Etude Logico-géométrique du Rubik's Cube

Introduction

Le succès du Rubik's Cube est tel que des études mathématiques ont été faites pour étudier ce que l'on appelle son groupe de transformations G (voir par exemple : https://fr.wikipedia.org/wiki/Th%C3%A9orie_math%C3%A9matique_sur_le_Rubik%27s_Cube)

Ce que l'on retient de cette théorie, dans le cadre du projet, c'est que seuls les mouvements 'directs ' U D L R F B (et leurs enchaînements bien sûr) sont nécessaires pour pouvoir résoudre un Rubik's Cube.

Les mouvements inverses U' D' L' R' F' et B' sont eux-mêmes composés de ces mouvements directs : par exemple effectuer F' revient à effectuer 3 fois F (idem pour les autres).

Cependant, ces mouvements inverses étant utilisés de manière classique pour les notations des étapes de résolution, ils seront utilisés dans le projet. Ils sont aussi simples à représenter que les mouvements directs.

Comment faire en pratique ?

La réalisation de ces mouvements au sein d'un programme peut à première vue sembler compliqué. Doit-on, pour chaque mouvement, décrire la manière dont chacune des cases est transformée ?

Sachant qu'un mouvement de face concerne la face elle-même (9 cases) plus les 4 'couronnes' des faces voisines (3 cases par face voisine : 12 cases), cela reviendrait à dire qu'il faudrait écrire 21 transformations individuelles du type cube[1].case[0][1] = cube[2].case[2][0] sans se tromper, pour chaque possibilité (12 types de mouvements), 252 lignes pour ne faire que des affectations…

Puisqu'un cube est une figure très régulière, il doit bien exister un moyen de faire autrement…

Ce document propose une étude logico-géométrique des transformations du Rubik's Cube. Le temps passé à lire et appliquer cette étude (étude de modélisation et de conception algorithmique) est autant de temps gagné pour coder un programme élégant et efficace.

Etude d'un mouvement

Un mouvement de face se décompose en 2 étapes :

1) Mouvement des cases de la face elle-même
2) Mouvement des cases sur une rangée des faces voisines (couronnes)

En réalité, on peut s'épargner un temps précieux, car on peut réutiliser l'étape 1 pour simplifier l'étape 2…

Etape n°1 – Mouvements des cases de la face sur elle-même

Mouvement dans le sens horaire

Le résultat que l'on souhaite est le suivant (tiré d'un programme qui fonctionne)

En utilisant un schéma pour représenter les indices des cases dans un tableau 2D comme des coordonnées, la face de départ est stockée ainsi :

Ligne / Colonne	0	1	2
0	**Y**	**Y**	**W**
1	**W**	**G**	**0**
2	**R**	**B**	**Y**

Si on note les transformations individuelles à faire en indiquant les indices de la case d'origine et les indices de la case de destination, on se rend compte qu'il est très simple de calculer ces derniers à partir des premiers (code couleur pour vous aiguiller)

Indices de la case d'origine	**Indices de la case de destination**
[0][0]	[0][2]
[0][1]	[1][2]
[0][2]	[2][2]
[1][0]	[0][1]
[1][1]	[1][1]
[1][2]	[2][1]
[2][0]	[0][0]
[2][1]	[1][0]
[2][2]	[2][0]

Vous devriez trouver une relation mathématique très simple entre les différents indices

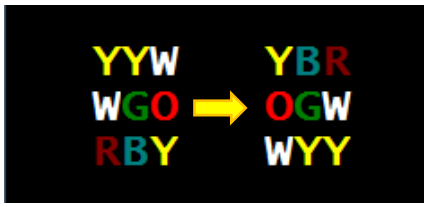
Mouvement dans le sens antihoraire

Le résultat que l'on souhaite est le suivant

Pour trouver les relations entre les indices, il suffit de reprendre le tableau précédent en inversant les deux colonnes – La relation entre les indices est encore une fois assez simple à trouver.

Demi-tour

Le résultat que l'on souhaite est le suivant



Il suffit dans ce cas d'enchaîner 2 mouvements en sens horaire (ou 2 mouvements en sens antihoraire).

