, [70](#)  
alternate\_color  
    moves\_rubiks.c, [53](#)  
    moves\_rubiks.h, [70](#)  
ARED  
    draw.h, [27](#)  
  
BLACK\_ON\_BLUE  
    draw.c, [23](#)  
    draw.h, [35](#)  
BLACK\_ON\_GREEN  
    draw.c, [23](#)  
    draw.h, [35](#)  
BLACK\_ON\_ORANGE  
    draw.c, [23](#)  
    draw.h, [35](#)  
BLACK\_ON\_RED  
    draw.c, [24](#)  
    draw.h, [36](#)  
BLACK\_ON\_WHITE  
    draw.c, [24](#)  
    draw.h, [36](#)  
BLACK\_ON\_YELLOW  
    draw.c, [24](#)  
    draw.h, [36](#)  
BLU  
    draw.h, [27](#)  
BLUE  
    rubiks.h, [90](#)  
BOARD  
    draw.c, [24](#)  
    draw.h, [36](#)  
  
CENTER  
    rubiks.h, [90](#)  
change\_color  
    draw.c, [18](#)  
    draw.h, [30](#)  
check\_and\_set\_term  
    draw.c, [19](#)  
    draw.h, [31](#)  
choice\_cubie  
    Menu.c, [40](#)  
    Menu.h, [47](#)  
choice\_menu  
    Menu.c, [41](#)  
    Menu.h, [47](#)  
choose\_color  
    Menu.c, [42](#)  
    Menu.h, [48](#)  
clear\_buffer  
    Menu.c, [42](#)  
    Menu.h, [49](#)  
color  
    cubies, [10](#)  
CORNER  
    rubiks.h, [90](#)  
create\_board  
    draw.c, [19](#)  
    draw.h, [31](#)  
create\_rubik\_side  
    draw.c, [20](#)  
    draw.h, [32](#)  
creation\_liste\_cubie  
    Menu.c, [43](#)  
    Menu.h, [49](#)  
cubie  
    rubiks\_side, [14](#)  
cubie\_side  
    cubies, [10](#)  
cubies, [9](#)  
    color, [10](#)  
    cubie\_side, [10](#)  
    neighbours, [10](#)  
    num, [10](#)  
    type, [10](#)  
    x, [11](#)  
    y, [11](#)  
  
destroy\_board  
    draw.c, [21](#)  
    draw.h, [33](#)  
detect\_resize  
    draw.c, [21](#)  
    draw.h, [33](#)  
DOWN  
    rubiks.h, [91](#)  
Doxygen-installation-usage.md, [17](#)  
draw.c, [17](#)  
    BLACK\_ON\_BLUE, [23](#)  
    BLACK\_ON\_GREEN, [23](#)  
    BLACK\_ON\_ORANGE, [23](#)  
    BLACK\_ON\_RED, [24](#)  
    BLACK\_ON\_WHITE, [24](#)  
    BLACK\_ON\_YELLOW, [24](#)  
    BOARD, [24](#)  
    change\_color, [18](#)  
    check\_and\_set\_term, [19](#)

