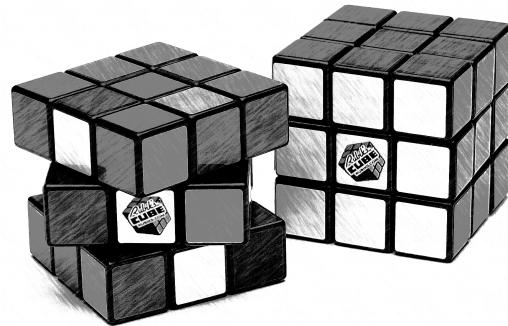


Lycée Carnot  
Classe préparatoire MP

---

## Rubik's Kube



Travail d'Initiative Personnelle Encadré de :  
**Guillaume BOULANGER et Axel ARDISSON**

Professeurs encadrants:  
**Monsieur LAMARD**  
**Monsieur LACOUTURE**

2013-2014



## Avant propos

Ce projet a été conjointement réalisé par Guillaume Boulanger et Axel Ardisson, élèves en classe préparatoire au lycée Carnot de Dijon.

Le Rubik's Cube, inventé par Ernö Rubik, est aujourd'hui considéré comme le casse-tête logique par excellence. Cette notoriété en fait un très bon objet d'étude dans l'optique de créer une solution informatique à des problèmes mathématiques complexes. En effet les méthodes de résolution du Rubik's Cube (que l'on désignera simplement par "Cube" dans la suite) sont algorithmiques, d'où l'idée de créer un programme résolvant le Cube automatiquement.

Cette solution informatique se base sur l'algorithme de résolution dite "layer by layer" (couche par couche). Ce choix se justifie par le fait que cette méthode est l'une des plus simple et intuitive, généralement pratiquée par les personnes débutant sur le Cube. La conception de cette solution informatique met en relief le fait que la machine est dépourvue de capacité de raisonnement, propre à l'homme. Cet algorithme de résolution sera présenté en intégralité dans ce document.

Le programme de résolution sera rédigé sous le langage Caml Light, enseigné en classe de CPGE scientifique. A noter que ce langage nécessite une grande rigueur, ce qui force à beaucoup de discipline lors de l'apprentissage de l'algorithmique. Cette étude portera aussi sur la portée des transferts de données relatifs au traitement informatique d'un algorithme, incluant des problèmes de flux de données dans un vecteur Caml (tableau de donnés), de mutabilité et autres .

Cette idée de projet découle directement d'une fascination de longue date pour les casses-têtes, démystifiée pour de bon en allant au fond des choses. En effet la réalisation d'un système résolvant le Cube de manière automatique nécessite une parfaite connaissance des "mécanismes" du Cube.

*À ceux qui n'ont jamais réussi à résoudre le "Cube"  
et ceux qui veulent réussir sans apprendre.*



## Remerciements

Il est de mise de remercier certaines personnes sans qui le projet n'aurait pas abouti.

Tout d'abord Monsieur Lamard pour son enseignement exigeant mais juste de l'informatique (et de la rigueur).

Remercions aussi Monsieur Lacouture, son enthousiasme communicatif, et son intérêt pour notre projet.

Des remerciements s'imposent pour tous nos autres professeurs de classes préparatoires, qui ont contribué à notre évolution ces deux dernières années.

Un immense merci au Comptoir Oriental et à Shen Nong : l'un pour nous avoir fourni la quantité astronomique de thé consommée pendant ce projet, l'autre pour avoir découvert son utilisation.



# Table des matières

Avant propos . . . . .	i
Remerciements . . . . .	iii
<b>Table des figures</b>	<b>vii</b>
<b>Liste des tableaux</b>	<b>ix</b>
<b>1 Prémices à un traitement algorithmique</b>	<b>1</b>
1.1 Position du problème . . . . .	1
1.2 Nécessité d'un vocabulaire inhérent au traitement du problème.	2
1.3 Méthode de représentation et espace associé . . . . .	3
<b>2 Démarche</b>	<b>5</b>
2.1 Pourquoi le Rubik's Cube? . . . . .	5
2.2 Comment créer un "Rubik's Cube informatique"? . . . . .	5
2.3 Comment en modéliser les mouvements? . . . . .	6
2.4 Quelle méthode de résolution? . . . . .	7
<b>3 Algorithme</b>	<b>9</b>
3.1 Algorithmes de mouvement . . . . .	9
3.1.1 Préambule à la création de fonctions de mouvement . .	9
3.1.2 Exemple développé de la rotation de la face blanche .	12
3.2 Étape 1 . . . . .	19
3.2.1 Description du but de l'étape . . . . .	19
3.2.2 Description des macros de l'étape 1 . . . . .	20
3.2.3 Le programme Étape 1 . . . . .	23
3.3 Étape 2 . . . . .	26
3.3.1 Description du but de l'étape . . . . .	26
3.3.2 Description des macros de l'étape 2 . . . . .	27
3.3.3 Le programme Étape 2 . . . . .	32
3.4 Étape 3 . . . . .	33
3.4.1 Description du but de l'étape . . . . .	33
3.4.2 Description des macros de l'étape 3 . . . . .	34
3.4.3 Le programme Étape 3 . . . . .	38
3.5 Étape 4 . . . . .	40
3.5.1 Description du but de l'étape . . . . .	40
3.5.2 Description des macros de l'étape 4 . . . . .	41
3.5.3 Le programme Étape 4 . . . . .	43

3.6 Étape 5 . . . . .	46
3.6.1 Description du but de l'étape . . . . .	46
3.6.2 Description des macros de l'étape 5 . . . . .	47
3.6.3 Le programme Étape 5 . . . . .	49
<b>4 Conclusion</b>	<b>51</b>
4.1 Idées d'améliorations . . . . .	51
4.1.1 Le filtre . . . . .	51
4.1.2 Traitement exhaustif des mouvements . . . . .	51
4.2 Traitement expérimental . . . . .	52
4.2.1 Fonction de mélange . . . . .	52
4.2.2 Résultats expérimentaux . . . . .	52
4.3 Gains personnels . . . . .	53
<b>A Ernö Rubik</b>	<b>55</b>
A.1 Biographie . . . . .	55
A.2 Une interview intéressante . . . . .	55
<b>B Cubestormer</b>	<b>59</b>
B.1 Notre modèle et motivation : le projet Cubestormer . . . . .	59
<b>C Les Méthodes de résolutions</b>	<b>63</b>
C.1 La méthode “Layer by Layer” : . . . . .	63
C.2 La méthode Sandwich . . . . .	63
C.3 Méthode de Lars Petrus . . . . .	64
C.4 Méthode de Jessica Fridrich (ou CFOP) . . . . .	64
C.5 Méthodes corners first (Guimond, Ortega, Waterman) . . . . .	65
C.6 Méthode Roux . . . . .	65
C.7 Les Cubes $4 \times 4 \times 4$ , $5 \times 5 \times 5$ , etc . . . . .	65
<b>D Le Rubik’s Cube en compétition</b>	<b>67</b>
D.1 Les règles du Speedcubing . . . . .	67
D.2 Compétition . . . . .	68
<b>E Notre programme entier et non commenté</b>	<b>69</b>
E.1 Les mouvements . . . . .	69
E.2 Les booléens . . . . .	75
E.3 Les macros . . . . .	76
E.4 Les étapes . . . . .	79
E.4.1 Étape 1 . . . . .	79
E.4.2 Étape 2 . . . . .	82
E.4.3 Étape 3 . . . . .	83
E.4.4 Étape 4 . . . . .	84
E.4.5 Étape 5 . . . . .	86
E.5 Exemple de fichier final . . . . .	86

# Table des figures

2.1	Représentation sous forme de patron . . . . .	6
3.1	Kube repéré . . . . .	9
3.2	Représentation sous forme de patron . . . . .	11
3.3	Sous-fonction : Arrêtes faces . . . . .	12
3.4	Sous-fonction : Coins faces . . . . .	13
3.5	Sous-fonction : Arrêtes externes . . . . .	14
3.6	Sous-fonction : Coins externes 1 . . . . .	15
3.7	Sous-fonction : Coins externes 2 . . . . .	16
3.8	Fonction général de rotation de la face blanche . . . . .	18
3.9	Représentation du patron du cube à la fin de l'étape 1 . . . . .	19
3.10	Description de la macro d'orientation . . . . .	22
3.11	Représentation du patron du cube à la fin de l'étape 2 . . . . .	26
3.12	Description de la macro de position des coins . . . . .	28
3.13	Description de la macro de position des coins . . . . .	29
3.14	Description du principe de conservation en 6 temps . . . . .	31
3.15	Représentation du patron du cube à la fin de l'étape 3 . . . . .	33
3.16	Explicitation de la dénomination a, b : ici a = rouge et b = vert .	35
3.17	Explicitation de la dénomination droite et gauche : ici droite = vert et gauche = bleu . . . . .	35
3.18	Description de la macro dite du manchot . . . . .	37
3.19	Représentation du patron du cube à la fin de l'étape 4 . . . . .	40
3.20	Description de la macro de résolution de la croix jaune . . . . .	41
3.21	Description de la macro "chaise" . . . . .	43
3.22	Description des fonctions de booléens . . . . .	44
3.23	Représentation du patron du cube à la fin de l'étape 5 . . . . .	46
3.24	Description de la macro de rotation des coins jaunes 1 . . . . .	47
3.25	Description de la macro de rotation des coins jaunes 2 . . . . .	48
B.1	David Gilday et Mike Dobson à la remise du record du monde du Cubestormer 3 . . . . .	59
B.2	Le Cubestormer 1 . . . . .	60
B.3	Le Cubestormer 2 . . . . .	61
B.4	Le Cubestormer 3 . . . . .	61



# Liste des tableaux

2.1	Vecteur représentant le Kube résolut en Caml . . . . .	6
3.1	Vecteur représentant le Kube résolut en Caml . . . . .	11
3.2	Vecteur représentant le Kube en fin d'étape 1 . . . . .	19
3.3	Vecteur représentant le Kube en fin d'étape 2 . . . . .	26
3.4	Vecteur représentant le Kube en fin d'étape 3 . . . . .	33
3.5	Vecteur représentant le Kube en fin d'étape 4 . . . . .	40
3.6	Vecteur représentant le Kube en fin d'étape 5 . . . . .	46



## 1.1 Position du problème

Au préalable, précisons que l'objectif de ce projet n'est pas de démontrer qu'il est possible de résoudre le Rubik's Kube informatiquement. Ce casse tête est par nature un exercice classique d'algorithmique qui a de nombreuses solutions avec des approches assez diverses. Certains ont même conçus des robots capable de résoudre physiquement des Kube et ce dans des temps très courts. Saluons ici la performance de Mike DODSON (ingénieur robotique) et David GILDAY (programmeur) dont le CUBESTORMER 3 a décroché un "World's Guinness des Records" grâce à une résolution classique de cube en 3.253 s.

Nous voyons donc que la question de la possibilité de traitement informatique du cube est déjà résolue, aussi ne la traiterons nous pas et partiront du principe que ce traitement est possible (ce que l'expérience a prouvé).

Ce projet s'inscrit en réalité dans la modélisation de l'homme par la machine. En réalité l'objectif que nous nous sommes fixé est de créer un programme qui imiterait le comportement d'un individu résolvant utilisant la méthode de résolution dite "Layer By Layer" (ndlr : couche par couche ).

Ce choix s'inscrit dans une idée qu'une machine (ici un ordinateur) bien que dénué de capacité de raisonnement propre est en mesure de résoudre des problèmes de natures complexes pour peu que l'on lui "apprenne" comment faire. La méthode de résolution employé ici est l'une des plus simples qui soit, souvent enseignées aux néophytes du cube désireux de débuter et est historiquement la première puisque c'est celle qu'Eno Rubik avait mise au point lorsqu'il s'intéressa à la résolution du casse tête qu'il avait inventé sans le vouloir.

Le traitement informatique est effectué dans le langage CamLight, et cette expérience nous aura notamment permis d'étudier plus en profondeur des problèmes de transferts d'informations à travers les différentes structures de données que nous avons étudiée au cours de ces 2 dernières années.

Ce projet nous aura permis d'avoir une vision plus large de la démarche scientifique. Nous avons en effet appris l'importance du travail d'équipe et la nécessité de la confiance entre nous et en nous. D'autre part nous avons mis à profit toutes les capacités de raisonnement que l'enseignement en CPGE nous a fourni, il nous a fallu apprendre à manier le rubik's cube, connaître le langage informatiquement et les bases de raisonnement logique et surtout, persévéérer et ne pas se décourager lorsque quelquechose ne fonctionnait pas ( ce qui est arrivé beaucoup plus souvent que nous n'oserions l'admettre).

Cette expérience nous a permis :

- d'approfondir les problèmes de transferts d'informations à travers les différentes structures de données étudiées au cours de 2 années de prépa.
- de nous perfectionner au language CamLight.

### 1.2 Nécessité d'un vocabulaire inhérent au traitement du problème.

L'une des premières choses qui nous a frappés lors des balbutiements de ce projet a été la difficulté à nous comprendre. En effet lors de la rédaction d'un problème Mathématique ou Physique Nous disposons d'un ensemble de notations et de définitions permettant de rendre compte d'un raisonnement scientifique sans aucune ambiguïté. Toutefois lorsque l'on en venait au traitement du Kube les quiproquo se multipliaient, que ce soit sur ce que nous appelions un mouvement ou même la conception que nous avions d'une "pièce" du Kube. Ceci a mis en avant la nécessité de définir un Vocabulaire précis inhérent à la résolution de ce casse-tête.

Un algorithme devant renvoyer un résultat sans ambiguïté, voici donc quelques points de définitions permettant d'envisager le raisonnement qui sera développé sans ambiguïté aucune.

**Vignette :** Une vignette est une zone géographique de la surface externe du Kube. Le Kube en comporte 54. C'est une zone immobile dans un référentiel par rapport au temps. Une vignette repère une zone spatiale propre au cube.

**Valeur :** Une valeur caractérise une vignette à un instant  $t$  donné. C'est une notion variable dans le temps, lorsque l'organisation du cube est modifiée, les valeurs des vignettes qui le composent sont modifiée (éventuellement pour elle-même). Les valeurs appartiennent à l'espace W,R,B,G,O,Y.

**Point fixe :** Vignette du Kube de valeur fixe dans le temps. Un Kube en comporte 6, une de chaque valeur.

**Face :** Une face est une zone géographique du cube composé de 8 vignettes disposée en carré autour d'un point fixe. On donnera comme dénomination à une face la valeur du point fixe en son centre, ce qui la caractérise totalement.

**Piece :** Une piece est une partie du Kube que l'on considère fixe dans le référentiel du Kube. Elle désigne un groupement de vignettes.

- Si elle compte 1 vignette, il s'agit d'un point fixe.
- Si elle compte 2 vignettes, il s'agit d'une arête.
- Si elle compte 3 vignettes, il s'agit d'un coin.

**Couronne :** Une couronne est un ensemble de 4 arêtes, 4 coins et 1 point fixe disposés en carré autour de ce dernier. Les vignettes d'une couronne compte 1 face complète et la couronne prendra comme désignation la valeur de cette face (ou bien encore la valeur du point fixe lui appartenant ce qui est strictement équivalent)

**Mouvement :** Un mouvement est une action physique transformant un Kube en Cube. Il s'agit d'une rotation d'une couronne autour de l'axe perpendiculaire à son point fixe en son centre et d'un angle  $\theta = 0(\frac{\pi}{2})$ .

**Kube :** Bien que cette désignation ait déjà été utilisée il convient de la définir de manière correcte. Dans ce document le terme "Kube" désigne un "Rubik's Cube 3x3 Classique". L'ensemble des vignettes d'un Kube contient 9 éléments de chaque valeur. Le Kube est supposé dans un état dis "soluble" à tout instant où il existe une combinaison de mouvement permettant de ramener le Kube dans son état résolu. Un Kube est dis "résolu" lorsque chacune de ses faces est unicolore.

### 1.3 Méthode de représentation et espace associé

Le Kube étant un objet complexe, il n'existe pas de représentation directe et simple. Nous avons donc choisi de le représenter par un vecteur Caml de dimension 54 à valeurs dans  $\{W, R, B, G, O, Y\}$ . Notons déjà que l'espace du Kube K est un sous espace de  $E = \{W, R, B, G, O, Y\}^{54}$ .

$$\boxed{\text{card}(E) = 6^{54} = 1,0.10^{42}}^1$$

Il est soumis à certaines restrictions :

- Le Kube doit forcément avoir 9 vignettes de chaque valeurs.
- Il possède 6 points fixes pré-déterminés.
- L'espace des Kube solubles est encore réduit, nous le noterons K, son cardinal se calcule de la façon suivante<sup>2</sup> :
  1. Il y a deux orientations possibles pour chaque arête. Étant donné qu'on ne peut pas changer l'orientation d'une arête seule, l'orientation de toutes les arêtes fixe l'orientation de la dernière. Cela donne 211 possibilités d'orientation des arêtes.