Utilisation pour le projet

Vous pouvez ainsi écrire :

- ✓ Une fonction qui effectue un mouvement horaire pour n'importe quelle face ;
- ✓ Une fonction qui effectue un mouvement antihoraire pour n'importe quelle face ;
- ✓ Une fonction qui effectue un demi-tour pour n'importe quelle face.

Ou mieux :

- ✓ Une fonction qui effectue un mouvement horaire ou antihoraire pour n'importe quelle face (le sens du mouvement étant un paramètre de la fonction) ;
- ✓ Une fonction qui effectue un demi-tour pour n'importe quelle face.

Conseil pratique : si vous avez un type `t_face` représentant une face et un type énuméré `t_sens` pour représenter le sens de rotation (`HORAIRE` ou `ANTIHORAIRE`), le prototype de la fonction de rotation de face serait :

```
t_face quart_de_tour(t_face, t_sens);
```

et le code de la fonction pour le demi-tour serait :

```
t_face demi_tour(t_face face)
{
    t_face res;

    res = quart_de_tour(quart_de_tour(face, HORAIRE), HORAIRE);

    return res;
}
```

Etape n°2 – Mouvements des cases des faces voisines

C'est ici que les choses semblent se compliquer, puisque les 'couronnes' des 4 faces voisines doivent être transférées d'une face à une autre, et selon la direction et la face en mouvement, ne concernent pas les mêmes cases…

Et pourtant, on peut simplifier grandement le processus en reprenant le schéma de base du Rubik's Cube. Pour cela, deux méthodes sont proposées, mais non imposées.

Méthode directe

Illustration : quart de tour en sens horaire de la face '**FRONT'**, on note par X les cases concernées

La gestion des mouvements d'une couronne ressemble finalement au mouvement au sein d'une face, si ce n'est que ce mouvement s'effectue d'une face à une autre. Sur le même principe, on peut utiliser un tableau de correspondance indiquant les indices et faces pour matérialiser les mouvements.

Ci-dessous, le tableau de correspondance pour un ¼ de tour de la face **FRONT** dans le sens horaire

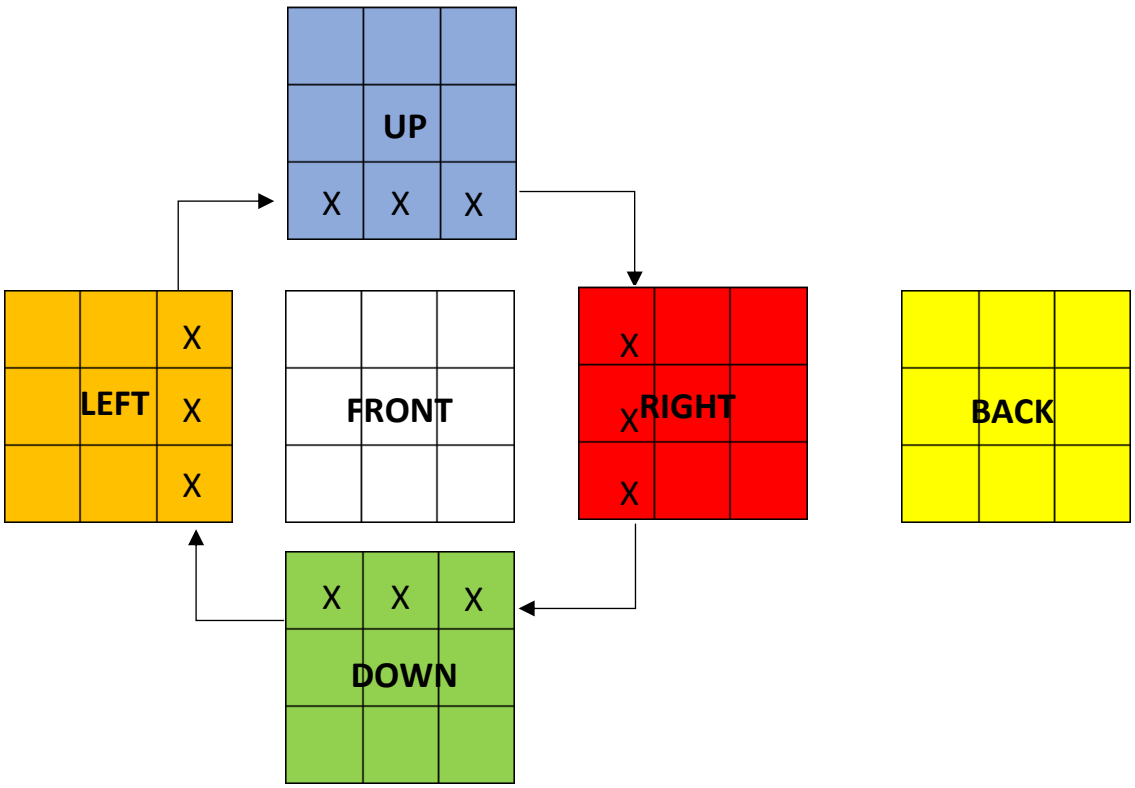
De **LEFT**	Vers **UP**
Indices de départ	Indices d'arrivée
[0][2]	[2][2]
[1][2]	[2][1]
[2][2]	[2][0]

