PROJET
"Le Rubik's Cube"

Veuillez lire attentivement le sujet du début jusqu'à la fin avant de commencer.

1 Préambule & Objectif

Le Rubik's Cube a été inventé par Erno Rubik, professeur d'architecture et sculpteur hongrois, en 1974. La commercialisation dans les magasins de jouets de son pays débute en 1977. La popularité croissante de ce jeu en Hongrie permet au Rubik's Cube d'être vendu dans le monde entier en 1980. En l'espace de deux ans, cent millions de Rubik's Cube sont vendus entre 1980 et 1982. Depuis, le Rubik's Cube a gagné le titre de casse-tête parfait, et aujourd'hui plus de deux cent millions d'exemplaires ont été vendus.

Cependant, loin des compétitions organisées autour du Rubik's cube, plusieurs mathématiciens trouvent dans ce petit jouet un problème d'optimisation intéressant vu le nombre exorbitant de configurations possibles si l'on devait lui trouver une solution au hasard. Ainsi, plusieurs algorithmes de résolution ont vu le jour.

Dans ce projet, nous allons découvrir l'un des algorithmes de résolution les plus simple et ce en réalisant les deux étapes suivantes :

1. Implémetation du Rubik's cube et ses différents mouvements ;
2. Résolution étape par étape du Rubik's cube.

Mais avant cela, allons voir quelles sont les caractéristiques d'un Rubik's Cube ...

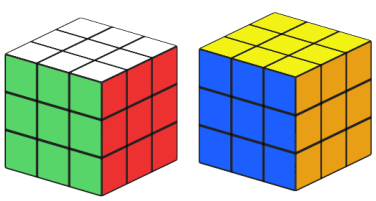
2 Fonctionnement du Rubik's cube

Il existe quelques désignations standards pour le Rubik's Cube qui sont indispensables pour la compréhension des méthodes de résolutions. Ces standards servent notamment à représenter les faces du cube et les mouvements qu'on doit produire sur celles-ci.

2.1 Les faces

Le Rubik's cube est composé de 6 faces portant chacune une couleur différente : Blanc, Orange, Jaune, Bleu, Rouge et Vert.
Ces couleurs sont réparties de sorte que :

— Le Rouge soit face à l'Orange
— Le Bleu soit face au Vert
— Le Blanc soit face au Jaune



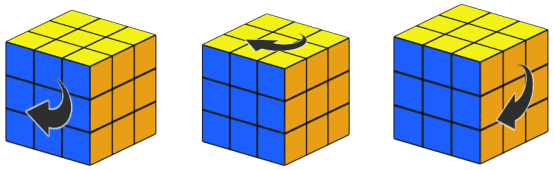
Des noms ont été choisis pour désigner chaque face en fonction de son orientation par rapport au joueur. L'on trouve fréquemment les conventions suivantes :

— la face Bleue est appelée face **UP**, abréviation **U** ;
— la face Blanche est appelée face **FRONT**, abréviation **F** ;
— la face Rouge est appelée face **RIGHT**, abréviation **R** ;
— la face Verte est appelée face **DOWN**, abréviation **D** ;
— la face Jaune est appelée face **BACK**, abréviation **B** ;
— la face Orange est appelée face **LEFT**, abréviation **L**

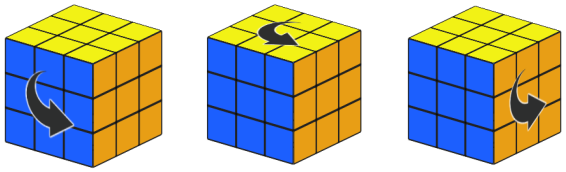
2.2 Les mouvements

Dans notre approche de résolution du Rubik's cube, ce sont les faces entières qui vont bouger. En effet, chacune des 6 faces peut faire deux mouvements :

— Rotation dans le sens des aiguillets d'une montre ;

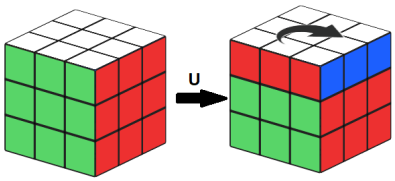


— Rotation dans le sens inverse des aiguillets d'une montre. Dans ce cas, le mouvement sera indiqué par une quote qui suit le nom de la face : U',F', ...