---

1. Ceci montre la place importante du chiffre 42 dans le questionnement sur l'univers.

2. Source : Wikipedia.

2. Il y a trois orientations possibles pour chaque coin. De même, on ne peut pas retourner un coin seul, l'orientation du dernier coin est donc fixée par les autres. Cela donne 37 possibilités d'orientation de coins.
3. Les arêtes peuvent s'interchanger entre elles, ce qui donne 12! possibilités de positionnements pour les arêtes.
4. Les coins peuvent s'interchanger entre eux. Cela fait 8! possibilités.
5. Mais il existe un problème dit de parité : on ne peut échanger juste deux coins ou deux arêtes (mais on peut interchanger deux coins et deux arêtes). La position des arêtes et des premiers coins fixe donc la position des deux derniers coins et il faut donc diviser le résultat par deux.

$$\text{card}(K) = 8! \times 3^7 \times 12! \times 2^{10} = 43\ 252\ 003\ 274\ 489\ 856\ 000$$

On remarque donc que si un être humain passe en revu chacune des combinaisons possible, à raison d'une combinaison par seconde, il lui faudra  $4,3 \cdot 10^{19}$  secondes. Sachant que l'univers est âgé d'environ  $5 \cdot 10^{17}$  secondes, on se rend bien compte que la méthode de résolution aléatoire est tout à fait inenvisageable.

## 2.1 Pourquoi le Rubik's Cube ?

Le Kube a toujours été, pour nous en tout cas, un objet mystérieux. En effet, comme la plupart des familles, les notre possédaient un vieux rubik's cube poussiéreux oublié au fond d'un placard et réussir à le résoudre à longtemps été un défis hors de notre portée.

Cependant ce n'est que plus récemment (au cours de notre première année de classe préparatoire) que nous avons eu l'idée d'apprendre une méthode de résolution afin d'en finir avec ce casse tête. Mais alors que nous pensions mettre cela derrière nous, c'est en voyant l'ingénieux Cubestormer à l'œuvre que nous avons décidé que écrire un programme capable de résoudre n'importe quel Kube serait notre nouveau défis et TIPE.

## 2.2 Comment créer un “Rubik's Cube informatique” ?

Une fois le sujet clairement défini dans nos esprits, un premier problème est apparu : Comment modéliser le Kube sur ordinateur ?

Si le choix du langage de programmation n'a pas été très long à prendre (Camllight est le langage informatique enseigné en classe préparatoire et l'utiliser nous a paru être un excellent moyen de progresser dans cette matière), trouver comment représenter le Kube sur ordinateur a été plus complexe. Plusieurs structures de données étaient envisageables, comme par exemple la liste, le tableau ou bien encore les matrices. Cependant nous nous sommes rendu compte que celui qui semblait le plus naturel ici était le tableau (ou vecteur Caml) un fois le cube déplié sous forme de patron et ses cases numérotées. (voir figure 2.1)

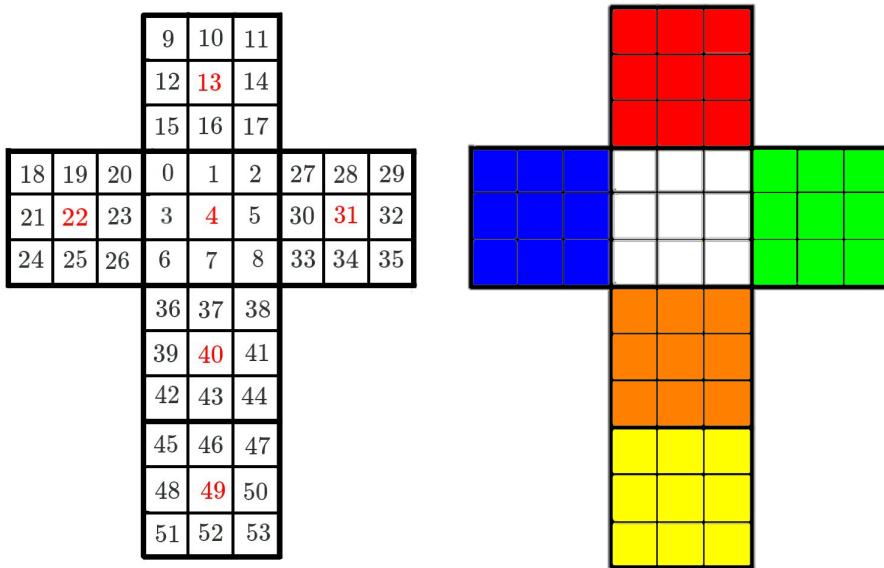


FIGURE 2.1: Représentation sous forme de patron

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs	R	R	R	R	R	R	R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs	B	B	B	B	B	B	B	B	B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G	G	G	G	G	G	G	G	G
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O	O	O	O	O	O	O
Indices	45	46	47	48	49	50	51	52	53
Valeurs	Y	Y	Y	Y	Y	Y	Y	Y	Y

TABLE 2.1: Vecteur représentant le Kube résolut en Caml

### 2.3 Comment en modéliser les mouvements ?

Quand nous avons un Kube entre les mains, les mouvements nous viennent naturellement, cependant informatiquement il nous a fallu trouver un moyen de définir chacun des mouvements et leurs répercussions sur le vecteur Caml.

Pour une explication détaillée des algorithmes de mouvements, se rapporter à la partie dédiée (3.1).

## 2.4 Quelle méthode de résolution ?

Résoudre le Kube peut se faire de très nombreuses façons (voir annexe 3), c'est donc pour cela qu'il a ensuite fallu choisir une méthode de résolution et la traduire en langage algorithmique puis en Camllight. Le but de notre TIPE étant de comparer l'apprentissage par un humain et par une machine, nous avons choisi d'utiliser la méthode "layer by layer" ("couche par couche") qui est la méthode d'apprentissage pour débutant par excellence. Cette méthode n'est pas la plus efficace en nombre de coups ou encore en temps de résolution, mais elle se prête très bien à l'objectif que nous nous sommes donné.

De plus c'est la seule méthode que nous savons appliquer sur un Kube réel, même si nous connaissons le principe et les bases d'autres méthodes.



### 3.1 Algorithmes de mouvement

#### 3.1.1 Préambule à la création de fonctions de mouvement

Le jour où Ernő Rubik's créa le Kube, il ne se doutait pas que cet objet allait devenir le plus renommé des casses têtes. En effet, étant prof d'architecture, il l'avait créé afin de faire deviner à ses étudiants le mécanisme interne du Kube permettant de faire tourner indépendamment les six faces initialement monochromes. La coloration du Kube, donnant naissance au casse tête, permis de rendre chaque mouvement unique. La première chose que nous avons donc du faire fut de décomposer le cube en sous systèmes physiques afin d'en étudier les mouvements. Pour cela nous avons défini un repère orthonormé  $(O, \vec{x}, \vec{y}, \vec{z})$  avec pour origine le centre géométrique du cube, point fixe par toute rotation, sur la représentation tri-dimensionnel du Kube.

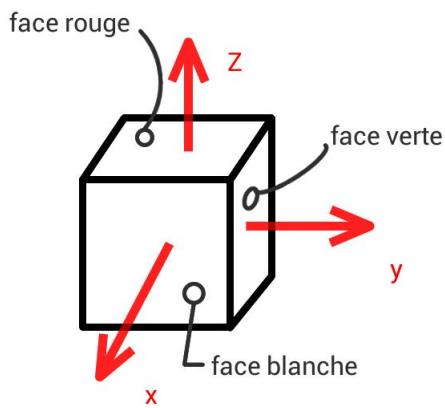


FIGURE 3.1: Kube repéré

Chaque couronne est un sous système. On remarque que chaque sous

### 3. ALGORITHME

---

système est interdépendant. Une rotation sur l'un deux laisse la couronne opposée inchangée mais agit sur toutes les autres.

On appellera :

— rotation  $R_z$  la rotation d'angle  $\theta$  définie par :

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

— rotation  $R_y$  la rotation d'angle  $\theta$  définie par :

$$\begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

— rotation  $R_x$  la rotation d'angle  $\theta$  définie par :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

Remarquons que  $\theta$  doit être défini congru à 0 modulo  $\pi/2$  afin de représenter un mouvement régulier. En ne considérant pas les identités ( $\theta = 0$ ), chacune des 6 rotations ainsi définies se décline en trois sous rotations (d'angle  $\pi/2$ ,  $\pi$  et  $3\pi/4$ ). Cependant une rotation d'angle  $k\pi/2$  peut se faire par k composition de la rotation de base d'angle  $\pi/2$ . Nous avons donc seulement 6 rotations à définir.

Ces notations mathématiques bien que pertinentes scientifiquement, sont moins naturelles et ne correspondent pas à la vision que peut avoir un cubeur. En effet, la manipulation intuitive du cube implique des changements de repère permettant de ramener tous les mouvements du Cube autour d'un axe unique. Pour cette raison on référence tous les mouvements du Cube avec 6 rotations :

- $w(\theta) = R_x(\theta)$ (couronne W)
- $g(\theta) = R_y(\theta)$ (couronne G)
- $r(\theta) = R_z(\theta)$ (couronne R)
- $y(\theta) = R_x(-\theta)$ (couronne Y)
- $b(\theta) = R_y(-\theta)$ (couronne B)
- $o(\theta) = R_z(-\theta)$ (couronne O)

De cette manière on définit bien la rotation horaire lorsque la face qui bouge est face à nous, ce qui correspond à la rotation intuitive.

La représentation tri-dimensionnelle du Cube n'étant pas adaptée au traitement algorithmique, on envisage d'abord le Cube sous sa forme "dépliée", c'est à dire par son patron. Nous avons ensuite numéroté les vignettes. (3.2)

Puis naturellement, l'idée de représenter le Cube sous la forme d'un tableau  $2 \times 54$  nous est venue. Ce qui nous donne le type sur lequel nous allons travailler en Caml : le vecteur de caractère de longueur 54. Ce qui justifie de travailler dans l'espace :

$$K = \{WRBGOY\}^{54}$$

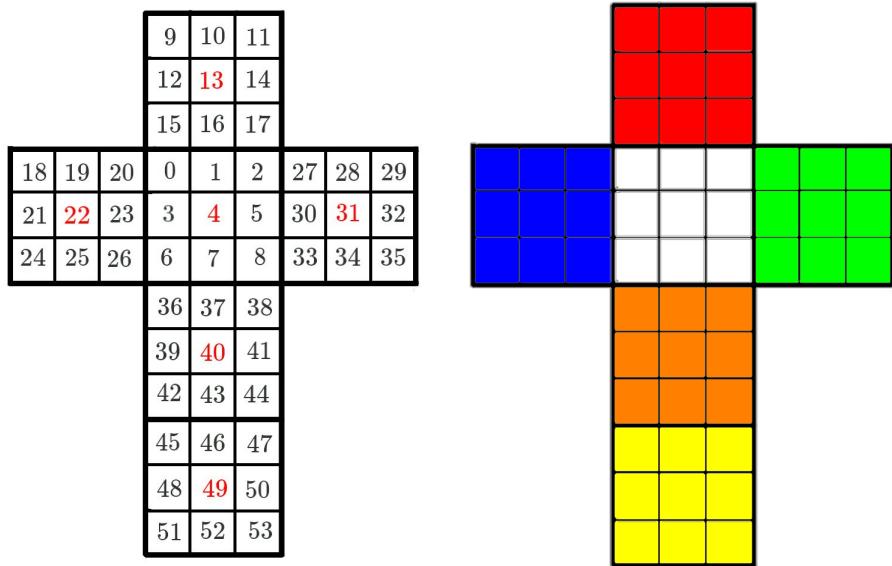


FIGURE 3.2: Représentation sous forme de patron

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs	R	R	R	R	R	R	R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs	B	B	B	B	B	B	B	B	B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G	G	G	G	G	G	G	G	G
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O	O	O	O	O	O	O
Indices	45	46	47	48	49	50	51	52	53
Valeurs	Y	Y	Y	Y	Y	Y	Y	Y	Y

TABLE 3.1: Vecteur représentant le Kube résolut en Caml

Dans cet espace les rotations deviennent alors de simples réorganisation d'éléments du vecteur ainsi définit. La résolution du Kube devient donc un tri de vecteur à l'aide de permutations de 20-cycles (appellation que nous justifierons plus loin) correspondant aux mouvements réels du Kube décris plus tôt.

### 3.1.2 Exemple développé de la rotation de la face blanche

On s'intéresse ici à la rotation d'un quart de tour dans le sens horaire de la face blanche, c'est à dire à la rotation :

$$w\left(\frac{\pi}{2}\right)$$

On a décomposé cette rotation en 5 sous-fonctions indépendantes, applicable sans avoir à considérer un ordre entre elles.

#### 3.1.2.1 Première sous-fonction : Arrêtes faces

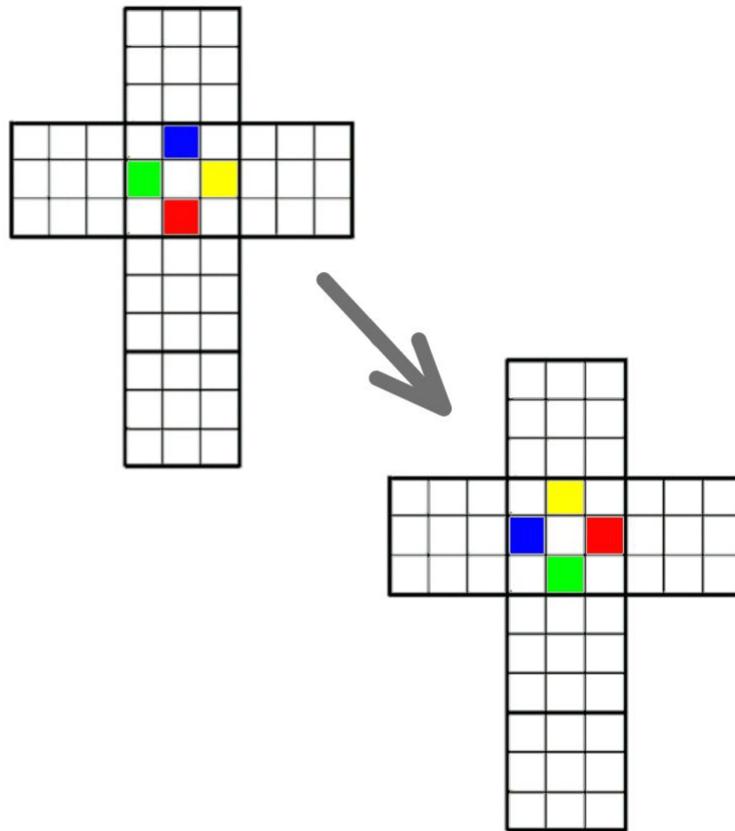


FIGURE 3.3: Sous-fonction : Arrêtes faces

```
1 let aretes_fW v =
```

```

2 v.(4) <- v.(1);
3 v.(1) <- v.(3);
4 v.(3) <- v.(7);
5 v.(7) <- v.(5);
6 v.(5) <- v.(4)
7 ;;

```

### 3.1.2.2 Deuxième sous-fonction : Coins faces

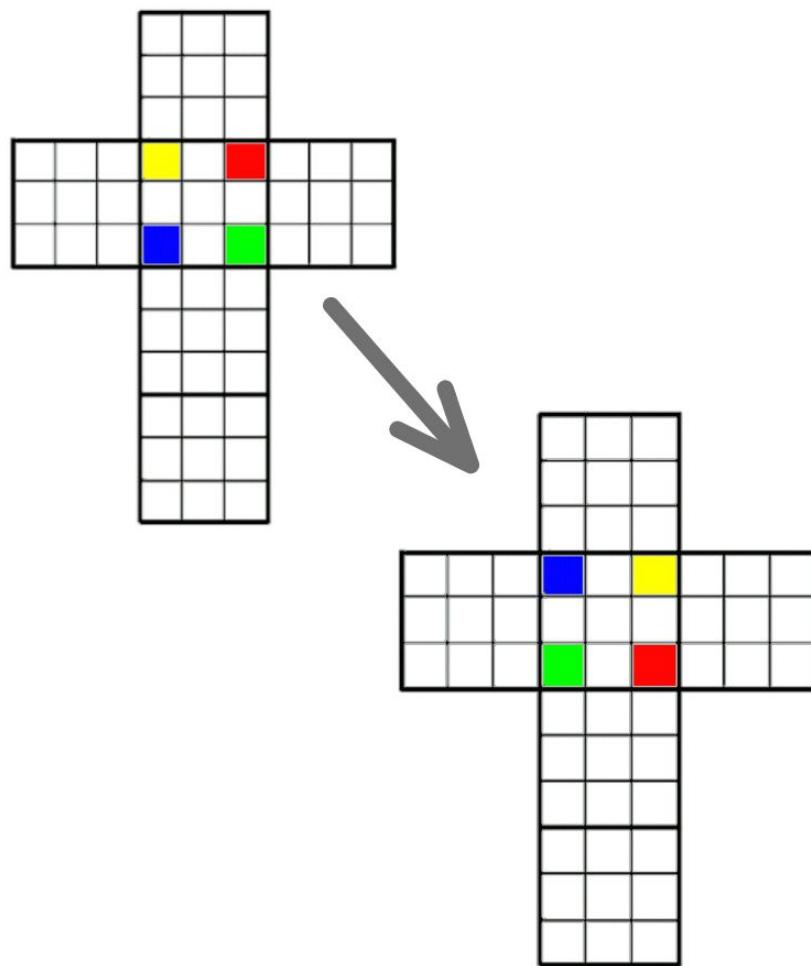


FIGURE 3.4: Sous-fonction : Coins faces

```
1 let coins_fW v =
```

### 3. ALGORITHME

---

```
2 v.(4) <- v.(0);  
3 v.(0) <- v.(6);  
4 v.(6) <- v.(8);  
5 v.(8) <- v.(2);  
6 v.(2) <- v.(4)  
7 ;;
```