De **UP**	Vers **RIGHT**
Indices de départ	Indices d'arrivée
[2][2]	[2][0]
[2][1]	[1][0]
[2][0]	[0][0]

De **RIGHT**	Vers **DOWN**
Indices de départ	Indices d'arrivée
[2][0]	[0][0]
[1][0]	[0][1]
[0][0]	[0][2]

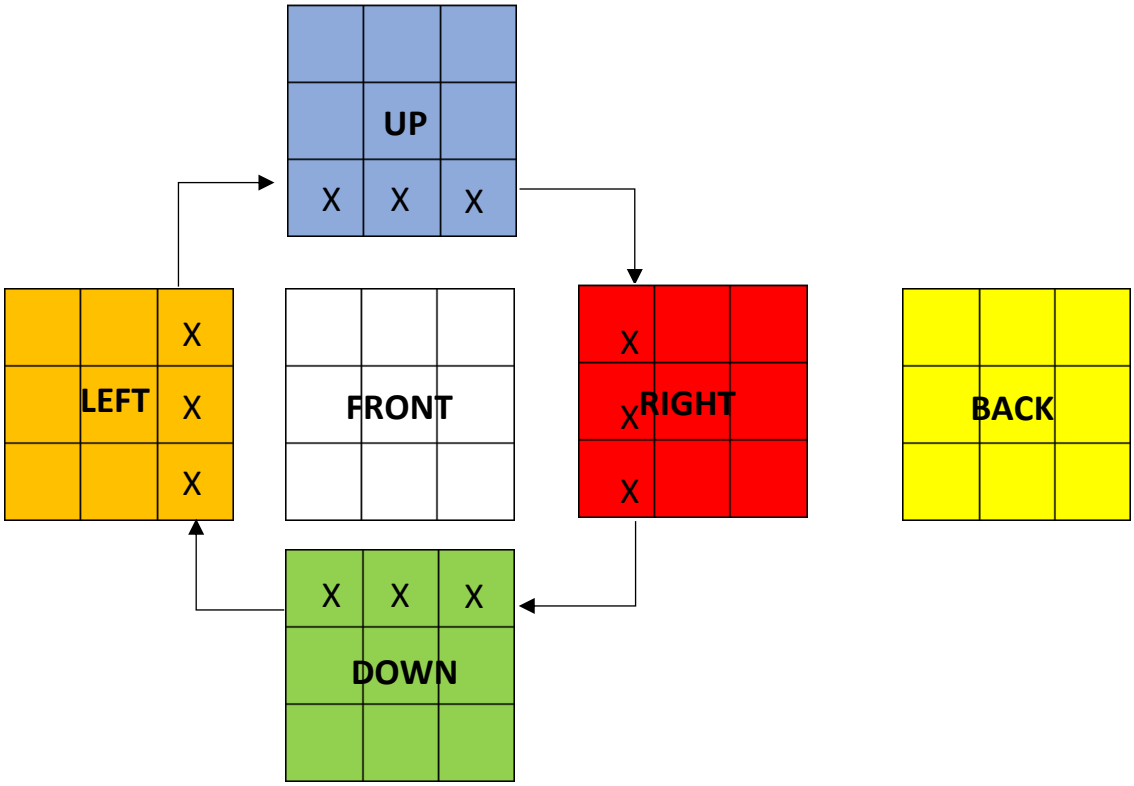
De **DOWN**	Vers **LEFT**
Indices de départ	Indices d'arrivée
[0][0]	[0][2]
[0][1]	[1][2]
[0][2]	[2][2]

Pour réaliser ces transferts, on peut utiliser une seule boucle `for` avec une variable nommée par exemple `cpt`, qui prend les valeurs `0`, `1`, `2`.