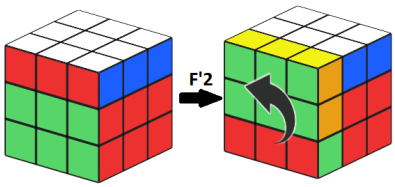


De plus, chaque rotation peut avoir deux type de longueur :

— Quart de tour : faire un seul tour dans le sens des aiguilles d'une montre ou dans le sens inverse.
Exemple : Faire une rotation dans le sens des aiguilles d'une montre en quart de tour de la face **Up** :

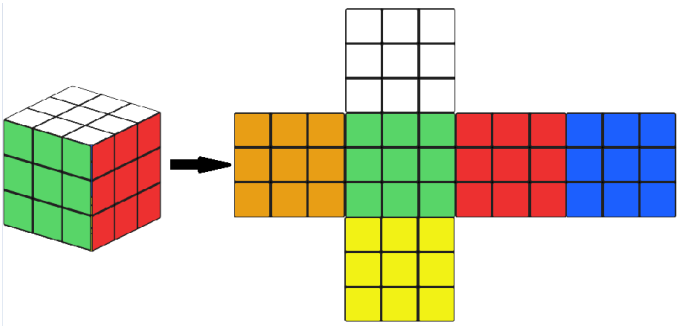


— Demi tour : faire deux tours dans le sens des aiguilles d'une montre ou dans le sens inverse. Demi tour = 2 * Quart de tour
Exemple : Faire une rotation dans le sens inverse des aiguilles d'une montre en demi tour de la face **Front** :



3 Implémentation du Rubik's cube en langage C

Visuellement le Rubik's cube sera représenté en deux dimensions comme montré sur la figure ci-dessous :

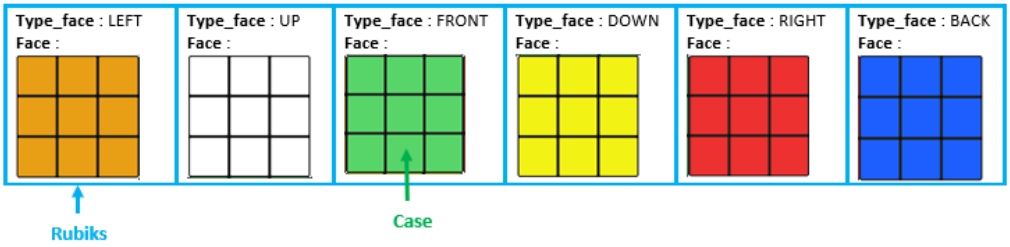


En mémoire, le Rubik's cube sera représenté sous forme d'un tableau à 3 dimensions **"rubiks"** de taille 6 correspondant au nombre de ses faces. Chaque position de ce tableau est un type structuré comportant deux champs :

— Un tableau à 2 dimensions de taille 3 x 3 représentant une face du tableau ;

— Une variable de type énuméré **T_SIDE** désignant le nom de la face.

```
typedef enum { FRONT, BACK, UP, DOWN, RIGHT, LEFT } T_SIDE ;
```



Chaque case d'une face peut être elle aussi de type structuré contenant au moins un champs de type énuméré **T_COLOR**.

```
typedef enum { R, B, G, W, Y, O, LG } T_COLOR ;
```

Pour vous aider à manier votre cube, voici la liste des fonctions que vous devez écrire :

3.1 Fonctions pour les types énumérés

— **select_color** qui prend une couleur en paramètres et qui retourne sa valeur associée.

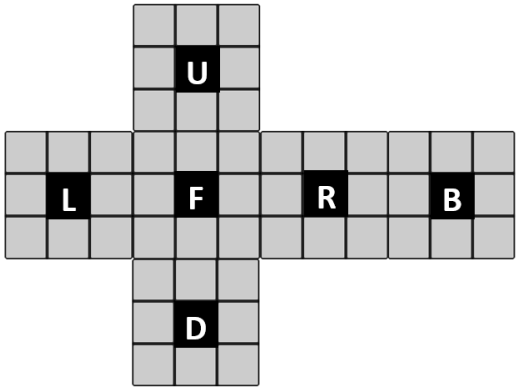
— **side_to_index** qui prend un nom de face en paramètre et qui retourne sa position dans le tableau rubiks.