#### 3.1.2.3 Troisième sous-fonction : Arrêtes externes

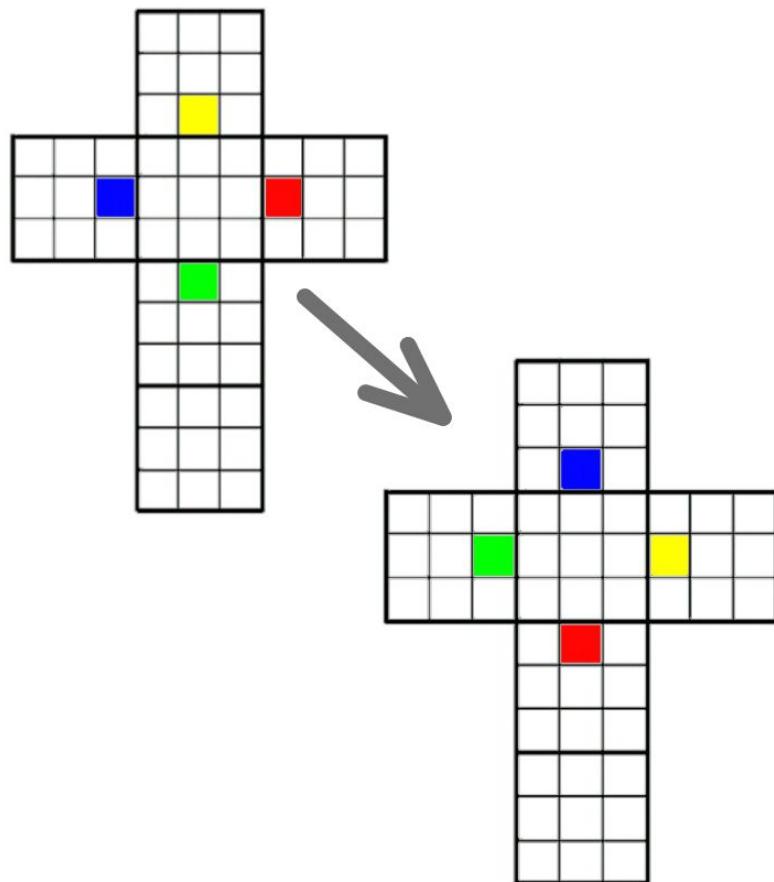


FIGURE 3.5: Sous-fonction : Arrêtes externes

```
1 let aretes_aW v =
```

```

2 v.(4) <- v.(16);
3 v.(16) <- v.(23);
4 v.(23) <- v.(37);
5 v.(37) <- v.(30);
6 v.(30) <- v.(4)
7 ;;

```

### 3.1.2.4 Quatrième sous-fonction : Coins externes 1

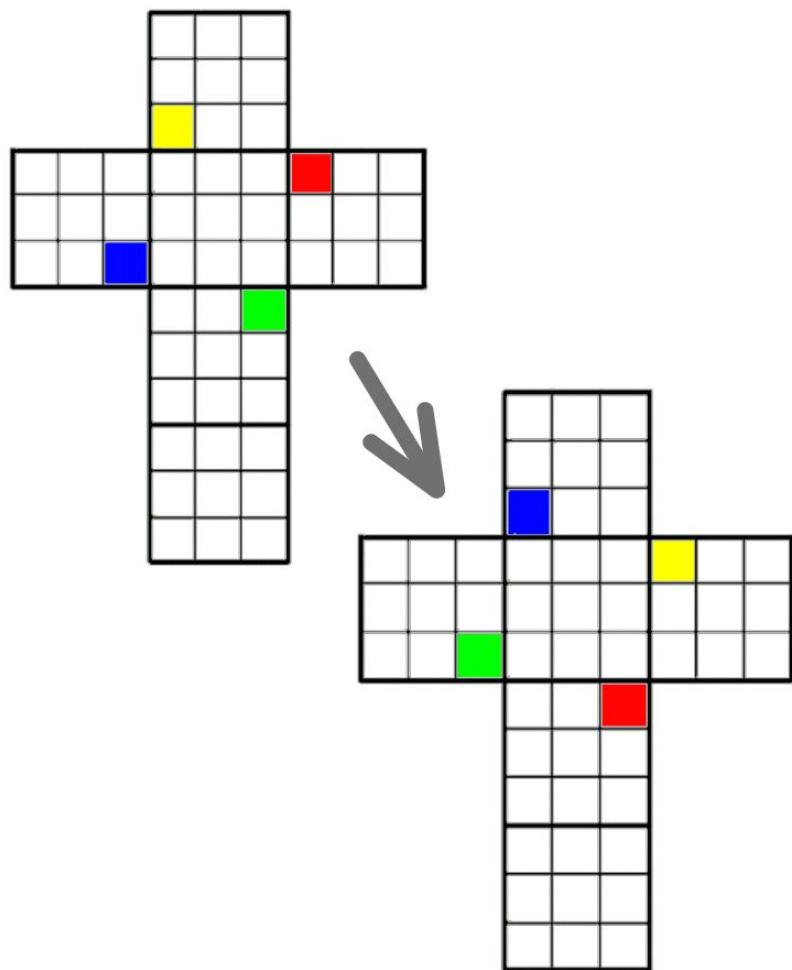


FIGURE 3.6: Sous-fonction : Coins externes 1

```
1 let coins_a1W v =
```

### 3. ALGORITHME

---

```
2 v.(4) <- v.(15);  
3 v.(15) <- v.(26);  
4 v.(26) <- v.(38);  
5 v.(38) <- v.(27);  
6 v.(27) <- v.(4)  
7 ;;
```

#### 3.1.2.5 Cinquième sous-fonction : Coins externes 2

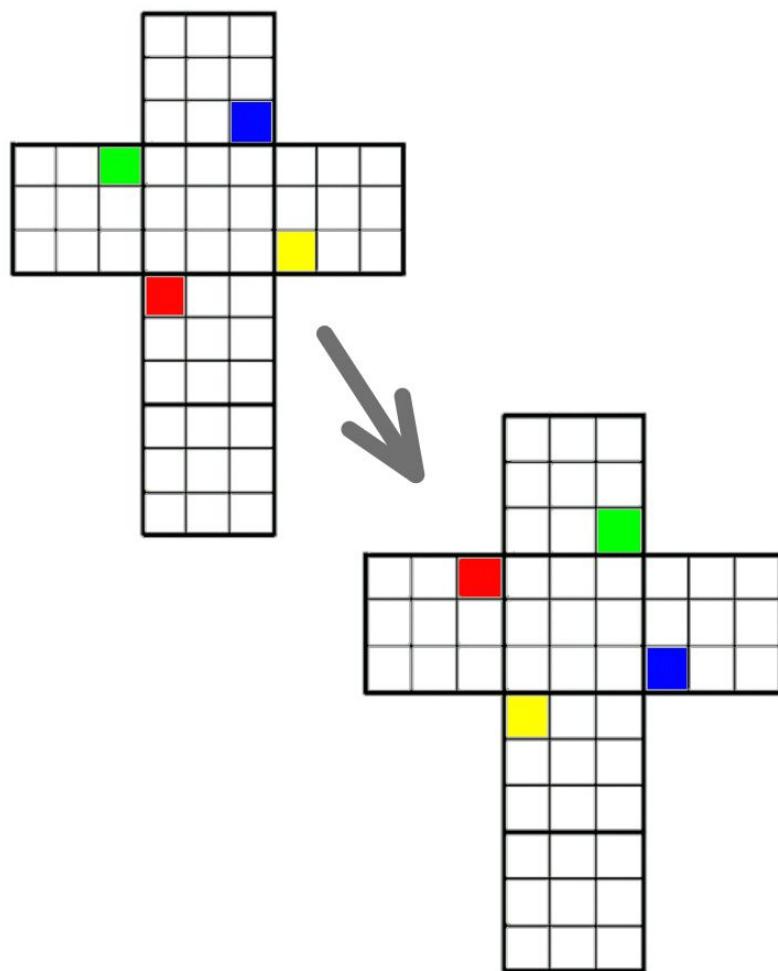


FIGURE 3.7: Sous-fonction : Coins externes 2

```
1 let coins_a2W v =
```

```
2 v.(4) <- v.(17);
3 v.(17) <- v.(20);
4 v.(20) <- v.(36);
5 v.(36) <- v.(33);
6 v.(33) <- v.(4)
7;;
```

### 3.1.2.6 Fonction général de rotation de la face blanche

```
1 let w v =
2 coins_fW v ;
3 aretes_fW v ;
4 coins_a1W v ;
5 coins_a2W v ;
6 aretes_aW v ;
7 v.(4) <- "W"
8;;
```

### 3. ALGORITHME

---

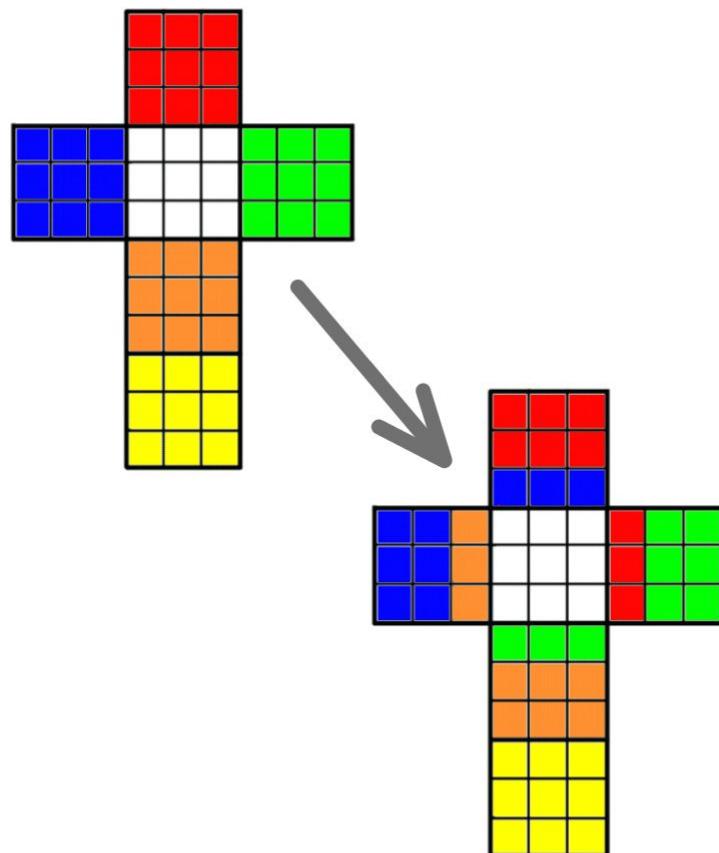


FIGURE 3.8: Fonction général de rotation de la face blanche

## 3.2 Étape 1

### 3.2.1 Description du but de l'étape

L'étape 1 prend comme argument un cube mélangé quelconque et son but est de faire ce qu'on appelle la “croix blanche alignée” (voir 3.9).

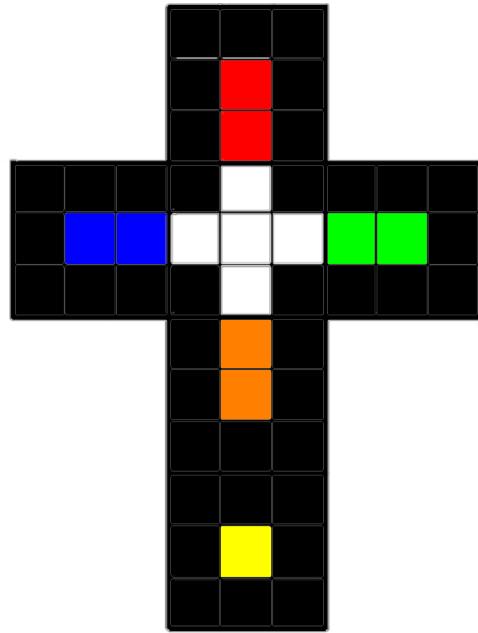


FIGURE 3.9: Représentation du patron du cube à la fin de l'étape 1

Indices	0	1	2	3	4	5	6	7	8
Valeurs		W		W	W	W		W	
Indices	9	10	11	12	13	14	15	16	17
Valeurs					R			R	
Indices	18	19	20	21	22	23	24	25	26
Valeurs					B	B			
Indices	27	28	29	30	31	32	33	34	35
Valeurs				G	G				
Indices	36	37	38	39	40	41	42	43	44
Valeurs		O		O					
Indices	45	46	47	48	49	50	51	52	53
Valeurs					Y				

TABLE 3.2: Vecteur représentant le Kube en fin d'étape 1

### 3.2.2 Description des macros de l'étape 1

#### 3.2.2.1 Macro de scan du vecteur “vect\_double”

```

1 let scan_double_aux v_d a b =
2   let position = ref 0 in
3     while ((v_d.(!position) <> (a,b)) && (v_d.(!position) <> (b,a))) do incr position done;
4   !position
5 ;;
6
7 let scan_double v a b = scan_double_aux (vect_double v) a b;;

```

Cette macro est un simple outil de scan du vecteur “vect\_double”, elle prend 2 arguments strings correspondant aux 2 couleurs de la pièce à rechercher dans le vecteur. Elle renvoie un entier correspondant à l'indice de la pièce cherchée dans ce même vecteur.

#### 3.2.2.2 Macro de placement de la pièce (“R” “W”)

```

1 let etape_1_placement_r v =
2   match (scan_double v "R" "W") with
3     |0 -> ()
4     |1 -> mouv 1 "W" v
5     |2 -> mouv 3 "W" v
6     |3 -> mouv 2 "W" v
7     |4 -> mouv 3 "R" v
8     |5 -> mouv 1 "R" v
9     |6 -> mouv 2 "R" v
10    |7 -> begin mouv 1 "O" v ; mouv 2 "W" v end
11    |8 -> begin mouv 2 "B" v ; mouv 1 "W" v end
12    |9 -> begin mouv 1 "G" v ; mouv 3 "W" v end
13    |10 -> begin mouv 2 "G" v ; mouv 3 "W" v end
14    |11 -> begin mouv 2 "O" v ; mouv 2 "W" v end
15    |_ -> failwith "Corrupted data etape 1"
16 ;;

```

Cette macro est une étude exhaustive des positions possibles pour la pièce (“R” “W”) avec un placement au cas par cas ensuite. à la fin de cette sous fonction, la pièce est à la bonne place mais pas forcément avec la bonne orientation.

#### 3.2.2.3 Macro de placement de la pièce (“B” “W”)

```

1 let etape_1_placement_b v =
2   match (scan_double v "B" "W") with
3     |0 -> failwith "ERREUR ETAPE 1"
4     |1 -> ()
5     |2 -> begin mouv 1 "R" v; mouv 2 "W" v; mouv 3 "R" v end
6     |3 -> begin mouv 1 "R" v; mouv 1 "W" v; mouv 3 "R" v end
7     |4 -> mouv 1 "B" v
8     |5 -> begin mouv 1 "R" v; mouv 3 "W" v; mouv 3 "R" v end
9     |6 -> begin mouv 1 "Y" v; mouv 2 "B" v end
10    |7 -> mouv 3 "B" v
11    |8 -> mouv 2 "B" v

```

```

12    | 9 -> begin mouv 2 "O" v ; mouv 3 "B" v end
13    | 10 -> begin mouv 2 "Y" v ; mouv 2 "B" v end
14    | 11 -> begin mouv 3 "Y" v ; mouv 2 "B" v end
15    | _ -> failwith "Corrupted data etape 1"
16 ;;

```

Cette macro est elle aussi une étude exhaustive puis un placement au cas par cas, par contre elle prend en compte le fait que la pièce (“R” “W”) est déjà placée et donc la laisse toujours en place.

### 3.2.2.4 Macro de placement de la pièce (“G” “W”)

```

1 let etape_1_placement_g v =
2   match (scan_double v "W" "G") with
3     | 0 -> failwith "ERREUR ETAPE 1"
4     | 1 -> failwith "ERREUR ETAPE 1"
5     | 2 -> ()
6     | 3 -> begin mouv 1 "O" v; mouv 1 "G" v end
7     | 4 -> begin mouv 3 "B" v; mouv 2 "Y" v; mouv 2 "G" v; mouv
        1 "B" v end
8     | 5 -> mouv 3 "G" v
9     | 6 -> begin mouv 3 "Y" v; mouv 2 "G" v end
10    | 7 -> begin mouv 2 "O" v; mouv 1 "G" v end
11    | 8 -> begin mouv 2 "Y" v; mouv 2 "G" v end
12    | 9 -> mouv 1 "G" v
13    | 10 -> mouv 2 "G" v
14    | 11 -> begin mouv 1 "Y" v; mouv 2 "G" v end
15    | _ -> failwith "Corrupted data etape 1"
16 ;;

```

Idem aux deux macros précédentes mais en gardant en place les pièces (“R” “W”) et (“B” “W”).