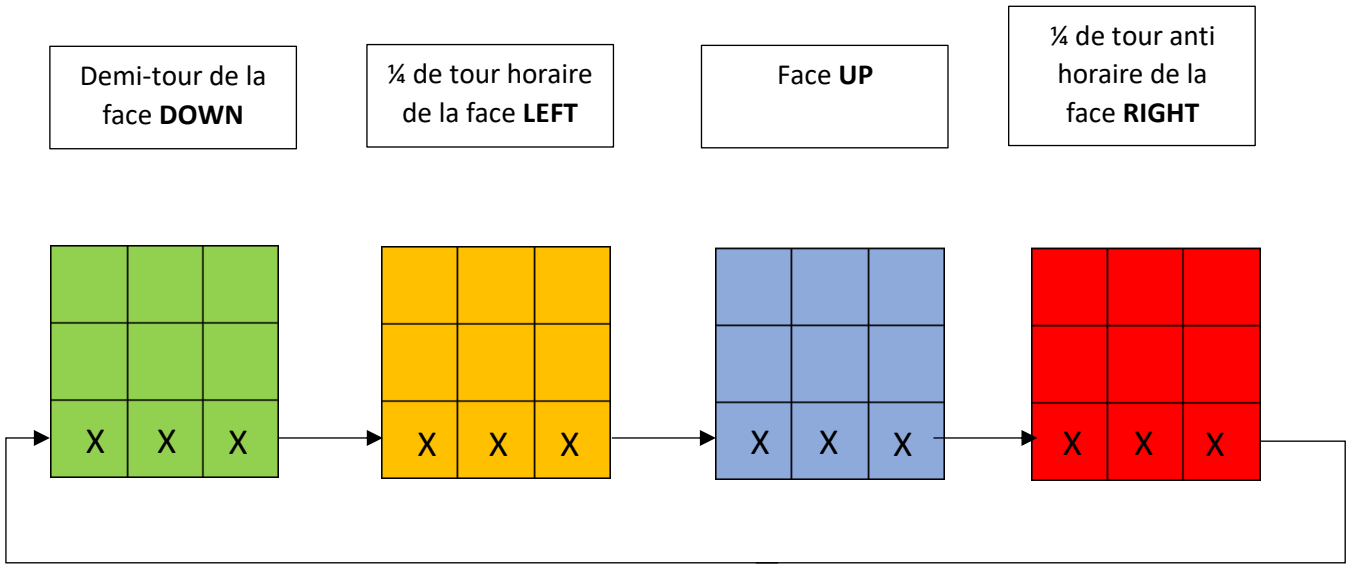


Méthode géométrique, ¼ de tours et échange de lignes

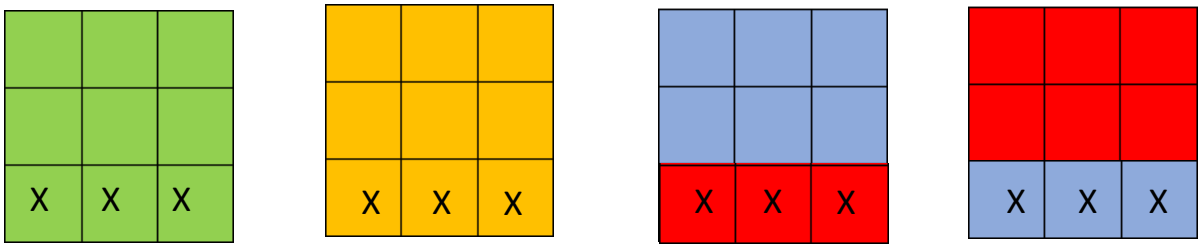
Repartons du schéma de mouvement des ‘couronnes’