3.2 Fonctions de représentation du cube

— **create_rubiks** qui permet de créer la structure générale du cube (Tableau à 3D) dynamiquement.

— **init_rubiks** qui prend une variable de type rubiks en paramètre et qui permet d'initiliser chaque face à une couleur unique. Cette fonction doit tenir compte de la régle de distribution des couleurs décrite dans la section 2.1.

— **display_rubiks** qui permet d'afficher à l'écran le Rubik's cube en 2D de sorte que l'ordre des faces soit comme suit quelque soit leur position dans le tableau **rubiks**.



— **blank_rubiks** pour permettre de griser toutes les cases du cube pour une éventuelle initialisation manuelle.

— **fill_rubiks** pour permettre l'affectation manuelle des couleurs aux différentes cases. La fonction doit demander à l'utilisateur de préciser la face, les coordonnées de la case à modifier ainsi que la couleur à affecter. La fonction doit être capable d'empêcher les combinaisons interdites (à déterminer).
Exemple : Deux cases adjacentes ne doivent pas avoir la même couleur.

— **scramble_rubiks** pour permettre de faire des mouvements aléatoires afin de mélanger le cube.

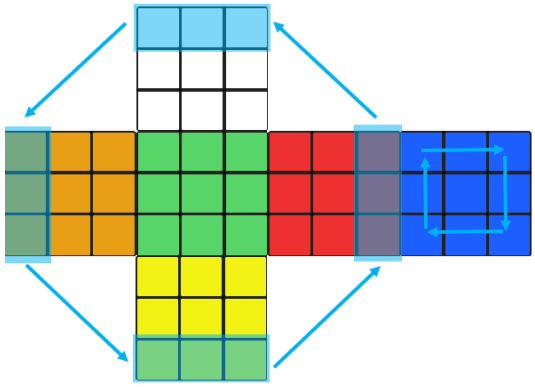
— **free_rubiks** pour permettre de libérer l'espace mémoire à la fin du programme.

3.3 Fonctions de mouvements

— **{side_name}_clockwise** avec side_name ∈ {FRONT, BACK, UP, DOWN, RIGHT, LEFT} est une fonction à deux paramètres : le cube ainsi que le type de rotation. Si type = 1, la fonction doit effectuer une rotation en quart de tour dans le sens des aiguilles d'une montre sur la face side_name. Si type = 2, la rotation doit être d'un demi tour.

Indication :
Une rotation en quart de tour dans le sens des aiguilles d'une montre de la face BACK (back_clockwise) revient à :

1. Faire la rotation de la face elle-même
2. Faire la rotation des lignes d'indice 0 des faces qui sont autour d'elle : UP,RIGHT, LEFT, DOWN (Attention au sens !!!)



— **{side_name}_anticlockwise** avec side_name $\in \{FRONT, BACK, UP, DOWN, RIGHT, LEFT\}$ est une fonction à deux paramètres : le cube ainsi que le type de rotation. Si type = 1, la fonction doit effectuer une rotation en quart de tour dans le sens inverse des aiguilles d'une montre sur la face side_name. Si type = 2, la rotation doit être d'un demi tour.

— **horizontal_rotation** permettant de faire une rotation horizontale où la face BACK devient FRONT (et vice-versa) et la face RIGHT devient LEFT (et vice-versa).

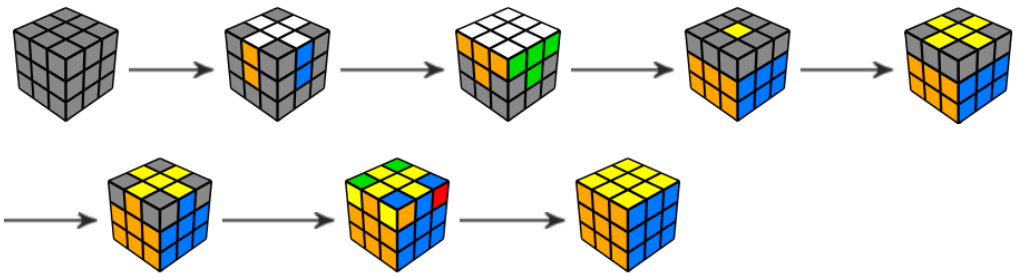
— **vertical_rotation** permettant de faire une rotation verticale où la face UP devient DOWN (et vice-versa) et la face FRONT devient BACK (et vice-versa).