- create\_board, 19
- create\_rubik\_side, 20
- destroy\_board, 21
- detect\_resize, 21
- draw\_rubiks, 22
- NB\_COLS, 24
- NB\_LINES, 25
- rubiks\_display, 22
- set\_colors, 23
- draw.h, 25
  - ARED, 27
  - BLACK\_ON\_BLUE, 35
  - BLACK\_ON\_GREEN, 35
  - BLACK\_ON\_ORANGE, 35
  - BLACK\_ON\_RED, 36
  - BLACK\_ON\_WHITE, 36
  - BLACK\_ON\_YELLOW, 36
  - BLU, 27
  - BOARD, 36
  - change\_color, 30
  - check\_and\_set\_term, 31
  - create\_board, 31
  - create\_rubik\_side, 32
  - destroy\_board, 33
  - detect\_resize, 33
  - draw\_rubiks, 34
  - GRN, 27
  - GRY, 27
  - MIN\_COLORS\_NUMBER, 27
  - NB\_COLS, 36
  - NB\_LINES, 37
  - ORG, 27
  - RESET, 28
  - RUBIK\_COLS, 28
  - RUBIK\_LINES, 28
  - rubiks\_display, 34
  - set\_colors, 35
  - SQ\_HEIGHT, 28
  - SQ\_WIDTH, 28
  - SQUARES, 29
  - TERM\_HAS\_NO\_COLORS, 29
  - TERM\_HAS\_NOT\_ENOUGH\_COLORS, 29
  - TERM\_NOT\_BIG\_ENOUGH, 29
  - WHT, 29
  - YEL, 30
- draw\_rubiks
  - draw.c, 22
  - draw.h, 34
- EDGE
  - rubiks.h, 90
- free\_solution
  - moves\_rubiks.c, 54
  - moves\_rubiks.h, 71
- GREEN
  - rubiks.h, 90
- GREY
  - rubiks.h, 90
- GRN
  - draw.h, 27
- GRY
  - draw.h, 27
- history
  - moves\_rubiks.c, 67
- init\_solution
  - moves\_rubiks.c, 54
  - moves\_rubiks.h, 72
- LEFT
  - rubiks.h, 91
- left\_move
  - moves\_rubiks.c, 55
  - moves\_rubiks.h, 72
- main
  - main.c, 38
- main.c, 37
  - main, 38
  - PRINT\_TEXT\_ONLY, 38
- Menu.c, 39
  - choice\_cubie, 40
  - choice\_menu, 41
  - choose\_color, 42
  - clear\_buffer, 42
  - creation\_liste\_cubie, 43
  - read\_ints, 43
  - show\_menu, 44
- Menu.h, 45
  - choice\_cubie, 47
  - choice\_menu, 47
  - choose\_color, 48
  - clear\_buffer, 49
  - creation\_liste\_cubie, 49
  - read\_ints, 50
  - show\_menu, 50
- MIN\_COLORS\_NUMBER
  - draw.h, 27
- mix\_rubiks
  - moves\_rubiks.c, 56
  - moves\_rubiks.h, 73
- move\_corner
  - moves\_rubiks.c, 57
  - moves\_rubiks.h, 74
- move\_edge
  - moves\_rubiks.c, 58
  - moves\_rubiks.h, 75
- move\_side\_anticlockwise
  - moves\_rubiks.c, 59
  - moves\_rubiks.h, 76
- move\_side\_clockwise
  - moves\_rubiks.c, 60
  - moves\_rubiks.h, 77
- moves\_rubiks.c, 51
  - add\_step\_to\_solution, 53



- alternate\_color, 53
- free\_solution, 54
- history, 67
- init\_solution, 54
- left\_move, 55
- mix\_rubiks, 56
- move\_corner, 57
- move\_edge, 58
- move\_side\_anticlockwise, 59
- move\_side\_clockwise, 60
- print\_solution, 61
- right\_move, 61
- search\_cubie, 62
- solve\_middle\_row, 63
- solve\_rubiks, 64
- solve\_white\_side, 64
- solve\_yellow\_corner, 65
- solve\_yellow\_cross, 66
- turn\_three\_corner, 67
- moves\_rubiks.h, 68
  - add\_step\_to\_solution, 70
  - alternate\_color, 70
  - free\_solution, 71
  - init\_solution, 72
  - left\_move, 72
  - mix\_rubiks, 73
  - move\_corner, 74
  - move\_edge, 75
  - move\_side\_anticlockwise, 76
  - move\_side\_clockwise, 77
  - print\_solution, 78
  - right\_move, 78
  - search\_cubie, 79
  - SIZE\_MOVE, 69
  - solutions\_steps, 70
  - solve\_middle\_row, 80
  - solve\_rubiks, 81
  - solve\_white\_side, 81
  - solve\_yellow\_corner, 82
  - solve\_yellow\_cross, 83
  - turn\_three\_corner, 84
- NB\_COLS
  - draw.c, 24
  - draw.h, 36
- NB\_LINES
  - draw.c, 25
  - draw.h, 37
- neighbour, 11
  - num\_cubie, 12
  - num\_side, 12
- neighbour\_side
  - rubiks\_side, 14
- neighbours
  - cubies, 10
- next\_step
  - solutions\_steps, 15
- NO\_COLOR
  - rubiks.h, 90
- num
  - cubies, 10
- num\_cubie
  - neighbour, 12
- num\_side
  - neighbour, 12
- opposite\_side
  - rubiks\_side, 14
- ORANGE
  - rubiks.h, 90
- ORG
  - draw.h, 27
- print\_solution
  - moves\_rubiks.c, 61
  - moves\_rubiks.h, 78
- PRINT\_TEXT\_ONLY
  - main.c, 38
- read\_ints
  - Menu.c, 43
  - Menu.h, 50
- README.md, 84
- RED
  - rubiks.h, 90
- research\_num
  - rubiks.c, 85
  - rubiks.h, 91
- research\_side
  - rubiks.c, 86
  - rubiks.h, 91
- RESET
  - draw.h, 28
- RIGHT
  - rubiks.h, 91
- right\_move
  - moves\_rubiks.c, 61
  - moves\_rubiks.h, 78
- RUBIK\_COLS
  - draw.h, 28
- RUBIK\_LINES
  - draw.h, 28
- rubiks.c, 84
  - research\_num, 85
  - research\_side, 86
  - rubiks\_creation, 87
  - rubiks\_neighbour, 87
- rubiks.h, 88
  - BLUE, 90
  - CENTER, 90
  - CORNER, 90
  - DOWN, 91
  - EDGE, 90
  - GREEN, 90
  - GREY, 90
  - LEFT, 91
  - NO\_COLOR, 90
  - ORANGE, 90