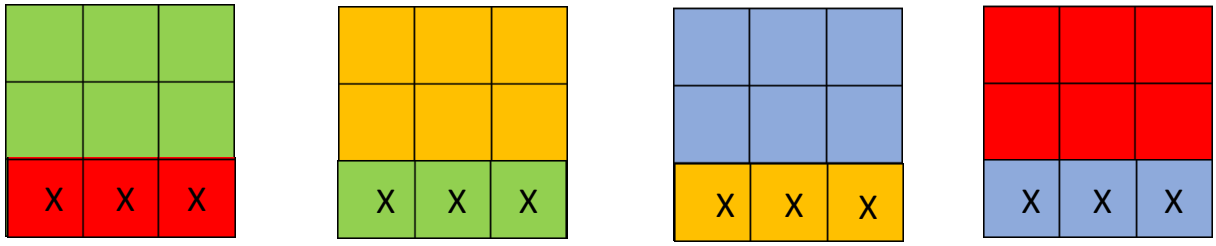
En réutilisant les quarts de tour et demi-tour de faces, on peut ‘déplier’ ce schéma de rotation des couronnes ainsi, en choisissant (par exemple) de matérialiser les mouvements de la couronne dans la ligne du bas des 4 faces concernées. Le plus important est que toutes les cases à déplacer soient au même endroit dans les 4 faces.



On procède ensuite par ‘échange’ des dernières lignes : **UP** / **RIGHT**



La face **RIGHT** (tournée) a sa bonne couronne, on continue à échanger : face **UP** et face **LEFT**, puis la face **LEFT** et la phase **DOWN**

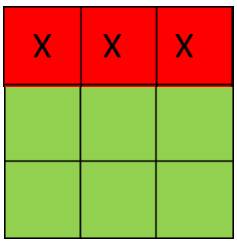


Pour une rotation en sens anti-horaire, il suffit de faire les échanges dans l’autre sens :

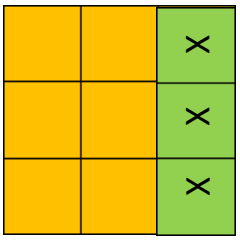
UP / **RIGHT** puis **RIGHT**/**DOWN** puis **DOWN** / **LEFT**

Dernière opération : appliquer à ces faces les transformations ‘inverses’ de celles dont on s’est servi pour pouvoir échanger les lignes.

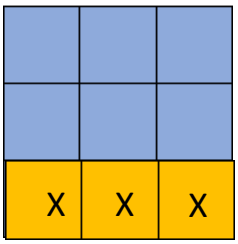
La face **DOWN** (verte) avait subi un demi-tour : elle subit de nouveau un demi-tour



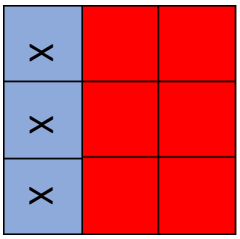
La face **LEFT** avait subi un ¼ de tour en sens horaire, elle subit un ¼ de tour en sens antihoraire



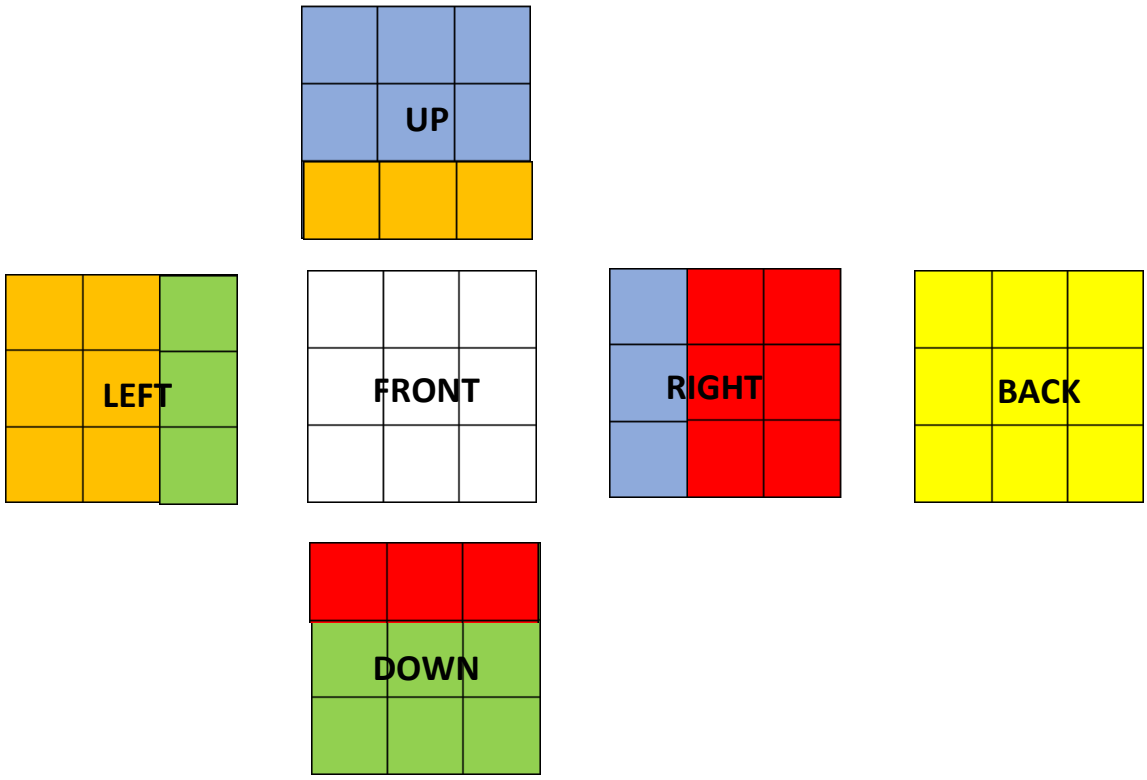
La face **UP** n'avait pas subi de rotation, elle reste la même :