### 3.2.2.5 Macro de placement de la pièce (“O” “W”)

```

1 let etape_1_placement_o v =
2   match (scan_double v "O" "W") with
3     | 0 -> failwith "ERREUR ETAPE 1"
4     | 1 -> failwith "ERREUR ETAPE 1"
5     | 2 -> failwith "ERREUR ETAPE 1"
6     | 3 -> ()
7     | 4 -> begin mouv 3 "B" v; mouv 1 "Y" v; mouv 2 "O" v; mouv
        1 "B" v end
8     | 5 -> begin mouv 1 "G" v; mouv 3 "Y" v; mouv 2 "O" v; mouv
        3 "G" v end
9     | 6 -> begin mouv 2 "Y" v; mouv 2 "O" v end
10    | 7 -> mouv 1 "O" v
11    | 8 -> begin mouv 1 "Y" v; mouv 2 "O" v end
12    | 9 -> mouv 3 "O" v
13    | 10 -> begin mouv 3 "Y" v; mouv 2 "O" v end
14    | 11 -> mouv 2 "O" v
15    | _ -> failwith "Corrupted data etape 1"
16 ;;

```

### 3. ALGORITHME

---

Idem aux trois macros précédentes mais en gardant en place les pièces ("R" "W"), ("B" "W") et ("G" "W").

#### 3.2.2.6 Macro d'orientation des pièces

```

1 let etape_1_replacement v =
2 if (v.(1)= "W") then ()
3 else begin mouv 1 "R" v; mouv 3 "W" v; mouv 1 "B" v; mouv 1 "W"
      v end ;
4
5 if (v.(3)= "W") then ()
6 else begin mouv 1 "B" v; mouv 3 "W" v; mouv 1 "O" v; mouv 1 "W"
      v end ;
7
8 if (v.(5)= "W") then ()
9 else begin mouv 1 "G" v; mouv 3 "W" v; mouv 1 "R" v; mouv 1 "W"
      v end ;
10
11 if (v.(7)= "W") then ()
12 else begin mouv 1 "O" v; mouv 3 "W" v; mouv 1 "G" v; mouv 1 "W"
      v end
13 ;;

```

Cette macro sert à orienter les pièces qui viennent d'être placées grâce aux fonctions précédentes. Elle n'agit que sur la pièce à orienter et laisse donc en place les pièces bien placées.

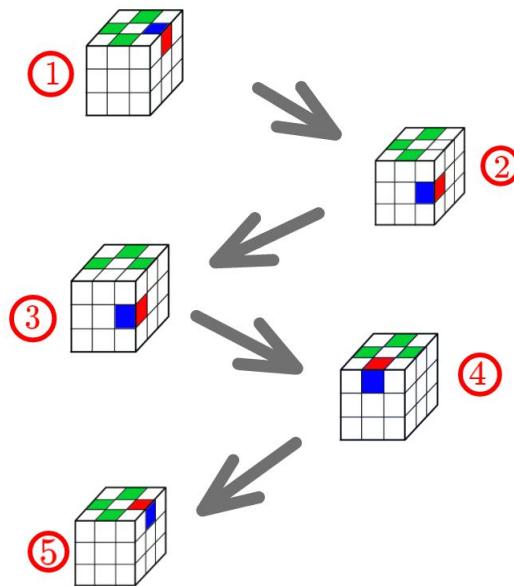


FIGURE 3.10: Description de la macro d'orientation

### 3.2.3 Le programme Étape 1

Dans cette étape, on utilise toutes les macros définies précédemment afin de traiter le placement de toutes les arrêtes blanches une par une, une méthode au cas par cas s'est imposée ici car aucune méthode algorithmique n'est envisageable pour ce mouvement. En effet dans la méthode "Layer by layer", l'étape 1 est décrite par : "faites la croix blanche", il n'y a pas de technique ici, c'est une étape très intuitive pour l'homme. Cependant nous avons tout de même trouvé et implémenté la macro "replacement" qui permet d'orienter une pièce déjà placée. Ainsi il nous suffit de placer les pièces une par une sans nous occuper de leur orientation puis d'orienter celles qui ne le sont pas.

```

1 let etape_1_placement_r v =
2   match (scan_double v "R" "W") with
3     |0 -> ()
4     |1 -> mouv 1 "W" v
5     |2 -> mouv 3 "W" v
6     |3 -> mouv 2 "W" v
7     |4 -> mouv 3 "R" v
8     |5 -> mouv 1 "R" v
9     |6 -> mouv 2 "R" v
10    |7 -> begin mouv 1 "O" v ; mouv 2 "W" v end
11    |8 -> begin mouv 2 "B" v ; mouv 1 "W" v end
12    |9 -> begin mouv 1 "G" v ; mouv 3 "W" v end
13    |10 -> begin mouv 2 "G" v ; mouv 3 "W" v end
14    |11 -> begin mouv 2 "O" v ; mouv 2 "W" v end
15    |_ -> failwith "Corrupted data etape 1"
16  ;;
17
18 let etape_1_placement_b v =
19   match (scan_double v "B" "W") with
20     |0 -> failwith "ERREUR ETAPE 1"
21     |1 -> ()
22     |2 -> begin mouv 1 "R" v; mouv 2 "W" v; mouv 3 "R" v end
23     |3 -> begin mouv 1 "R" v; mouv 1 "W" v; mouv 3 "R" v end
24     |4 -> mouv 1 "B" v
25     |5 -> begin mouv 1 "R" v; mouv 3 "W" v; mouv 3 "R" v end
26     |6 -> begin mouv 1 "Y" v; mouv 2 "B" v end
27     |7 -> mouv 3 "B" v
28     |8 -> mouv 2 "B" v
29     |9 -> begin mouv 2 "O" v ; mouv 3 "B" v end
30     |10 -> begin mouv 2 "Y" v ; mouv 2 "B" v end
31     |11 -> begin mouv 3 "Y" v ; mouv 2 "B" v end
32     |_ -> failwith "Corrupted data etape 1"
33  ;;
34
35 let etape_1_placement_g v =
36   match (scan_double v "W" "G") with
37     |0 -> failwith "ERREUR ETAPE 1"
38     |1 -> failwith "ERREUR ETAPE 1"
39     |2 -> ()
40     |3 -> begin mouv 1 "O" v; mouv 1 "G" v end

```

### 3. ALGORITHME

---

```

41  |4 -> begin mouv 3 "B" v; mouv 2 "Y" v; mouv 2 "G" v; mouv
42    1 "B" v end
43  |5 -> mouv 3 "G" v
44  |6 -> begin mouv 3 "Y" v; mouv 2 "G" v end
45  |7 -> begin mouv 2 "O" v; mouv 1 "G" v end
46  |8 -> begin mouv 2 "Y" v; mouv 2 "G" v end
47  |9 -> mouv 1 "G" v
48  |10 -> mouv 2 "G" v
49  |_ -> failwith "Corrupted data etape 1"
50 ;;
51
52 let etape_1_placement_o v =
53 match (scan_double v "O" "W") with
54  |0 -> failwith "ERREUR ETAPE 1"
55  |1 -> failwith "ERREUR ETAPE 1"
56  |2 -> failwith "ERREUR ETAPE 1"
57  |3 -> ()
58  |4 -> begin mouv 3 "B" v; mouv 1 "Y" v; mouv 2 "O" v; mouv
59    1 "B" v end
60  |5 -> begin mouv 1 "G" v; mouv 3 "Y" v; mouv 2 "O" v; mouv
61    3 "G" v end
62  |6 -> begin mouv 2 "Y" v; mouv 2 "O" v end
63  |7 -> mouv 1 "O" v
64  |8 -> begin mouv 1 "Y" v; mouv 2 "O" v end
65  |9 -> mouv 3 "O" v
66  |10 -> begin mouv 3 "Y" v; mouv 2 "O" v end
67  |11 -> mouv 2 "O" v
68  |_ -> failwith "Corrupted data etape 1"
69 ;;
70
71 let etape_1_replacement v =
72 if (v.(1)= "W") then ()
73 else begin mouv 1 "R" v; mouv 3 "W" v; mouv 1 "B" v; mouv 1 "W"
74   v end ;
75
76 if (v.(3)= "W") then ()
77 else begin mouv 1 "B" v; mouv 3 "W" v; mouv 1 "O" v; mouv 1 "W"
78   v end ;
79
80 if (v.(5)= "W") then ()
81 else begin mouv 1 "G" v; mouv 3 "W" v; mouv 1 "R" v; mouv 1 "W"
82   v end ;
83
84 let etape1 v =
85  etape_1_placement_r v;
86  etape_1_placement_b v;

```

```
87  etape_1_placement_g v;
88  etape_1_placement_o v;
89  etape_1_replacement v
90 ;;
```

### 3.3 Étape 2

#### 3.3.1 Description du but de l'étape

L'étape 2 prend comme argument un cube avec la “croix blanche aligné” et son but et de faire ce qu'on appelle la “première couronne” (voir 3.11).

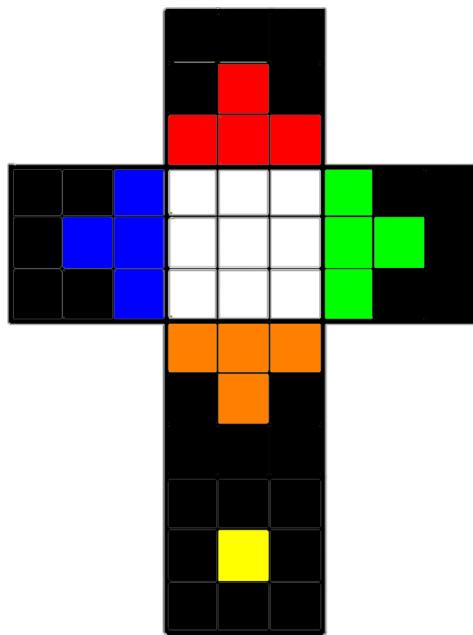


FIGURE 3.11: Représentation du patron du cube à la fin de l'étape 2

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs					R		R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs			B		B	B			B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G			G	G		G		
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O		O				
Indices	45	46	47	48	49	50	51	52	53
Valeurs					Y				

TABLE 3.3: Vecteur représentant le Kube en fin d'étape 2

### 3.3.2 Description des macros de l'étape 2

#### 3.3.2.1 Macro de position des coins de la face “White”

```

1 let position_coins_aux v a b =
2 mouv 3 a v;
3 mouv 3 b v;
4 mouv 1 a v;
5 mouv 1 b v
6 ;;

```

Cette macro échange deux pièces, l'une appartenant à la couronne supérieure, l'autre étant située directement en dessous sur la couronne inférieure. (voir figure 1).

C'est un mouvement absolument conservatif en 6 répétitions. il permet de faire prendre aux deux pièces concernées toutes les positions possibles sur ces deux emplacements en 6 temps.

Les arguments “a” et “b” sont des strings correspondant aux couleurs des faces sur lesquelles on applique cette macro, dans cette étape, on la curryifie en :

```
1 let position_coins_fW v a = position_coins_aux v a "Y" ;;
```

Le seul argument restant : “a”, est un string correspondant à la couleur de la face contenant les deux pièces que l'on souhaite échanger.

#### 3.3.2.2 Macro de scan du vecteur “vect\_triple”

```

1 let scan_triple_aux v_t a b c =
2   let position = ref 0 in
3     while ( (v_t.(!position) <> (a,b,c)) && (v_t.(!position) <>
4           (a,c,b)) && (v_t.(!position) <> (b,a,c)) && (v_t.(!
5             position) <> (b,c,a)) && (v_t.(!position) <> (c,a,b)) &&
6               (v_t.(!position) <> (c,b,a)) && (!position <= 7) ) do
7       incr position done;
8   if (!position <8) then !position
9   else failwith "erreur scan_triple"
10  ;;
11
12 let scan_triple v a b c = scan_triple_aux (vect_triple v) a b c
13   ;;

```

Cette macro est un simple outil de scan du vecteur “vect\_triple”, elle prend 3 arguments strings correspondant aux 3 couleurs de la pièce à rechercher dans le vecteur. Elle renvoie un entier correspondant à l'indice de la pièce cherchée dans ce même vecteur.

Une sécurité a été ajoutée afin d'éviter les boucles infinies, si la variable locale “position” dépasse la longueur du vecteur, cette fonction revoie une erreur.

#### 3.3.2.3 Macro de sortie de coins

```

1 let sortie_coins v a b =
2   mouv 3 a v;

```

### 3. ALGORITHME

---

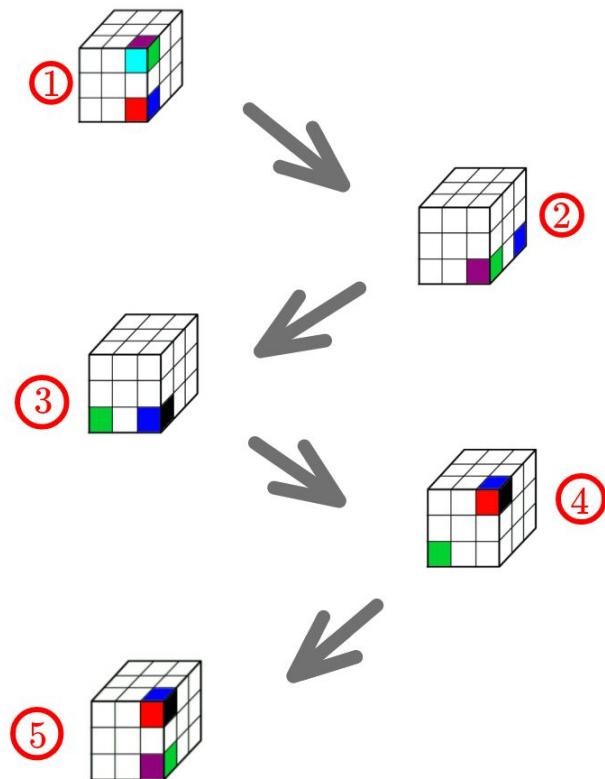


FIGURE 3.12: Description de la macro de position des coins

```

3   mouv 1 b v;
4   mouv 1 a v
5 ;;

```

Cette macro place une pièce de la couronne supérieure directement en dessous sur la couronne inférieure. (voir figure deux)

On la curryifie en 4 sous-fonctions :

— “sortie\_coins\_0” sortant la pièce ayant pour indice 0 dans vect\_triple.

```
1 let sortie_coins_0 v = sortie_coins v "B" "Y";;
```

— “sortie\_coins\_1” sortant la pièce ayant pour indice 1 dans vect\_triple.

```
1 let sortie_coins_1 v = sortie_coins v "R" "Y";;
```

— “sortie\_coins\_2” sortant la pièce ayant pour indice 2 dans vect\_triple.

```
1 let sortie_coins_2 v = sortie_coins v "O" "Y";;
```

— “sortie\_coins\_3” sortant la pièce ayant pour indice 3 dans vect\_triple.

```
1 let sortie_coins_3 v = sortie_coins v "G" "Y";;
```

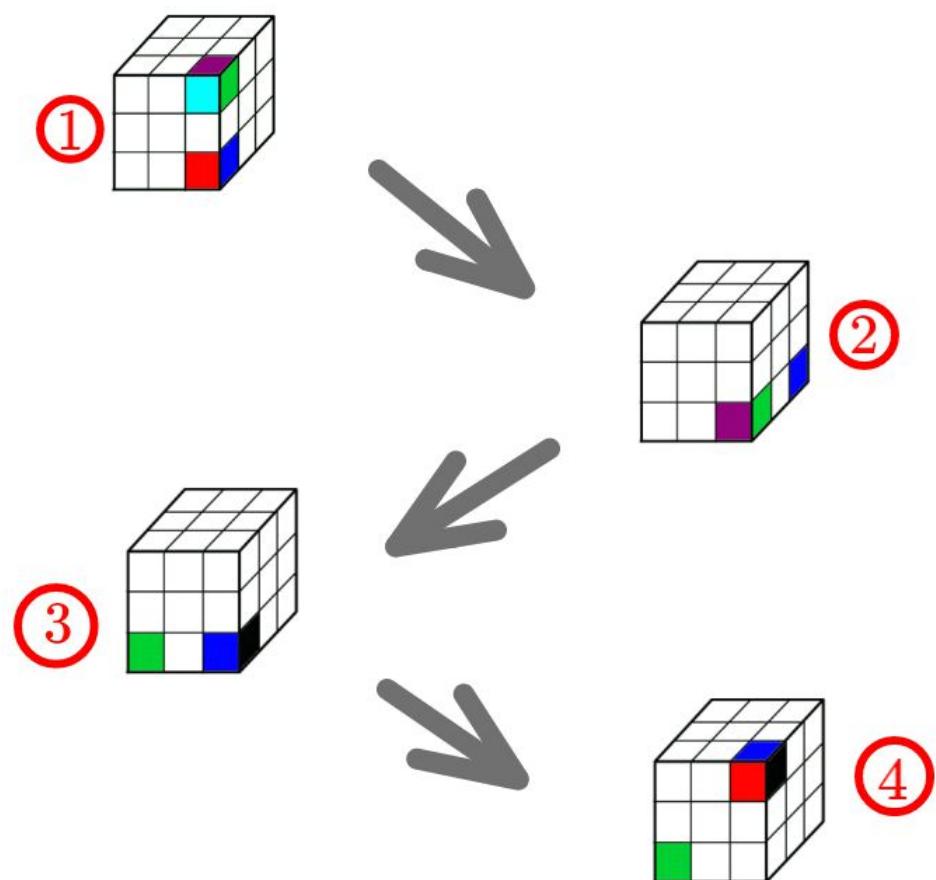


FIGURE 3.13: Description de la macro de position des coins

### 3.3.2.4 Macro “temps6”

```

1 let temps6 v a b c place nb=
2   let temp = ref 1 in
3     if (nb = 0) then
4       while (((vect_triple v).(place) <> (a,b,c)) && (!temp <
7)) do
5       position_coins_fW v c;
6       incr temp
7     done
8   else
9     while (((vect_triple v).(place) <> (a,b,c)) && (!temp <
7)) do
10    position_coins_fW v b;
11    incr temp
12  done
13 ;;

```

Cette macro à pour but de placer la pièce A (voir figure 3) à sa place définitive (bonne orientation) sur la couronne supérieure (à la place de la place B). C'est une boucle utilisant le principe de conservation de la macro “position\_coins\_fW” ainsi que le fait qu'en 6 répétitions, les deux pièces A et B ont pris toutes les orientations possibles. Les arguments “a”, “b” et “c” sont des strings désignant les couleurs et la bonne orientation a donner à la pièce en B. “place” quant à lui correspond à l'indice de la pièce que l'on veut placer dans vect\_triple.