- RED, 90
- research\_num, 91
- research\_side, 91
- RIGHT, 91
- rubiks\_creation, 92
- rubiks\_neighbour, 93
- T\_COLOR, 90
- T\_CUBIE\_TYPE, 90
- T\_SIDE, 90
- UP, 91
- WHITE, 90
- YELLOW, 90
- rubiks\_creation
  - rubiks.c, 87
  - rubiks.h, 92
- rubiks\_display
  - draw.c, 22
  - draw.h, 34
- rubiks\_neighbour
  - rubiks.c, 87
  - rubiks.h, 93
- rubiks\_side, 12
  - cubie, 14
  - neighbour\_side, 14
  - opposite\_side, 14
  - side, 14
- search\_cubie
  - moves\_rubiks.c, 62
  - moves\_rubiks.h, 79
- set\_colors
  - draw.c, 23
  - draw.h, 35
- show\_menu
  - Menu.c, 44
  - Menu.h, 50
- side
  - rubiks\_side, 14
- SIZE\_MOVE
  - moves\_rubiks.h, 69
- solution\_step
  - solutions\_steps, 15
- solutions\_steps, 15
  - moves\_rubiks.h, 70
  - next\_step, 15
  - solution\_step, 15
- solve\_middle\_row
  - moves\_rubiks.c, 63
  - moves\_rubiks.h, 80
- solve\_rubiks
  - moves\_rubiks.c, 64
  - moves\_rubiks.h, 81
- solve\_white\_side
  - moves\_rubiks.c, 64
  - moves\_rubiks.h, 81
- solve\_yellow\_corner
  - moves\_rubiks.c, 65
  - moves\_rubiks.h, 82
- solve\_yellow\_cross
  - moves\_rubiks.c, 66
  - moves\_rubiks.h, 83
- SQ\_HEIGHT
  - draw.h, 28
- SQ\_WIDTH
  - draw.h, 28
- SQUARES
  - draw.h, 29
- T\_COLOR
  - rubiks.h, 90
- T\_CUBIE\_TYPE
  - rubiks.h, 90
- T\_SIDE
  - rubiks.h, 90
- TERM\_HAS\_NO\_COLORS
  - draw.h, 29
- TERM\_HAS\_NOT\_ENOUGH\_COLORS
  - draw.h, 29
- TERM\_NOT\_BIG\_ENOUGH
  - draw.h, 29
- turn\_three\_corner
  - moves\_rubiks.c, 67
  - moves\_rubiks.h, 84
- type
  - cubies, 10
- UP
  - rubiks.h, 91
- WHITE
  - rubiks.h, 90
- WHT
  - draw.h, 29
- x
  - cubies, 11
- y
  - cubies, 11
- YEL
  - draw.h, 30
- YELLOW
  - rubiks.h, 90