La face **RIGHT** avait subi un ¼ de tour en sens antihoraire, elle subit un ¼ de tour en sens horaire



En remettant ces faces à leur place sur le schéma, on obtient donc le résultat escompté.



Cet exemple est illustré avec des faces unies, mais fonctionne parfaitement bien indépendamment des couleurs des cases sur les faces.

Conseil pratique : si vous avez un type **t_face** représentant une face et un type énuméré **t_sens** pour représenter le sens de rotation (***HORAIRE*** ou ***ANTIHORAIRE***), il est conseillé de découper le traitement en fonctions, dont voici les prototypes :

Si vous voulez utiliser la méthode géométrique, un échange de lignes est nécessaire :

```
void echange_lignes(t_face *, t_face *); // echange des lignes de deux faces
```

sinon pour les 2 méthodes présentées :

```
void couronnes(t_face *, t_face *, t_face *, t_face *, t_sens);
// gère les déplacements des rangées dans 4 faces, avec la convention suivante
// le premier paramètre est l'@ de la voisine de GAUCHE de la face que l'on fait tourner
// le premier paramètre est l'@ de la voisine du DESSUS de la face que l'on fait tourner
// le premier paramètre est l'@ de la voisine de DROITE de la face que l'on fait tourner
// le premier paramètre est l'@ de la voisine du DESSOUS de la face que l'on fait tourner
// le dernier paramètre est le sens de rotation
```

Dans cette fonction ***couronnes***

- ✓ Ecrire le programme correspondant à l'une des méthodes proposées si vous le souhaitez.

Le principe de la méthode des couronnes peut être généralisé à toutes les faces, comme indiqué dans la suite de ce document. Le principe est le suivant : dans l'exemple utilisé pour la face FRONT, les mouvements sont matérialisés pour les faces voisines de cette même face FRONT. Chaque face du Rubik's Cube est en contact avec 4 autres, et il est possible de généraliser l'exemple précédent en trouvant, pour chaque face, quelles sont ses voisines.

Quid des autres faces ?