— **move_rubiks** proposant un menu à l'utilisateur pour pouvoir appliquer chacun de ces mouvements manuellement.

4 Algorithme de résolution d'un Rubik's cube

Pour résoudre un Rubik's cube, plusieurs algorithmes sont disponibles. Celui que nous retenons ici est l'algorithme dit "couche par couche" qui est le plus simple et le plus employé par les débutants. Il consiste en les étapes suivantes :

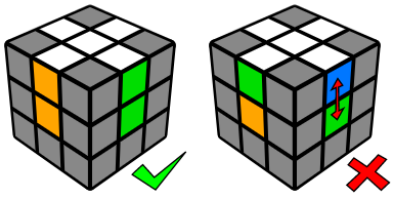
1. Réalisation d'une croix blanche parfaite ;
2. Réalisation de la face blanche entièrement ainsi que la première couronne du cube ;
3. Réalisation de la deuxième couronne ;
4. Réalisation de la dernière face.



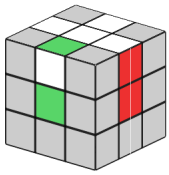
Pour certaines positions des cases, et selon l'étape à laquelle l'on se trouve, des algorithmes existent pour aider à déplacer la case vers sa position finale. Cependant, pour d'autres cas, nous n'avons pas cet avantage. Ainsi, pour chacune des étapes ci-dessous, vous devez programmer les algorithmes donnés en fonction de l'état du cube. Puis, à la fin de chaque étape, **vous devez proposer à l'utilisateur la possibilité d'effectuer des mouvements manuellement pour compléter ou provoquer une configuration prédéfinie**.

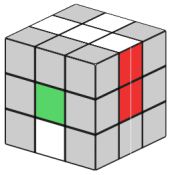
4.1 Réaliser une croix parfaite

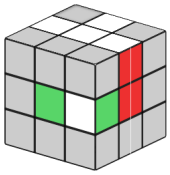
Une croix parfaite est celle où l'on trouve le signe "+" sur la face UP et où les cases arêtes supérieures de chaque face sont de la même couleur que leurs centres.

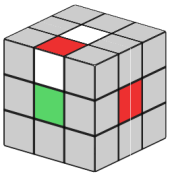


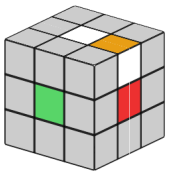
Pour réaliser une croix parfaite, appliquer l'un des algorithmes suivants (en fonction de la position de la case blanche à déplacer) de manière séquentielle si une ou plusieurs de ces situations se présente(ent). Puis, suggérer à l'utilisateur de la compléter manuellement si nécessaire.


F U' R U


F' R' D' R F2'


R' D' R F2'

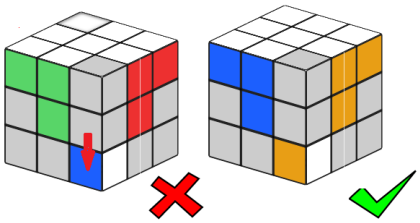

F R


R' F' U

4.2 Réaliser les coins blancs et la première couronne

Cela se fait en deux étapes :

1. Amener le coin sous l'endroit auquel il appartient. Ceci implique de tester les couleurs du coin inférieur droit (ses positions FRONT, RIGHT, DOWN) :
   (a) Si l'une des couleurs du coin est blanche et que les deux autres couleurs correspondent aux faces autour de lui, alors passer à l'étape suivante. Ici l'orientation des couleurs n'est pas importante.
   (b) Sinon, il faudra effectuer un mouvement sur la face DOWN afin de positionner le coin au bon endroit.



2. Répéter l'algorithme ci-dessous jusqu'à ce que le coin blanc se positionne à l'endroit ciblé dans la bonne orientation (ci-dessous quelques exemples de positions où l'on peut appliquer l'algorithme).