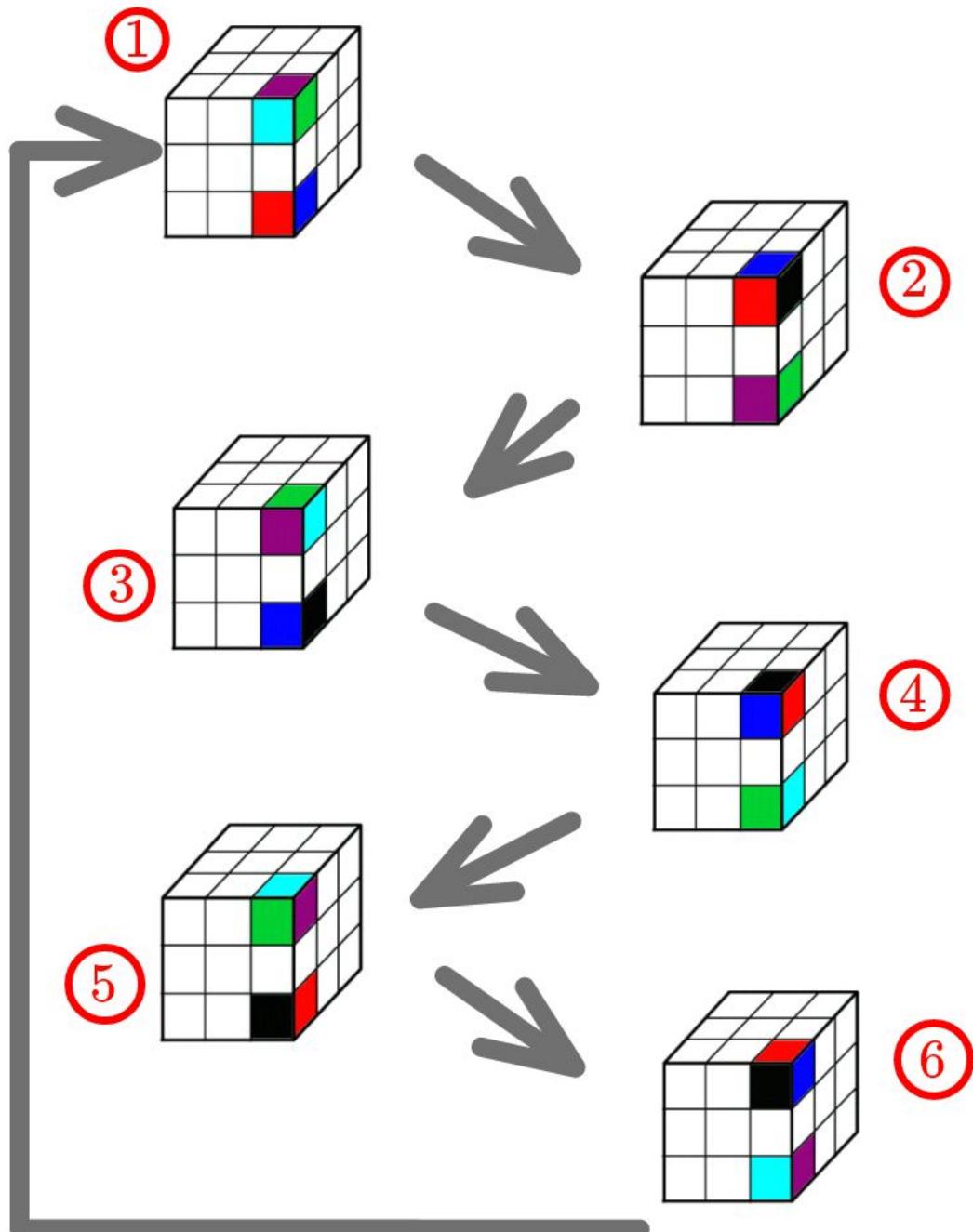


FIGURE 3.14: Description du principe de conservation en 6 temps

### 3.3.2.5 Macro “placement\_coins\_fW”

```

1 let placement_coins_fW v a b c place nb = let test = ref 0 in
2   while (((vect_triple v).(place) <> (a,b,c)) && (!test<50)) do
3     if (!test=50) then failwith "erreur plac_coins"
4     else let temp2 = (place +4) and temp1 = place in
5       match 0 with
6         | _ when ((scan_triple v a b c)=temp1) -> temps6 v a b c
7           place nb
8         | _ when ((scan_triple v a b c)=0) -> sortie_coins_0 v
9         | _ when ((scan_triple v a b c)=1) -> sortie_coins_1 v
10        | _ when ((scan_triple v a b c)=2) -> sortie_coins_2 v
11        | _ when ((scan_triple v a b c)=3) -> sortie_coins_3 v
12        | _ when ((scan_triple v a b c)=temp2) -> temps6 v a b c
13          place nb
14        | _ -> mouv 1 "Y" v
13   done
14 ;;

```

Le but de cette macro est de placer le coin d’indice “place” dans le vecteur triple. Pour cela, une fonction de matching va boucler tant que la pièce ne sera pas à sa place. Si la pièce est bien placée, cette fonction ne fera rien, si elle est juste “en dessous”, la fonction “temps6” va venir la placer. Enfin si elle est sur l’un des trois coins supérieur ou elle ne doit pas être, la fonction de sortie correspondante est appliquée. Enfin si la pièce n’est dans aucune de ces configurations, on tourne la face jaune d’un cran puis on boucle.

### 3.3.3 Le programme Étape 2

Dans cette étape, on utilise toutes les macros définies précédemment afin de placer chaque coin un par un.

```

1 let placement_coins_fW_0 v = placement_coins_fW v "W" "R" "B"
2   0 0;;
2 let placement_coins_fW_1 v = placement_coins_fW v "W" "R" "G"
3   1 1;;
3 let placement_coins_fW_2 v = placement_coins_fW v "W" "B" "O"
4   2 0;;
4 let placement_coins_fW_3 v = placement_coins_fW v "W" "G" "O"
5   3 1;;
5
6
7
8 let etape2 v =
9   placement_coins_fW_0 v ;
10  placement_coins_fW_1 v ;
11  placement_coins_fW_2 v ;
12  placement_coins_fW_3 v
13 ;;

```

## 3.4 Étape 3

### 3.4.1 Description du but de l'étape

L'étape 3 prend comme argument un cube avec la “première couronne” et son but et de faire ce qu'on appelle la “dixième couronne” (voir 3.15).

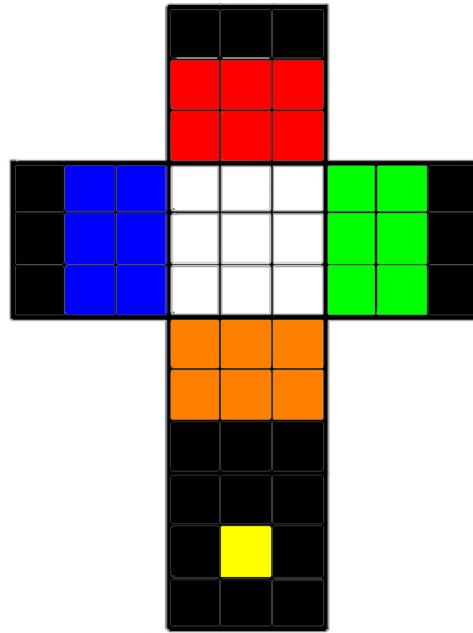


FIGURE 3.15: Représentation du patron du cube à la fin de l'étape 3

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs				R	R	R	R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs		B	B		B	B		B	B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G	G		G	G		G	G	
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O	O	O	O			
Indices	45	46	47	48	49	50	51	52	53
Valeurs					Y				

TABLE 3.4: Vecteur représentant le Kube en fin d'étape 3

### 3. ALGORITHME

---

#### 3.4.2 Description des macros de l'étape 3

##### 3.4.2.1 Macro “valeur”

```

1 let valeur a =
2   match a with
3     | "R" -> 1
4     | "B" -> 2
5     | "O" -> 3
6     | "G" -> 4
7     | _    -> failwith "erreur val"
8 ;;

```

Cette macro toute simple va servir à attribuer aux couleurs “R”, “B”, “O” et “G” les valeurs 1, 2, 3 et 4.

##### 3.4.2.2 Macro “oriente”

```

1 let oriente a b = let n = ((valeur a) - (valeur b)) in
2   match n with
3     | 1 -> gauche
4     | (-3) -> gauche
5     | 3 -> droite
6     | (-1) -> droite
7     | 2 -> failwith "2"
8     | (-2) -> failwith "-2"
9     | 0 -> failwith "louche 0"
10    | _ -> failwith "erreur ori"
11 ;;

```

Cette Macro sera utilisée dans la macro suivante et sert à définir si le mouvement devra être fait à gauche ou à droite.

Valeur a	Valeur b	Orientation	$(\text{Valeur a}) - (\text{Valeur b})$
R (1)	B (2)	droite	$-1 \equiv 3 \pmod{4}$
R (1)	G (4)	gauche	$-3 \equiv 1 \pmod{4}$
B (2)	O (3)	droite	$-1 \equiv 3 \pmod{4}$
B (2)	R (1)	gauche	$1 \equiv 1 \pmod{4}$
O (3)	G (4)	droite	$-1 \equiv 3 \pmod{4}$
O (3)	B (2)	gauche	$1 \equiv 1 \pmod{4}$
G (4)	R (1)	droite	$3 \equiv 3 \pmod{4}$
G (4)	O (3)	gauche	$1 \equiv 1 \pmod{4}$

Elle se base sur le fait que l'orientation gauche sera toujours à faire pour placer la pièce (“a” “b”) lorsque :

$$((\text{valeur a}) - (\text{valeur b})) \equiv 1 \pmod{4}$$

Et que l'orientation droite sera toujours à faire pour placer la pièce (“a” “b”) lorsque :

$$((\text{valeur a}) - (\text{valeur b})) \equiv 3 \pmod{4}$$

Toute les autres congruences modulo 4 sont donc impossibles et renverront une erreur.

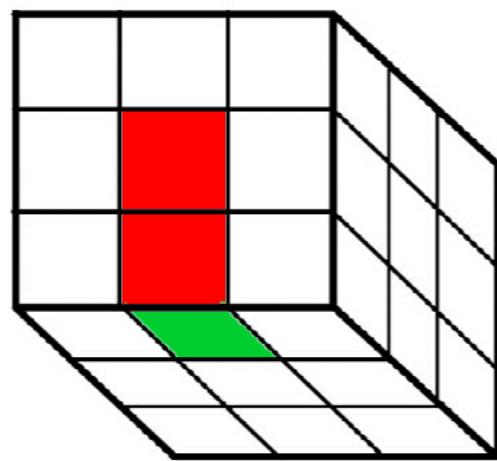


FIGURE 3.16: Explication de la dénomination a, b : ici a = rouge et b = vert

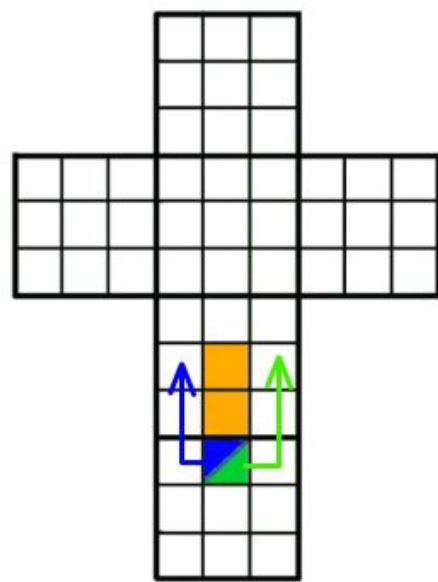


FIGURE 3.17: Explication de la dénomination droite et gauche : ici droite = vert et gauche = bleu

### 3.4.2.3 Macro “manchot”

```
1 let manchot_aux v a b c oriente =
2 if ((oriente a b) = gauche)
3 then begin mouv 1 c v;
4           mouv 1 b v;
5           mouv 3 c v;
6           mouv 3 b v;
7           mouv 3 c v;
8           mouv 3 a v;
9           mouv 1 c v;
10          mouv 1 a v
11      end
12
13 else begin mouv 3 c v;
14           mouv 3 b v;
15           mouv 1 c v;
16           mouv 1 b v;
17           mouv 1 c v;
18           mouv 1 a v;
19           mouv 3 c v;
20           mouv 3 a v
21     end
22 ;;
23
24 let manchot v a b = manchot_aux v a b "Y" oriente ;;
```

Cette macro est une macro de placement de pièce. Son nom vient de la petite histoire servant de moyen mnémotechnique pour la retenir dans les méthodes de résolution du cube.

Grâce aux deux macros précédentes, elle s’effectuera soit à gauche soit à droite afin de placer la pièce (“a”,“b”) à sa place définitive. (explication grâce au schéma à insérer !!)

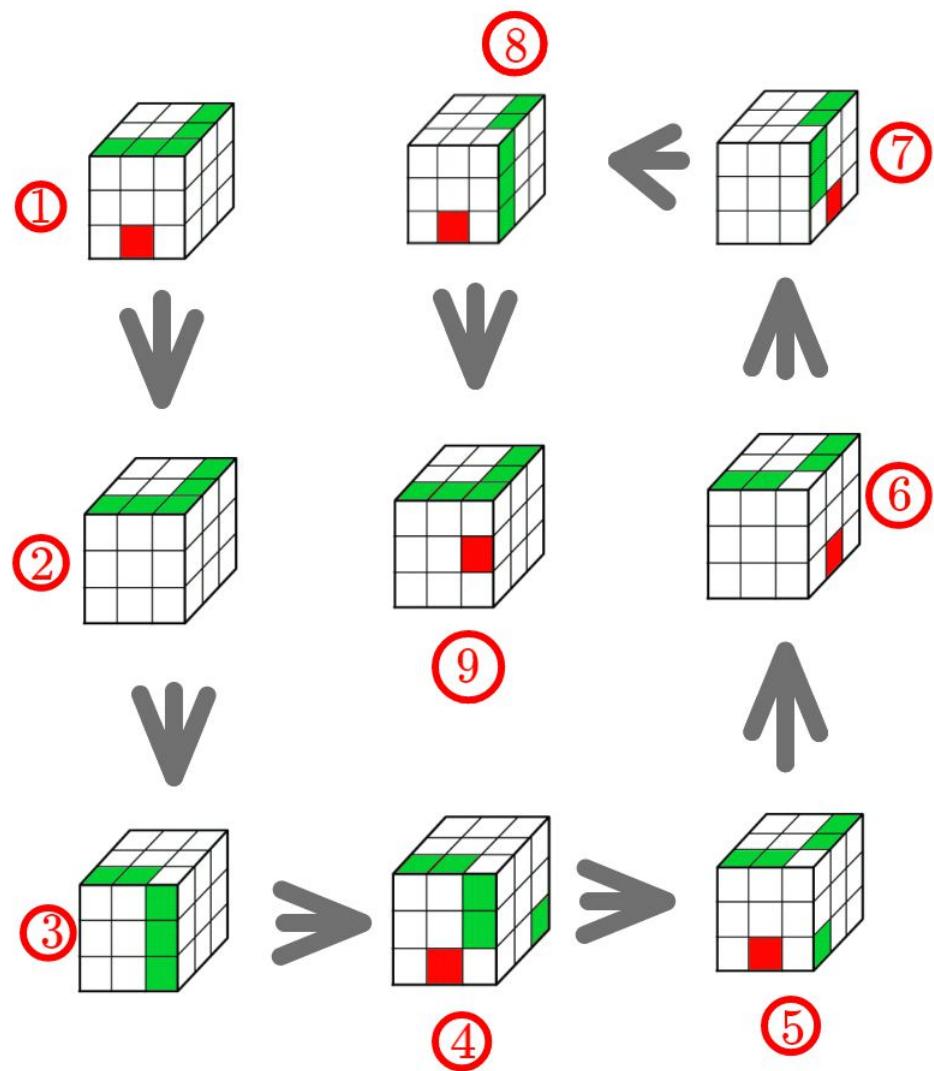


FIGURE 3.18: Description de la macro dite du manchot

#### 3.4.2.4 Macro “recherche\_centre”

```

1 let recherche_centre v_d = let i = ref 0 in
2 while (!i <= 3) do
3   match v_d.(extraction.(!i)) with
4   | ("Y",_) -> incr i
5   | (_, "Y") -> incr i
6   | ("R", "B") when (!i=0) -> incr i
7   | _ when (!i=0) -> i:=4
8   | ("R", "G") when (!i=1) -> incr i
9   | _ when (!i=1) -> i:=5
10  | ("B", "O") when (!i=2) -> incr i
11  | _ when (!i=2) -> i:=7
12  | ("G", "O") when (!i=3) -> incr i
13  | _ when (!i=3) -> i:=9
14  | _ -> failwith "pb recherche"
15 done;
16 !i
17 ;;

```

Lorsque la macro “manchot” ne peut plus s’appliquer, cela signifie qu’aucune pièce de la couronne inférieure ne peut être mise en place solution. Il y a dès lors deux possibilités :

- La couronne centrale est résolue.
- Des pièces de la couronnes centrale sont mal placées sur celle ci.