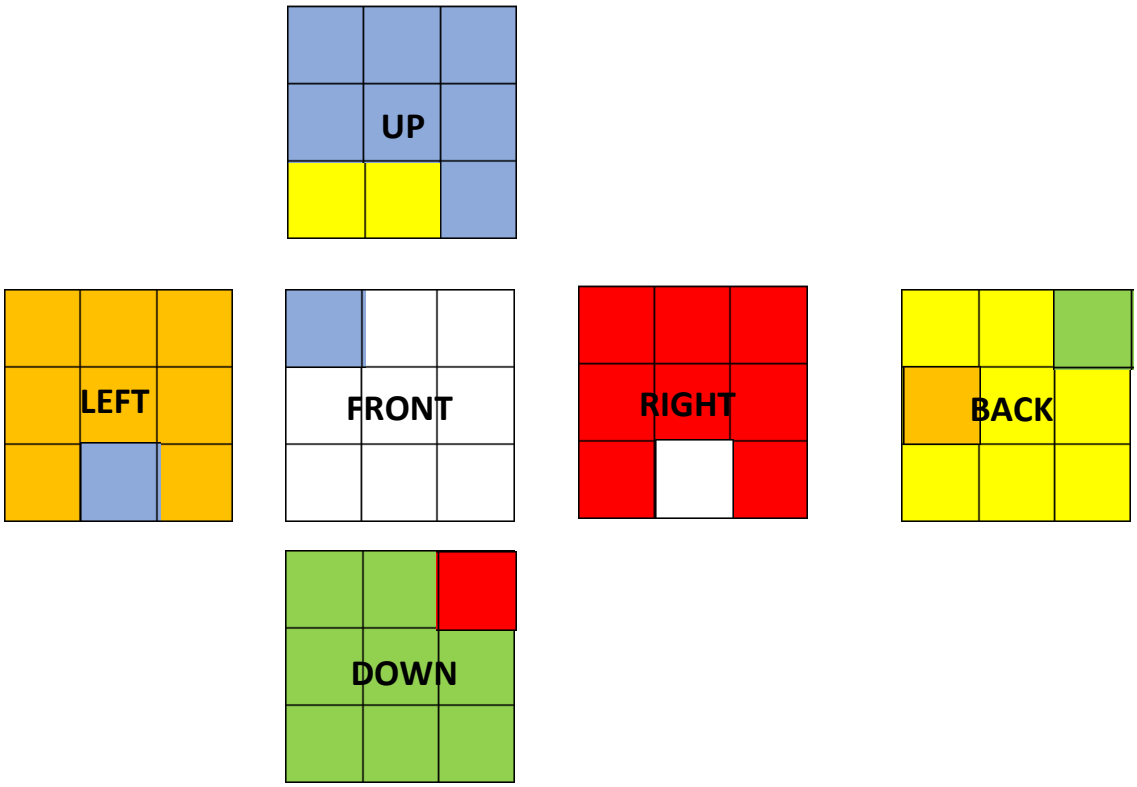
Cela fonctionne parfaitement pour la face **FRONT** de l'exemple, mais pas encore avec les autres faces. L'avantage de la face **FRONT** est qu'elle dispose, sur le schéma, de 4 voisines, ce qui permet d'illustrer le point précédent facilement.

Cependant, toujours en utilisant les ¼ de tours et ½ tours, on peut systématiquement se ramener au cas précédent, où on peut trouver pour chaque face :