R' D' R D

4.3 Réaliser la deuxième couronne

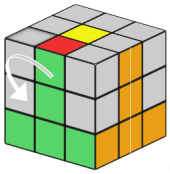
Pour résoudre cette couronne, il faut utiliser, essentiellement, deux algorithmes que l'on appelera ici : **right_move** et **left_move**. Les étapes de résolution sont comme suit :

1. Faire une rotation verticale du cube : la face UP blanche déjà résolue à ce stade doit se retrouver à la position DOWN.
2. Amener la case arête centrale de la face UP à une position de sorte que les deux couleurs qu'elle porte correspondent soit à aux couleurs des faces FRONT-RIGHT ou aux couleurs des faces FRONT-LEFT.
3. Appliquer les algorithmes **right_move** et **left_move** selon l'état du cube comme montré ci-dessous :

RIGHT_MOVE

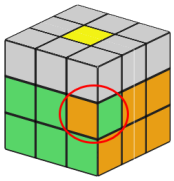
U R U' R' U' F' U F

LEFT_MOVE



U' L' U L U F U' F'

Si une situation de mauvaise orientation se présente, alors il faut appliquer l'algorithme suivant :



U R U' R' U' F' U F U2 U R U' R' U' F' U F

4.4 Réaliser la dernière couronne

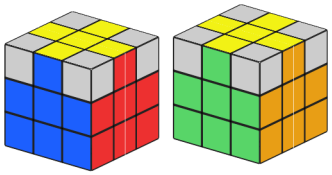
A ce niveau, l'on cherchera à :

1. Réaliser une croix jaune sur la face UP
2. Réaliser la totalité de la face jaune

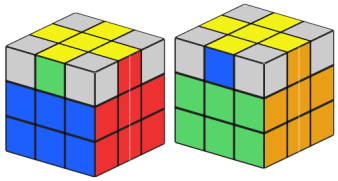
4.4.1 Réaliser la croix jaune

A la fin de l'étape précédente, vous devez être dans l'une des situations ci-dessous. Voici donc les algorithmes à appliquer pour atteindre la croix jaune.

1. **Cas d'une croix jaune**
   Pour valider une croix jaune, elle doit être parfaite comme pour la croix blanche du début du jeu. Une croix jaune parfaite est celle où les 4 arêtes des 4 côtés sont bien placées.
   (a) Si la croix jaune parfaite est réalisée, alors passer directement à la dernière étape de résolution présentée en section 4.4.2.



   (b) Si c'est une croix jaune avec deux arêtes face à face bien placées et deux arêtes mal placées, alors :
   — Désigner comme face FRONT celle qui contient une des arêtes bien placées et la face BACK l'autre face contenant l'autre arête bien placée.
   — Appliquer l'algorithme suivant, puis appliquer l'étape c)

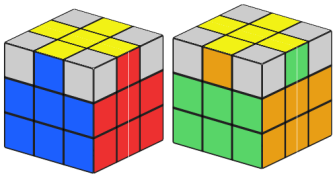


R U2 R' U' R U' R'

(c) Si c'est une croix jaune avec deux arêtes côte à côte bien placées et deux arêtes mal placées, alors :

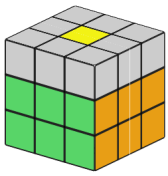
— Désigner comme face FRONT celle qui contient une des arêtes mal placées et la face RIGHT pour l'autre arête mal placée. Ainsi, les deux arête bien placées vont être sur les faces LEFT et BACK.

— Appliquer l'algorithme suivant :



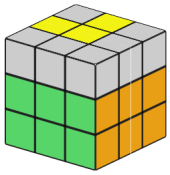
R U2 R' U' R U' R' U'

2. **Cas d'un seul point jaune**
Appliquer l'algorithme suivant pour obtenir une croix puis vérifier si elle est parfaite.



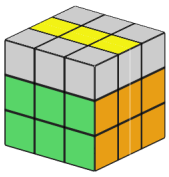
R' U' F' U F R F R U R' U' F'

3. **Cas du L**
Pour pouvoir appliquer les algorithmes de cette section, vous devez identifier une forme L sur la face UP de votre cube suivant la même orientation montrée dans l'image ci-dessous (un L commençant par la face LEFT et orienté vers la face BACK). Une fois la croix jaune obtenue, vérifier si elle est parfaite.