Dans le second cas, on applique la macro “recherche\_centre” qui renvoie la place d’une pièce, répondant à ces critères, dans le vecteur double. Puis en appliquant le manchot sur n’importe quelle pièce en considérant cette place comme position finale, on récupère une pièce, sur la couronne inférieure, pouvant être placée par le manchot.

Il suffit alors d’itérer jusqu’à ce que la couronne centrale soit résolue.

#### 3.4.3 Le programme Étape 3

Dans cette étape, on utilise toutes les macros définies précédemment afin de placer chaque arrête de la deuxième couronne.

```

1 let sortie_centre v n = match n with
2   | 4 -> manchot v "R" "B"
3   | 5 -> manchot v "R" "G"
4   | 7 -> manchot v "B" "O"
5   | 9 -> manchot v "G" "O"
6   | _ -> failwith "pb sortie centre"
7 ;;
8
9
10 let rota_y v = let nb_mouv = ref 0 in
11   while ( ((red v) = false) && ((green v) = false) && ((blue v)
12           = false) && ((orange v) = false) && (!nb_mouv<=3)) do
13     mouv 1 "Y" v; incr nb_mouv
14   done;
15 ;;

```

```
15
16 let place_manchot v = match 0 with
17   |_ when (red v)-> manchot v v.(10) v.(52)
18   |_ when (green v)-> manchot v v.(32) v.(50)
19   |_ when (blue v)-> manchot v v.(21) v.(48)
20   |_ when (orange v)-> manchot v v.(43) v.(46)
21   |_ -> sortie_centre v (recherche_centre (vect_double v))
22 ;;
23
24
25
26 let etape3 v =
27   while not ((rg v) && (go v) && (ob v) && (br v)) do
28     rota_y v;
29     place_manchot v
30   done
31;;
```

### 3.5 Étape 4

#### 3.5.1 Description du but de l'étape

L'étape 4 prend comme argument un cube avec la “deuxième couronne” et son but et de faire ce qu'on appelle la “croix jaune alignée” (voir 3.19).

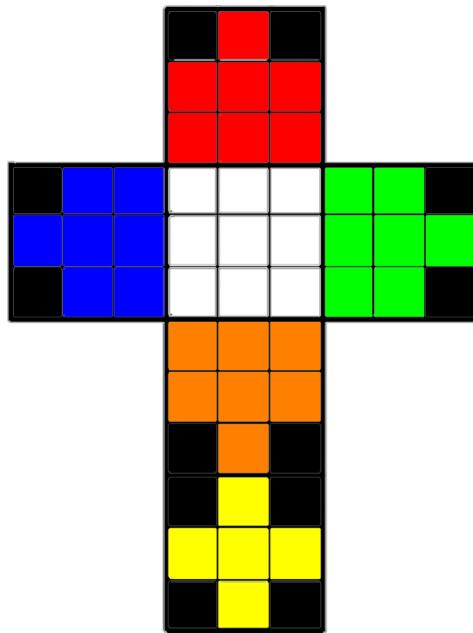


FIGURE 3.19: Représentation du patron du cube à la fin de l'étape 4

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs		R		R	R	R	R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs		B	B	B	B	B		B	B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G	G		G	G	G	G	G	
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O	O	O	O		O	
Indices	45	46	47	48	49	50	51	52	53
Valeurs		Y		Y	Y	Y		Y	

TABLE 3.5: Vecteur représentant le Kube en fin d'étape 4

### 3.5.2 Description des macros de l'étape 4

#### 3.5.2.1 Macro “croix\_jaune”

```

1 let croix_jaune_aux v a b c =
2   mouv 3 a v;
3   mouv 3 c v;
4   mouv 3 b v;
5   mouv 1 c v;
6   mouv 1 b v;
7   mouv 1 a v
8 ;;
9
10 let croix_jaune v a b = croix_jaune_aux v a b "Y";;

```

Cette macro permet de résoudre la croix jaune (figure 1.2). Elle permet, si le cube est en position 1 (figure 1.1) d'obtenir la croix jaune en une application, sinon si elle est dans la position 3 ou 4, une application permettra de faire appaitre la position 1 et donc d la croix jaune.

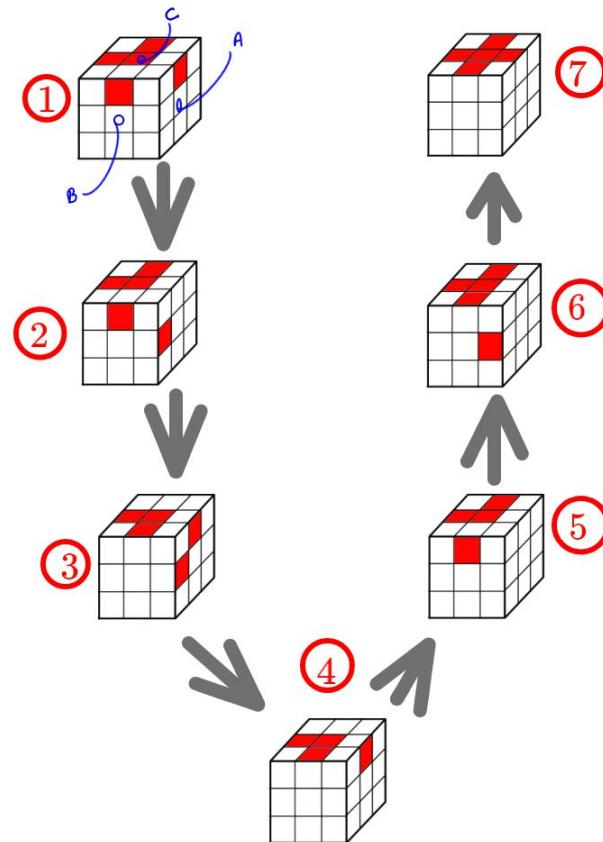


FIGURE 3.20: Description de la macro de résolution de la croix jaune

### 3. ALGORITHME

---

#### 3.5.2.2 Macro “chaise”

```
1 let chaise_aux v a b =
2   while ([|v.(10);v.(21);v.(32);v.(43)|] <> [|"R";"B";"G";"O"
|]) do
3     mouv 1 a v;
4     mouv 2 b v;
5     mouv 3 a v;
6     mouv 3 b v;
7     mouv 1 a v;
8     mouv 3 b v;
9     mouv 3 a v
10 done
11 ;;
12
13 let chaise v a = chaise_aux v a "Y" ;;
```

Cette macro, tirant son nom du moyen mnémotechnique permettant de la retenir, permet d’aligner la croix jaune avec la couronne centrale. Toute sa spécificité est qu’elle laisse l’une des arrêtes jaunes fixe et rotationne les trois autres dans le sens horaire.

#### 3.5.2.3 Fonctions de booléens de reconnaissance de valeur

```
1 let test_orange v =
2   if (v.(43)="0") then 1 else 0
3 ;;
4
5 let test_blue v =
6   if (v.(21)="B") then 1 else 0
7 ;;
8
9 let test_red v =
10  if (v.(10)="R") then 1 else 0
11 ;;
12
13 let test_green v =
14  if (v.(32)="G") then 1 else 0
15 ;;
```

Ces fonctions renvoient des booléens permettent simplement de dire si les vignettes centrales extérieures de la couronne jaune possèdent les bonnes valeurs.

Sur 3.22, la fonction “test\_green” renverrai “0” car la vignette est bleu et non verte.

#### 3.5.2.4 Fonction “somme\_test”

```
1 let somme_test v = (test_green v)
2           + (test_orange v)
3           + (test_blue v)
4           + (test_red v)
5 ;;
```

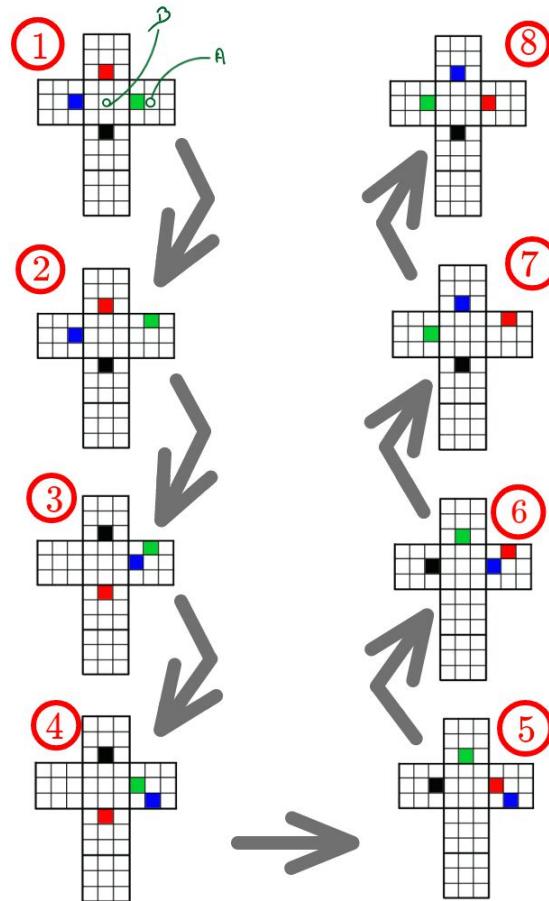


FIGURE 3.21: Description de la macro “chaise”

Cette petite fonction sert à renvoyer le nombre d’arêtes extérieures à la couronne jaune bien placées. Elle sert dans l’étape 5 car la macro “chaise” permet de placer les arêtes à condition qu’uniquement l’une d’entre elles soit déjà bien placée.

### 3.5.3 Le programme Étape 4

Dans cette étape, on utilise toutes les macros définies précédemment afin de réaliser la croix jaune alignée.

```

1 let etape_4_1 v =
2   while ([|v.(52);v.(48);v.(46);v.(50)|] <> [| "Y"; "Y"; "Y"; "Y" |])
3     do
4       match (v.(52),v.(48),v.(46),v.(50)) with
5         | ("Y","Y",_,_) -> croix_jaune v "O" "G"
6         | (_, "Y", "Y", _) -> croix_jaune v "G" "R"
7         | (_, _, "Y", "Y") -> croix_jaune v "R" "B"
8         | ("Y", _, _, "Y") -> croix_jaune v "B" "O"
9         | (_, _, _, _) -> croix_jaune v "O" "G"
9   done

```

### 3. ALGORITHME

---

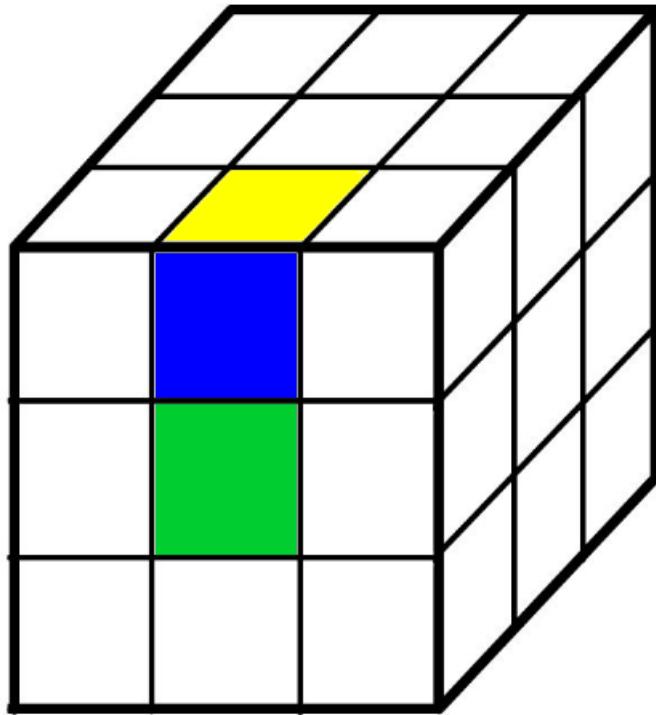


FIGURE 3.22: Description des fonctions de booléens

```
10 ;;
11
12
13
14 let test_orange v =
15   if (v.(43)="O") then 1 else 0
16 ;;
17
18 let test_blue v =
19   if (v.(21)="B") then 1 else 0
20 ;;
21
22 let test_red v =
23   if (v.(10)="R") then 1 else 0
24 ;;
25
26 let test_green v =
27   if (v.(32)="G") then 1 else 0
28 ;;
29
30
31 let somme_test v = (test_green v)
32           + (test_orange v)
```

```

33           + (test_blue v)
34           + (test_red v)
35 ;;
36
37 let etape_4_2 v = let compteur = ref 0 in
38   while ((somme_test v) <> 1) do
39     if (!compteur<=3) then begin mouv 1 "Y" v; incr compteur end
40     else begin
41       mouv 1 "R" v;
42       mouv 2 "Y" v;
43       mouv 3 "R" v;
44       mouv 3 "Y" v;
45       mouv 1 "R" v;
46       mouv 3 "Y" v;
47       mouv 3 "R" v;
48       compteur := 0
49     end
50   done
51 ;;
52
53 let etape_4_3 v = match ((test_orange v),
54                           (test_blue v),
55                           (test_red v),
56                           (test_green v)) with
57   |(1,0,0,0) -> chaise v "B"
58   |(0,1,0,0) -> chaise v "R"
59   |(0,0,1,0) -> chaise v "G"
60   |(0,0,0,1) -> chaise v "O"
61   |_ -> failwith "4_3"
62 ;;
63
64 let etape4 v =
65   etape_4_1 v;
66   etape_4_2 v;
67   etape_4_3 v
68 ;;

```

### 3.6 Étape 5

#### 3.6.1 Description du but de l'étape

L'étape 5 prend comme argument un cube avec la “croix jaune alignée” et son but et de finir la résolution du cube (voir 3.23).

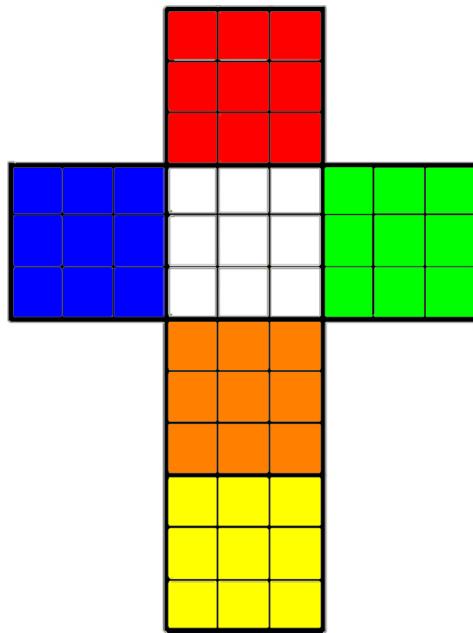


FIGURE 3.23: Représentation du patron du cube à la fin de l'étape 5

Indices	0	1	2	3	4	5	6	7	8
Valeurs	W	W	W	W	W	W	W	W	W
Indices	9	10	11	12	13	14	15	16	17
Valeurs	R	R	R	R	R	R	R	R	R
Indices	18	19	20	21	22	23	24	25	26
Valeurs	B	B	B	B	B	B	B	B	B
Indices	27	28	29	30	31	32	33	34	35
Valeurs	G	G	G	G	G	G	G	G	G
Indices	36	37	38	39	40	41	42	43	44
Valeurs	O	O	O	O	O	O	O	O	O
Indices	45	46	47	48	49	50	51	52	53
Valeurs	Y	Y	Y	Y	Y	Y	Y	Y	Y

TABLE 3.6: Vecteur représentant le Kube en fin d'étape 5

### 3.6.2 Description des macros de l'étape 5

#### 3.6.2.1 Macro “rotation\_cY”

```

1 let rotation_cY_aux v a b c =
2 mouv 1 a v;
3 mouv 3 b v;
4 mouv 1 c v;
5 mouv 3 a v;
6 mouv 3 c v;
7 mouv 1 b v;
8 mouv 1 c v;
9 mouv 1 a v;
10 mouv 3 c v;
11 mouv 3 a v
12 ;;
13
14 let rotation_cY v a b = rotation_cY_aux v a b "Y" ;;

```

Cette macro permet de conserver un coin de fixe et de faire tourner les trois autres dans le sens horaire. Elle permet donc dès qu'un des quatres coins de la face jaune est bien placé de placer les 3 autres en une ou deux applications.