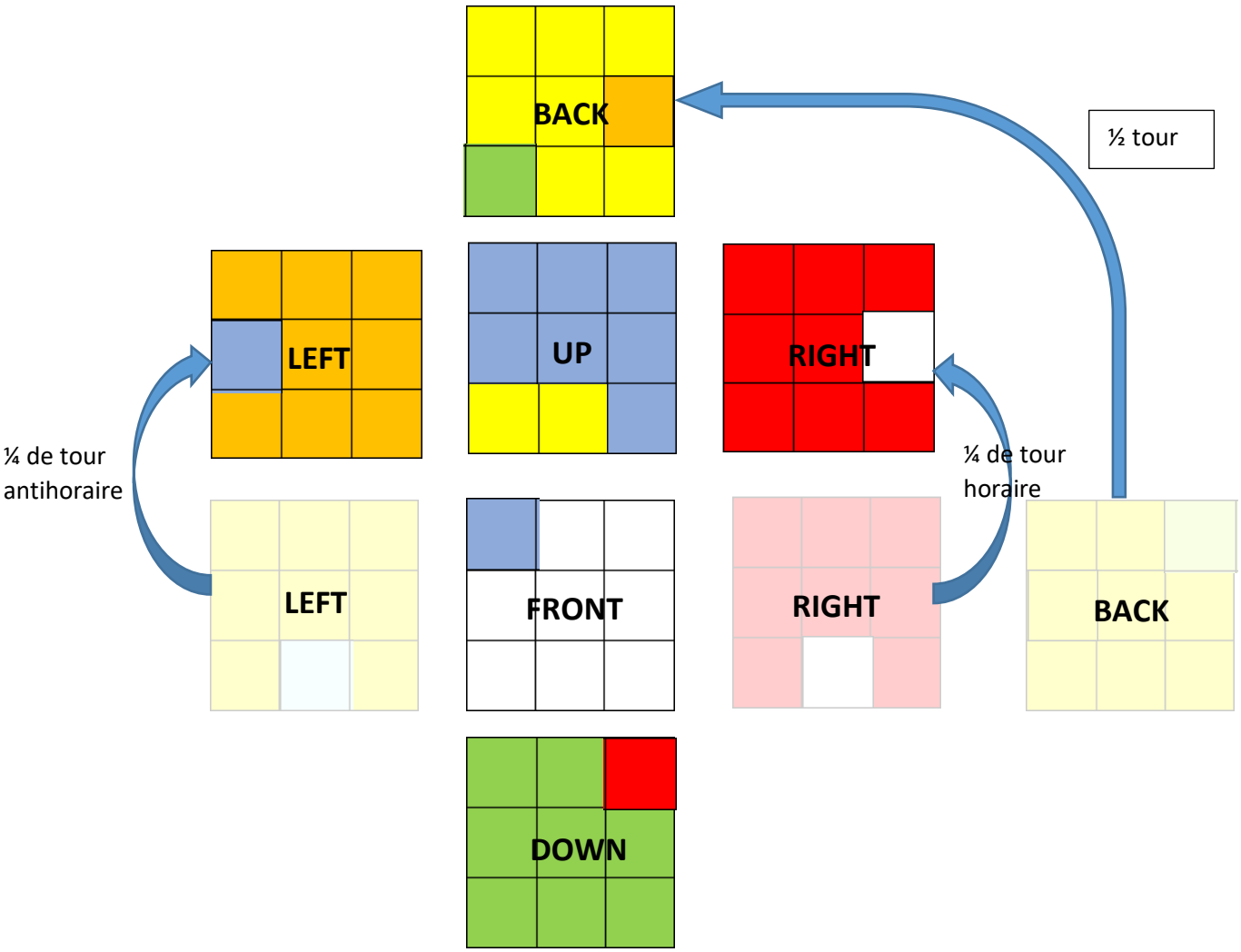
Sa voisine de gauche, sa voisine du dessus, sa voisine de droite et sa voisine du dessous.

Ainsi, la méthode précédente fonctionne pour toutes les faces, si on fait attention à bien repérer les 4 voisines d'une face.

Exemple : soit le Rubik's cube suivant (le mélange a été fait à la main pour l'illustration):



Supposons que l'on souhaite faire tourner la face **UP** (bleue avec 2 cases jaunes) d'1/4 de tour (peu importe le sens). On cherche à appliquer la méthode précédente, il faut donc trouver quelles sont les 4 voisines de la face **UP**. Pour cela, on redessine le schéma en faisant tourner les faces de manière à ce que la face **UP** ait 4 voisines (voir page suivante)



Ainsi :

- ✓ La voisine de gauche de la face **UP** est la face **LEFT**, tournée d'1/4 de tour dans le sens antihoraire ;
- ✓ La voisine de droite de la face **UP** est la face **RIGHT**, tournée d'1/4 de tour dans le sens horaire ;
- ✓ La voisine du dessous de la face **UP** est la face **FRONT** ;
- ✓ La voisine du dessus de la face **UP** est la face **BACK**, tournée d'un ½ tour.

En résumé, pour faire tourner le cube autour de la face **UP** : faire tourner la face **UP** seule, puis appliquer la méthode des couronnes comme précédemment, avec les voisines déterminées ci-dessus.

Enfin, comme pour la méthode géométrique des couronnes, une fois les opérations précédentes effectués, il suffit de remettre les faces voisines dans leur état d'origine.

Sur cet exemple, une fois la rotation de la face **UP** et la rotation des couronnes faites, on doit appliquer :

- ✓ La face **LEFT** est tournée d'¼ de tour dans le sens horaire ;
- ✓ La face **RIGHT** est tournée d'¼ de tour dans le sens antihoraire ;
- ✓ La face **FRONT** n'a pas été tournée, donc elle reste à l'identique ;
- ✓ La face **BACK** est tournée d'½ tour de nouveau pour revenir à sa position d'origine.