R' U' F' U F R

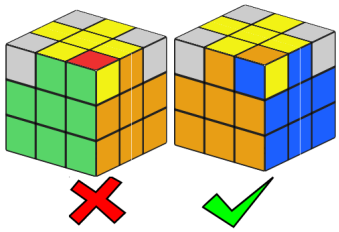
4. **Cas d'une barre jaune**
Pour former une croix jaune à partir d'une barre jaune, la face FRONT de votre cube devrait être celle sui vous permet d'avoir la barre en face de vous à l'horizontale. Selon l'image ci-dessous, les deux faces qui permettent d'avoir la barre en face de nous en horzontale sont les VERTE et BLEUE. Appliquer l'algorithme suivant puis vérifier si la croix jaune obtenue est parfaite.



F R U R' U' F'

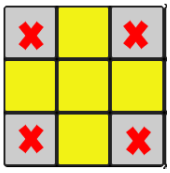
4.4.2 Bien placer les coins

Enfin, il ne reste plus qu'à placer les cases des coins. Pour cela, il faut regarder leurs placements et leurs orientations. Dire qu'une case est bien placée, signifie que la case se trouve à l'intersection des couleurs qui la composent, sinon, elle est dite mal placée.



1. Si les 4 coins sont bien placés alors, il faut donc passer à la section 4.4.3 pour bien les orienter.

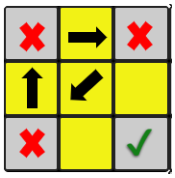
2. Si les 4 coins sont mal placés, alors placer le cube dans n'importe quel sens pourvu que les cases jaunes soient orientées vers la face UP et appliquer l'algorithme ci-dessous avant d'appliquer l'un des deux algorithmes qui suivent.



L' U R U' L U R' U'

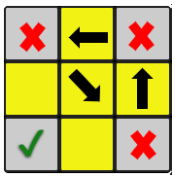
3. Si seulement 3 coins sont mal placés, alors positionner le coin bien placé entre la face UP et la face FRONT et observer quelle situation se présente parmi deux :

Un tour dans le sens des aiguilles d'une montre les remettra en place
Le coin bien placé se trouve entre les faces FRONT-UP-RIGHT



L' U R U' L U R' U'

Un tour dans le sens inverse des aiguilles d'une montre les remettra en place
Le coin bien placé se trouve entre les faces FRONT-UP-LEFT

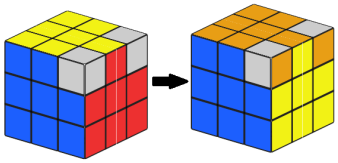


R U' L' U R' U' L U

4.4.3 Bien orienter les coins

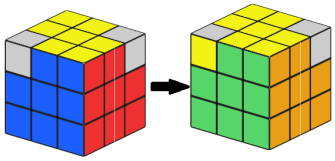
A ce stade, les quatres coins sont bien placés, il ne restera plus qu'à les orienter. Quatre cas se distinguent :

1. Orienter 2 coins côte à côte : Repérer les deux cases semblables et tourner le cube de sorte qu'elles soient orientées vers la face RIGHT, puis appliquer l'algorithme suivant :



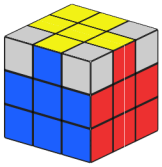
R U2 R' U' R U' R' L' U2 L U L' U L

2. Orienter 2 coins en diagonale : Repérer la case mal placée qui permet d'avoir la face jaune en UP et qui oriente une des cases jaune mal placée vers le FRONT, puis appliquer l'algorithme suivant :

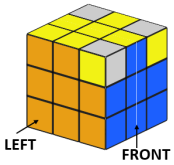


F R U2 R' U' R U' R' L' U2 L U L' U L F'

3. Orienter 3 coins : Dans ce cas, il faudra faire attention à l'orientation des cases jaunes mal placées. L'application des algorithmes de ce niveau nous amène à l'un des deux cas des 2 coins mal orientés. Deux cas se distinguent :

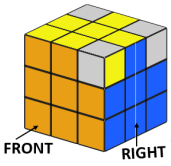


Orienter une case jaune vers la face LEFT
et donc les deux autres vont être
sur les cases FRONT et RIGHT



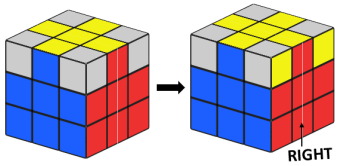
R U2 R' U' R U' R' L' U2 L U L' U L

Orienter une case jaune vers la face RIGHT
et donc les deux autres vont être
sur les cases BACK et LEFT



R U2 R' U' R U' R' L' U2 L U L' U L

4. Orienter 4 coins : repérer deux couleurs identiques (pas nécessairement des jaunes) sur la même face et désigner cette face comme RIGHT, puis appliquer l'algorithme qui suit qui vous amènera au cas où 2 coins côte à côte sont à orienter.