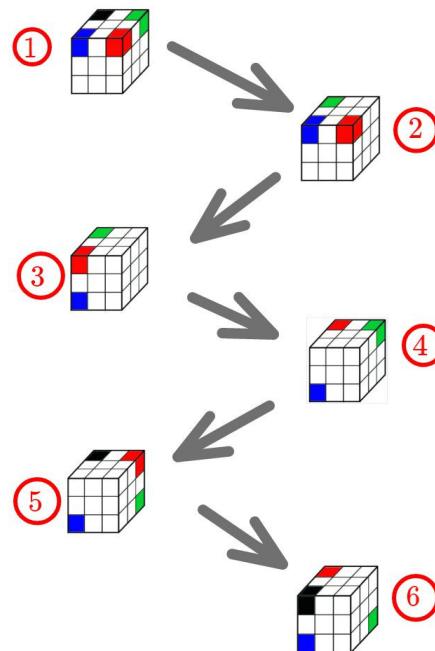


FIGURE 3.24: Description de la macro de rotation des coins jaunes 1

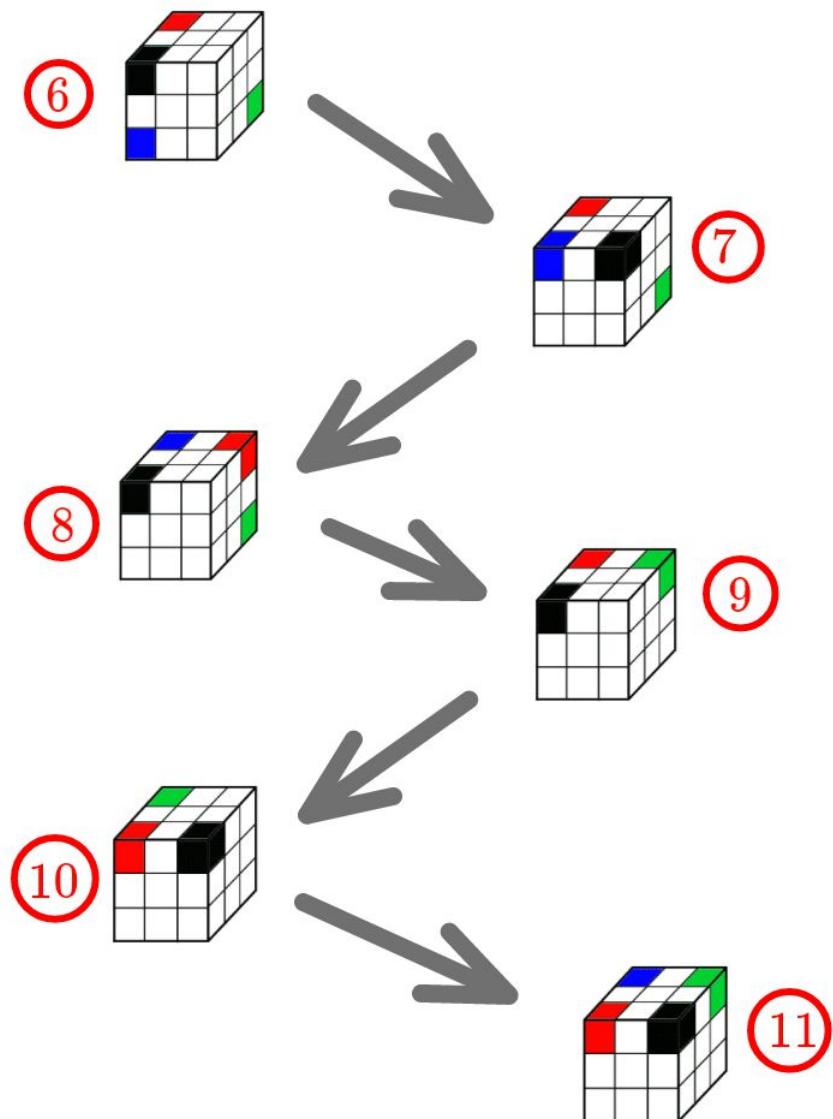


FIGURE 3.25: Description de la macro de rotation des coins jaunes 2

### 3.6.2.2 Macro “position\_coins\_bool”

```

1 let position_coins_bool_aux v a b bool =
2   let booleen = ref (bool (vect_triple v)) in
3     while !booleen do
4       mouv 3 a v;
5       mouv 3 b v;
6       mouv 1 a v;
7       mouv 1 b v;
8       booleen := (bool (vect_triple v))
9     done
10;;
11
12 let position_coins_bool v bool = position_coins_bool_aux v "R"
  "W" bool;;

```

Cette macro est la même que celle utilisé dans l'étape 2 mais cette fois on l'applique sur la face opposée. De même une boucle gérée par un booléen a été ajoutée afin d'appliquer cette fonction jusqu'à ce que le coin en question soit bien placé.

### 3.6.3 Le programme Étape 5

Dans cette étape, on utilise toutes les macros définies précédemment afin de finir la résolution du cube.

```

1 let etape_5_1 v =
2   while not ((rby (vect_triple v)) && (rgy (vect_triple v)) &&
3             (boy (vect_triple v)) && (goy (vect_triple v))) do
4     match 0 with
5       |_ when rby (vect_triple v) -> rotation_cY v "R" "O"
6       |_ when rgy (vect_triple v) -> rotation_cY v "G" "B"
7       |_ when boy (vect_triple v) -> rotation_cY v "B" "G"
8       |_ when goy (vect_triple v) -> rotation_cY v "O" "R"
9       |_ -> rotation_cY v "R" "O"
10    done
11
12 let etape_5_2 v =
13   position_coins_bool v (orient_rby); mouv 3 "Y" v;
14   position_coins_bool v (orient_boy); mouv 3 "Y" v;
15   position_coins_bool v (orient_ogy); mouv 3 "Y" v;
16   position_coins_bool v (orient_gry); mouv 3 "Y" v
17;;
18
19 let etape5 v =
20   etape_5_1 v;
21   etape_5_2 v
22;;

```



## 4.1 Idées d'améliorations

Une fois le programme fonctionnel, il nous est apparu que nous pouvions l'améliorer significativement sans pour autant en changer la structure.

### 4.1.1 Le filtre

```

1 let rec filtre l = match l with
2   | [] -> []
3   | a::b::c::d::suite when ((a=b) && (b=c) && (c=d)) -> filtre
      suite
4   | a::b::c::suite when ((a=b) && (b=c))
5     -> ("-"^a^";")::(filtre suite)
6   | a::b::suite when (a=b)-> ("2"^a^";")::(filtre suite)
7   | a::suite -> (a^";")::(filtre suite)
8 ;;

```

Comme nous avons décidé de décrire les 18 mouvements du Kube à partir de 6 générateurs, donc nous avons ensuite créer une fonction de filtre afin de rendre la liste des mouvements plus lisible et plus proche d'une résolution humaine du cube.

Nous avons donc considérer les modifications suivantes :

- 4 mouvements identiques à la suite représentent l'identité, le cube ne change pas au final,
- 3 mouvements identiques à la suite représentent un quart de tour en sens trigonométrique,
- 2 mouvements identiques à la suite représentent un demie tour.

### 4.1.2 Traitement exhaustif des mouvements

Nous avons eu une autre idée, mais qui s'est révélée inutile au final car notre but n'est pas de faire le programme le plus efficace possible mais simplement

un programme fonctionnel.

En effet à un certain moment nous voulions écrire les 18 mouvements et non pas seulement les 6 générateurs car cela permettait une économie en temps de calcul, cependant ce gain aurai été minime et grâce au filtre défini précédemment, la liste des mouvements est identique à celle que nous aurions obtenue si nous avions implémenter cette modification.

## 4.2 Traitement expérimental

Nous avons dans un dernier temps voulu tester notre programme sur un échantillon conséquent de Kube, cela pour deux raisons :

- Afin de nous assurer qu'il était réellement fonctionnel et que les résolutions précédentes n'étaient pas du à la chance et à des cas exotiques
- Mais aussi de pouvoir faire une moyenne du nombre de coups et du temps de résolution pour un Kube.

### 4.2.1 Fonction de mélange

Pour cela nous avons commencer par créer une fonction de mélange aléatoire du cube :

```

1 let melange v =
2   for i = 1 to 42 do
3     let temp = random_int 5 in
4       match temp with
5         |0 -> mouv 1 "W" v
6         |1 -> mouv 1 "R" v
7         |2 -> mouv 1 "B" v
8         |3 -> mouv 1 "G" v
9         |4 -> mouv 1 "O" v
10        |5 -> mouv 1 "Y" v
11        |_ -> failwith "probleme random_int"
12   done
13 ;;

```

Ensuite nous avons appliquer la fonction de mélange puis notre programme un million de fois afin d'avoir une moyenne représentative (notre programme résolvant le Kube, il n'était pas nécessaire de réinitialiser ce dernier à chaque boucle mais seulement de s'assurer qu'il était bien résolu).

### 4.2.2 Résultats expérimentaux

Grâce à ce test à grande échelle nous avons pu obtenir les résultats suivants :

- Une moyenne de 164 coups par Kube,  $\approx 40$  de plus que pour un humain avec la même méthode et 144 de plus que le chiffre de Dieu : 20 (voir annexe 4)
- Une moyenne de  $3,325 \cdot 10^{-3}$  seconde par résolution, cela est bien en dessous des records de monde de résolution humaine et mécanique mais ce n'est pas un miracle, loin de la car ce chiffre ne représente que la résolution informatique et non pas physique du cube.

### 4.3 Gains personnels

Ce TIPE nous a beaucoup apporté, en effet par ce projet nous avons pu mieux comprendre les enjeux du métier d'ingénieur, comme le travail d'équipe ou encore le fait de mener à bien un projet complet du début jusqu'à la fin.

Alors que dans les premiers temps, réaliser un algorithme de résolution du Kube nous paraissait très compliqué voir impossible, en travaillant à deux, nous avons pu dépasser ces difficultés en voyant que la plupart du temps ce qui était problématique pour l'un de nous ne l'était pas pour l'autre et inversement et en confrontant toujours nos idées afin de trouver le meilleur moyen de faire chaque partie de ce programme.

De plus cela nous permis une bien meilleur compréhension de l'algorithme et du langage Caml en particulier.



## A.1 Biographie

Ernő Rubik naît pendant la guerre à Budapest en Hongrie, d'un père ingénieur-mécanicien constructeur d'avions planeurs et d'une mère poète. Il sort diplômé en architecture de l'Université polytechnique et économique de Budapest (Budapesti Műszaki Egyetem) en 1967, puis entame des études à l'École supérieure hongroise des arts appliqués (Magyar Iparművészeti Főiskola). De 1971 à 1975, il travaille en tant qu'architecte, puis devient professeur à l'École supérieure hongroise des arts appliqués.

Il se marie en 1977 avec une architecte d'intérieur. En 1978 naît de leur relation une petite fille : Anne. En 1974, il invente un casse-tête bientôt connu sous le nom de Rubik's Cube. Celui-ci connaît une explosion de ses ventes au tout début des années 1980, quand un partenariat de distribution est conclu avec la firme internationale Ideal Toys. Le Rubik's Cube envahit alors le monde entier, et reste aujourd'hui comme un symbole des années 1980, souvent utilisé par la publicité ou le marketing comme icône représentative de la décennie.

Des concours sont organisés dans le monde entier depuis plus de 30 ans pour battre le record du monde de rapidité. Le Rubik's Cube est sans conteste le casse-tête moderne le plus connu au monde. Il connaît de nombreuses déclinaisons (tailles, couleurs, formes, etc.).

Au début des années 1980, il devient éditeur d'un journal sur les jeux et les puzzles appelé ...Et jeux (...És játék) puis devient son propre employeur en 1983, en fondant le Rubik Stúdió, où il conçoit des meubles et des jeux.

En 2007, Ernő Rubik reçoit le Prix Kossuth des mains du Président de la République hongroise, pour la création du Rubik's Cube.

## A.2 Une interview intéressante

" Je suis né en 1944 à Budapest, en Hongrie. Mon père était ingénieur-mécanicien, constructeur d'avions planeurs, un spécialiste renommé, créateur

de plus de vingt-six types de planeurs. Ma mère était femme de lettres, poétesse et artiste. La présence conjuguée de ces impulsions, celle de la technique et celle des arts, a été pour moi, j'en suis persuadé, un facteur déterminant. Au début, j'inclinais vers les Beaux-Arts : j'ai beaucoup dessiné et beaucoup peint. J'ai fait mes études secondaires dans une école dépendant des Beaux- Arts, comme sculpteur. Dès cette époque, mon goût pour la technique s'éveilla. Aussi l'étape suivante de mes études devenait, tout naturellement, l'Université de l'Enseignement technique de Budapest, et, en 1967, j'y obtins un diplôme d'architecte. L'architecture me passionne toujours, comme une des activités les plus complexes qui réunit en son sein les traits caractéristiques de la science, de la technique et des arts. Diplôme en main, je ne me suis pas encore senti un homme complètement formé et j'ai continué mes études à l'Ecole supérieure des Arts décoratifs dans la section architecture intérieure. Mon second diplôme me conférait le titre de "designer" ; il me fut remis en 1970. Ces études m'ont sensibilisé aux facteurs artistiques. Depuis 1970 je suis resté constamment à l'Ecole supérieure enseignant plans et constructions, dessins d'architecture intérieure, plans et projets de meubles, études de forme et géométrie descriptive. L'enseignement est le meilleur moyen d'apprendre, j'en suis toujours convaincu ; en communiquant nos connaissances nous continuons à découvrir et à apprendre. En outre, cette activité nous oblige chaque fois à une nouvelle formulation de ce que nous désirons exprimer, nous force à de nouveaux essais, à la recherche constante de nouvelles méthodes. Les liens permanents avec la jeunesse nous aident à rester jeunes d'esprit, nous rendent capables de nous étonner constamment. (...) Je me suis marié en 1977 ; ma femme est architecte d'intérieur. Notre petite fille, née en 1978, s'appelle Anne.

(...)

L'espace m'a toujours intrigué, avec ses possibilités d'une richesse extraordinaire, la transformation de l'espace par des objets (l'architecture), la transformation des objets dans l'espace (la sculpture, le design), le mouvement dans l'espace et dans le temps, leur corrélation, leur répercussion sur l'homme, la relation de l'homme à l'espace, à l'objet et au temps. Je pense que le CUBE est né de cet intérêt, de cette recherche d'expression et de l'acuité toujours accrue de ces réflexions. (...) J'aime jouer, je l'avoue, j'aime surtout les jeux où le partenaire, l'adversaire véritable est la nature elle-même, avec ses mystères bien particuliers mais déchiffrables. Le jeu le plus passionnant pour moi est le jeu avec l'espace, la recherche des formes possibles de l'espace, c'est-à-dire la construction logique et concrète d'ordonnancements divers.

On ne peut évidemment pas préciser à la minute près la date de naissance d'une idée ; cela me paraît impossible, à moi surtout, pour qui le temps, de ce point de vue, n'a que peu d'intérêt. Ce pouvait être au printemps 1974 que l'idée fondamentale surgit en moi, comme une possibilité digne d'attention. Je suis d'un caractère attaché aux expériences, ainsi, dès le début, j'ai étudié les variations d'un cube de type 2x2x2. Je fus immédiatement saisi par la richesse que l'on pouvait pressentir dès ce début. La solution technique finale, qui est la plus simple dans sa forme 3x3x3, la plus commodément réalisable en modèles, après quelques essais, s'est imposée à moi vers la fin de l'automne 1974. Quelques modèles prêts à fonctionner furent ainsi réalisés ; pour moi et mes

amis, ce fut passionnant de jouer avec eux pour la première fois. Nous fûmes tout surpris de découvrir progressivement que nous avions réalisé quelque chose d'original, de neuf. Comme la question du brevet d'invention fut immédiatement soulevée, je mis en marche le processus nécessaire le 30 janvier 1975. Presque en même temps, pressentant quelque chose de l'importance du jeu inventé, de ses possibilités et de sa valeur réelle, je me suis mis à la recherche d'un associé pour la fabrication industrielle et par une chance extraordinaire, j'en ai effectivement trouvé un. La suite est relativement simple : après son apparition sur le marché (1977) le jeu est devenu, rapidement et comme par enchantement, fort populaire en Hongrie, puis, dès 1980 dans le monde entier. J'ai l'impression que l'histoire n'en est qu'à ses débuts, et que l'on ne peut prévoir la fin, de même que personne, je pense, n'aurait pu deviner son avenir."