R U2 R' U' R U' R' L' U2 L U L' U L

5 Interface utilisateur

L'interface utilisateur doit proposer à celui-ci, à tout moment, un menu permettant de :

1. Initialiser un Rubik's Cube
2. Griser (vider) un Rubik's Cube
3. Mélanger aléatoirement un Rubik's Cube
4. Attribuer des couleurs manuellement aux cases du Rubik's Cube : Un sous menu doit s'afficher proposant à l'utilisateur de préciser :
   (a) La face à modifier
   (b) Le numéro de la ligne de la case à modifier
   (c) Le numéro de colonne de la case à modifier
   (d) La couleur à lui attribuer
5. Réinitialiser à tout moment le Rubik's Cube
6. Tourner le Rubik's verticalement ou horizontalement
7. Pouvoir jouer manuellement en effectuant des rotations sur les faces
8. Résoudre le Rubik's Cube "couche par couche" :
   (a) Commencer par la réalisation d'une croix blanche parfaite
   (b) Afficher le résultat et proposer à l'utilisateur de passer (ou pas) à l'étape suivante (réalisation de la face blanche).
   (c) Suivre le même processus pour toutes les prochaines étapes en affichant l'état du cube après chacune d'entre elles, jusqu'à sa résolution totale.

Exemple d'affichage :

```
        W W W
        W W W
        W W W

O O O   G G G   R R R   B B B
O O O   G G G   R R R   B B B
O O O   G G G   R R R   B B B

        Y Y Y
        Y Y Y
        Y Y Y

-------------------------------------------------------------------------------------
1: Scramble      2: Reset        3: Blank        4: Play         5: Fill         6: Solve
-------------------------------------------------------------------------------------
Action: 1
        Y W O
        W W Y
        R Y W

W R B   R G O   B R G   W G G
O O O   G G W   B R O   W B B
O O G   B R Y   W R O   R B B

        R G G
        Y Y B
        Y Y Y
```

6 Ressources supplémentaires

— Pour l'affichage des couleurs, utiliser la librairie "conio" (les fichiers "conio.h" et "conio.c" fournis sur moodle) pour Windows, ou la bibliothèque "ncurse" (`https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/ncurses.3x.html`) sous linux ou mac OS.

— Plus de détail sur le fonctionnement d'un Rubik's Cube : `https://rubikscu.be/#cubesolver`https ://-rubikscu.be/#cubesolver

— Vidéo explicative : Résolution d'un Rubik's Cube

7 Consignes générales

1. Le code source doit être composé de trois fichiers :
   — `rubiks.h` : contient les définitions des structures utilisées, des types énumérés, constantes, etc, et les prototypes des fonctions utilisée,
   — `rubiks.c` : inclut `rubiks.h` et contient les définitions des ses fonctions,
   — `main.c` : inclut `rubiks.h` et contient le programme principal.
2. Le projet est à réaliser en binôme (monôme possible). Un seul groupe à trois étudiants pourrait être accepté dans le cas d'une imparité dans le groupe.
3. La remise de votre travail doit inclure :
   — Un fichier .zip contenant l'ensemble des fichiers sources.
   — Un rapport de 10 à 15 pages permettant de présenter :
      — vos solutions (en langage algorithmique) et vos choix de structures de données.
      — les difficultés rencontrées
      — les enseignements de ce projet
      — perspectives d'amélioration de votre rendu
4. Tout dossier remis doit être renommé comme suit : NOM1_NOM2 (avec NOM1 et NOM2 : noms des étudiants qui ont réalisé le projet).
5. Le projet est à rendre sur Moodle.
6. Les travaux sont à rendre au plus tard le 24/05/2021 à 23 :59.
7. La note finale du projet sera composée de :
   (a) La note du code
   (b) La note du rapport
   (c) La note de soutenance.

8 Planning

— Vendredi 16/04/2021 : Lancement du projet (publication sur moodle)

— Semaine du 3/05/2021 : Suivi du projet

— Semaine du 24/05/2021 : Soutenances du projet