*Londres, le 31 janvier 1981,  
Ernö RUBIK*



## B.1 Notre modèle et motivation : le projet Cubestormer

Le projet Cubestormer est mené par David Gilday et Mike Dobson. Il vise à inventer et réaliser un robot capable de résoudre le Rubik's Cube.



FIGURE B.1: David Gilday et Mike Dobson à la remise du record du monde du Cubestormer 3

## B. CUBESTORMER

---

En 2010, ils réussirent à créer Cubestormer, un robot capable de résoudre physiquement le Kube en un temps raisonnable (12 secondes).

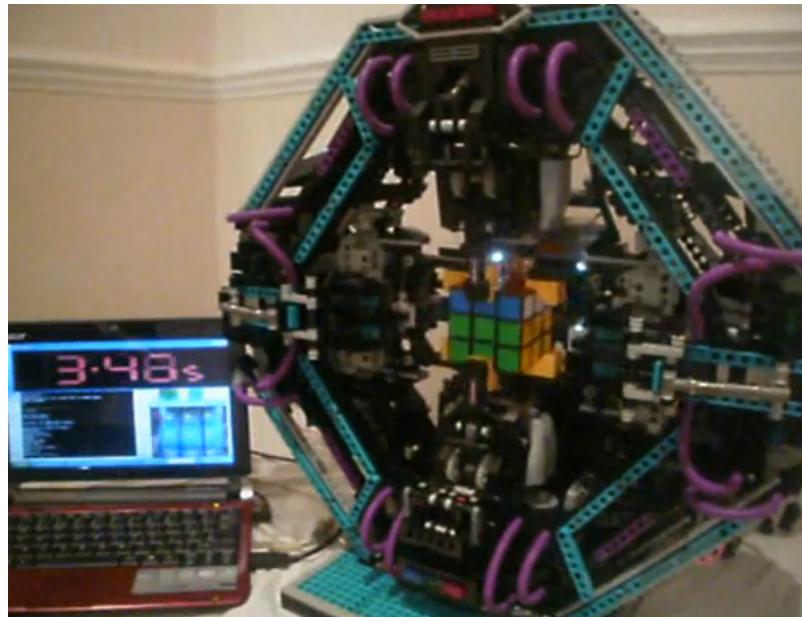


FIGURE B.2: Le Cubestormer 1

Mais l'équipe ne s'arrêta pas là, le 11 Novembre 2011, Cubestormer 2 vit le jour et parvint à battre le record du monde de résolution par un robot (5.270 secondes).



FIGURE B.3: Le Cubestormer 2

---

### B.1. Notre modèle et motivation : le projet Cubestormer

Cependant cela n'était toujours pas assez pour eux et le 15 Mars 2014, ils présentèrent le Cubestormer 3 qui leur permit de battre leur propre record du monde (3,253 secondes).

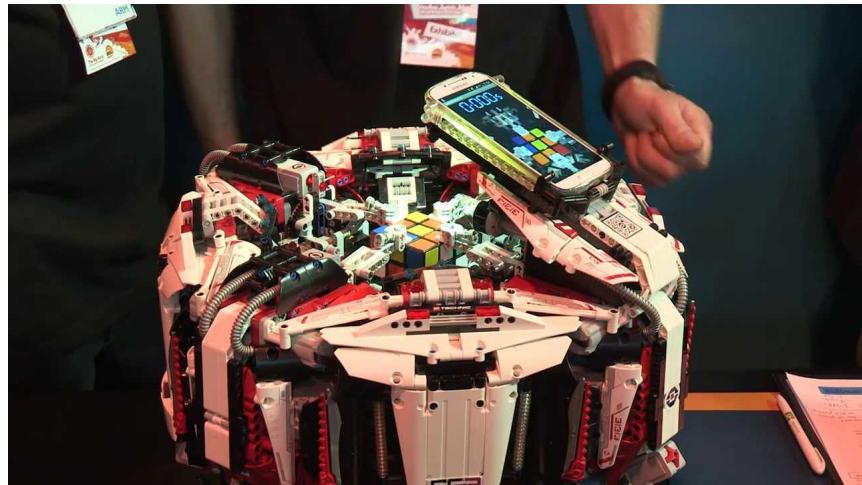


FIGURE B.4: Le Cubestormer 3



### C.1 La méthode “Layer by Layer” :

C'est la plus intuitive et la plus simple à mettre en œuvre. La résolution nécessite en moyenne un peu plus de 110 mouvements :

1. Réaliser une face, par exemple la face supérieure blanche, en prenant bien soin de placer correctement la couronne (placer les cubes entourant cette face) et les cubes centraux (bleu, orange, vert et rouge),
2. puis la deuxième couronne (la rangée horizontale à mi-hauteur),
3. déplacer les cubes-arête de la face du bas à leur place et les orienter correctement,
4. déplacer les cubes-sommet à leur place,
5. enfin les orienter.

Chaque opération (tourner une arête ou un sommet, échanger deux arêtes ou deux sommets) pourra être réalisée deux fois, après avoir placé les cubes concernés sur la même face, et en prenant soin de ne pas modifier cette face pendant l'opération. La première exécution mélange le reste du cube, mais en tournant alors la face d'un quart ou d'un demi-tour pour placer le(s) sujet(s) de la deuxième opération au même endroit relativement au reste du cube et en refaisant l'opération à l'envers, on réalisera la deuxième opération tout en remettant le reste du cube en place.

### C.2 La méthode Sandwich

Une autre méthode intuitive :

1. Réaliser une face, par exemple la face rouge.
2. Réaliser la face opposée à celle déjà correcte (ici la face orange), pour cela il faut d'abord placer correctement tous les coins, puis les orienter correctement, et enfin mettre les arêtes.

3. Par échanges, amener chaque arête restante à sa place (à ce stade il ne reste plus que 4 arêtes à placer).
4. Enfin placer, puis orienter ces 4 arêtes correctement.

### C.3 Méthode de Lars Petrus

C'est une approche différente des deux premières : elle est moins automatisée, mais a l'avantage de conserver au maximum les cubes bien placés. La résolution nécessite en moyenne 60 mouvements :

1. Réaliser un « petit cube » de dimensions  $2 \times 2 \times 2$  (constitué de 3 couleurs).
2. Étendre ce « petit cube » à un parallélépipède  $2 \times 2 \times 3$  (constitué de 4 couleurs), sans jamais détruire le « petit cube ».
3. Orienter les arêtes restantes, de façon à pouvoir les placer orientées correctement en utilisant deux faces.
4. Étendre l'objet  $2 \times 2 \times 3$  à un objet  $2 \times 3 \times 3$  (c'est-à-dire deux couches du cube complet), sans jamais détruire ce qui a été fait auparavant.
5. Placer et orienter les 4 coins restants.
6. Et enfin, placer les 4 arêtes restantes.

### C.4 Méthode de Jessica Fridrich (ou CFOP)

C'est encore une approche différente qui, comme celle de L. Petrus, nécessite environ 60 mouvements. Cette méthode est très utilisée en speedcubing car systématique :

1. Réaliser une croix sur une face.
2. Réaliser les F2L c'est-à-dire de placer les coins de la face blanche en même temps que la deuxième couronne.
3. Réaliser l'OLL (orientate last layer), c'est-à-dire orienter les cubes de la dernière face.
4. Réaliser la PLL (permute last layer), c'est-à-dire replacer les cubes de la dernière face.

Cette méthode est utilisée par les plus grands champions mais nécessite l'apprentissage de nombreuses séquences :

- 42 pour les F2L (les F2L ne nécessitent cependant pas d'être appris par cœur, ils peuvent être effectués de manière intuitive)
- 57 pour l'OLL
- 21 pour la PLL

Des méthodes alternatives permettent d'apprendre moins de séquences, comme l'OLL ou la PLL en deux étapes.

## C.5 Méthodes corners first (Guimond, Ortega, Waterman)

Une approche encore différente et assez intuitive consiste à commencer par les coins ; l'avantage d'une telle méthode est qu'il est ensuite facile de résoudre les arêtes en gardant les coins bien placés. Ces méthodes étaient très utilisées dans les années 1980. Elles sont devenues plus rares aujourd'hui. La résolution nécessite 60 à 70 mouvements (une cinquantaine seulement si on compte un mouvement de tranche centrale comme un seul mouvement et non deux) :

1. Placer et orienter les coins (plusieurs approches sont possibles pour cela).
2. Placer et orienter les arêtes de deux couronnes opposées.
3. Résoudre la couche intermédiaire.

## C.6 Méthode Roux

Basée sur la construction de bloc, cette méthode nécessite moins de coups. Elle est donc très adaptée pour le speedcubing. Elle porte le nom de son inventeur<sup>6</sup>. Elle consiste à réaliser en premier deux faces opposées puis terminer par la tranche centrale.

1. construire 1 bloc  $1 \times 2 \times 3$
2. construire le bloc opposé  $1 \times 2 \times 3$
3. placer et orienter les derniers coins (PLL)
4. résoudre les deux arêtes latérales (6 coups maximum)
5. résoudre la tranche centrale (4 coups maximum)

## C.7 Les Cubes $4 \times 4 \times 4$ , $5 \times 5 \times 5$ , etc

Pour un supercube, la méthode la plus simple (et la plus longue) reprend quelques algorithmes de la méthode couche par couche pour un  $3 \times 3 \times 3$  :

1. Former la face inférieure et sa couronne
2. Placer les coins de la couche supérieure
3. Placer les arêtes de la face supérieure
4. Placer les arêtes intermédiaires sur leurs couches respectives
5. Orienter correctement les arêtes intermédiaires
6. Placer les centres.

On peut facilement placer les arêtes de la deuxième couche entre les étapes 1 et 2. Cela permet d'éviter l'étape 4 pour les cubes de  $4 \times 4 \times 4$  et  $5 \times 5 \times 5$ .

Pour résoudre un  $3 \times 3 \times 3$  en couche par couche, on doit apprendre 5 algorithmes au minimum. Pour résoudre n'importe quel "supercube" (même un  $500 \times 500 \times 500$ ), 2 algorithmes supplémentaires sont nécessaires.

Une méthode plus efficace consiste à :

## C. LES MÉTHODES DE RÉSOLUTIONS

---

1. Placer les centres
2. Placer toutes les arêtes de même couleur ensemble (possible seulement si on les oriente dans un même sens)
3. Résoudre le cube comme s'il s'agissait d'un  $3 \times 3 \times 3$ .

Il est fréquent de finir l'étape 3 avec une seule rangée d'arête mal orientée (ce qui ne peut pas arriver sur un  $3 \times 3 \times 3$ ). Une quatrième étape peut parfois consister à orienter cette rangée d'arête.

Une variante de cette méthode, la méthode Yau, est très utilisée en speed-cubing pour le  $4 \times 4 \times 4$  :

1. Résoudre deux centres opposés
2. Résoudre trois arêtes de la couleur de l'un des deux centres
3. Terminer les quatre derniers centres en se servant de l'arête libre
4. Résoudre la quatrième arête et ainsi terminer la croix
5. Mettre toutes les arêtes en paires
6. Terminer le cube comme un  $3 \times 3 \times 3$  en méthode de Fridrich, en sachant que la croix a déjà été faite.

On peut, grâce à l'informatique, aller jusqu'à des cubes  $1000 \times 1000 \times 1000$ .

## D.1 Les règles du Speedcubing

Le speedcubing consiste à résoudre un Rubik's Cube le plus rapidement possible. Il existe à travers le monde de nombreux adeptes de cette activité, des compétitions, des prix à remporter.

Les premiers championnats ont commencé dans les années 1980 (précisément en 1982 à Budapest) avec un nombre restreint d'adeptes, mais la libre circulation des informations sur Internet a permis d'augmenter considérablement le nombre de speedcubers et les performances des compétiteurs ces dernières années.

Comme expliqué précédemment, on va chercher à résoudre un Rubik's Cube le plus rapidement possible. Un ensemble de règles strictes doit être respecté pour que le résultat soit valide[1].

La résolution se passe comme ceci :

- Le compétiteur a 15 secondes pour regarder son cube mélangé. Un juge devra enlever le "cache-cube", afin que le compétiteur puisse saisir le cube et l'inspecter. À la fin de ces 15 secondes, il doit poser le cube sur la table. Ce temps est appelé préinspection.
- Le compétiteur pose ses mains sur le chronomètre, démarre le chronomètre en décollant ses mains de ce dernier, et commence à résoudre le cube.
- Le chronomètre s'arrête une fois que le compétiteur a reposé le cube sur la table et a posé ses deux mains à plat devant lui, sur le chronomètre. Le juge décide alors de valider ou non sa résolution.

Le classement des compétiteurs se fait de la façon suivante : le compétiteur résout 5 cubes, le meilleur et le plus mauvais temps sont retirés, et la moyenne des 3 restants est conservée.

## D.2 Compétition

Conformément à la World Cube Association, les compétiteurs doivent résoudre un cube mélangé suivant la même séquence (chaque compétiteur a le même cube mélangé). Le chronomètre officiel utilisé en compétition est le Stackmat timer. Ce chrono possède deux surfaces sensibles au toucher sur lesquelles la personne pose ses mains. Lorsqu'au moins une des deux mains est soulevé, le chronomètre se déclenche. Le temps est arrêté quand les deux mains reviennent en place après la résolution. Un juge est présent, il s'assure du bon suivi des règles par les compétiteurs et prend note du temps.

## E.1 Les mouvements

```

1 let vect_double v =
2 [| (v.(1),v.(16)) ; (v.(3),v.(23)) ; (v.(5),v.(30)) ;
3   (v.(7),v.(37)) ; (v.(12),v.(19)) ; (v.(14),v.(28)) ;
4   (v.(10),v.(52)) ; (v.(25),v.(39)) ; (v.(21),v.(48)) ;
5   (v.(34),v.(41)) ; (v.(32),v.(50)) ; (v.(43),v.(46)) |]
6 ;;
7
8 let vect_triple v =
9 [| (v.(0),v.(15),v.(20)) ; (v.(2),v.(17),v.(27)) ;
10   (v.(6),v.(26),v.(36)) ; (v.(8),v.(33),v.(38)) ;
11   (v.(9),v.(18),v.(51)) ; (v.(11),v.(29),v.(53)) ;
12   (v.(24),v.(42),v.(45)) ; (v.(35),v.(44),v.(47)) |]
13 ;;
14
15
16 let solution = ref [] ;;
17
18
19 let coins_fW v =
20   v.(4) <- v.(0);
21   v.(0) <- v.(6);
22   v.(6) <- v.(8);
23   v.(8) <- v.(2);
24   v.(2) <- v.(4)
25 ;;
26
27
28
29 let aretes_fW v =
30   v.(4) <- v.(1);
31   v.(1) <- v.(3);
32   v.(3) <- v.(7);

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```
33 v.(7) <- v.(5);
34 v.(5) <- v.(4)
35 ;;
36
37
38
39 let coins_a1W v =
40   v.(4) <- v.(15);
41   v.(15) <- v.(26);
42   v.(26) <- v.(38);
43   v.(38) <- v.(27);
44   v.(27) <- v.(4)
45 ;;
46
47
48 let coins_a2W v =
49   v.(4) <- v.(17);
50   v.(17) <- v.(20);
51   v.(20) <- v.(36);
52   v.(36) <- v.(33);
53   v.(33) <- v.(4)
54 ;;
55
56
57 let aretes_aW v =
58   v.(4) <- v.(16);
59   v.(16) <- v.(23);
60   v.(23) <- v.(37);
61   v.(37) <- v.(30);
62   v.(30) <- v.(4)
63 ;;
64
65
66 let w v =
67   coins_fW v ;
68   aretes_fW v ;
69   coins_a1W v ;
70   coins_a2W v ;
71   aretes_aW v ;
72   v.(4) <- "W"
73 ;;
74
75
76 let coins_fR v =
77   v.(13) <- v.(9);
78   v.(9) <- v.(15);
79   v.(15) <- v.(17);
80   v.(17) <- v.(11);
81   v.(11) <- v.(13)
82 ;;
83
84 let aretes_fR v =
85   v.(13) <- v.(10);
```

```

86 v.(10) <- v.(12);
87 v.(12) <- v.(16);
88 v.(16) <- v.(14);
89 v.(14) <- v.(13)
90 ;;
91
92 let coins_a1R v =
93 v.(13) <- v.(51);
94 v.(51) <- v.(20);
95 v.(20) <- v.(2);
96 v.(2) <- v.(29);
97 v.(29) <- v.(13)
98 ;;
99
100 let coins_a2R v =
101 v.(13) <- v.(53);
102 v.(53) <- v.(18);
103 v.(18) <- v.(0);
104 v.(0) <- v.(27);
105 v.(27) <- v.(13)
106 ;;
107
108 let aretes_aR v =
109 v.(13) <- v.(52);
110 v.(52) <- v.(19);
111 v.(19) <- v.(1);
112 v.(1) <- v.(28);
113 v.(28) <- v.(13)
114 ;;
115
116 let r v =
117 coins_fR v;
118 aretes_fR v;
119 coins_a1R v;
120 coins_a2R v;
121 aretes_aR v;
122 v.(13) <- "R"
123 ;;
124
125
126
127 let coins_fB v =
128 v.(22) <- v.(18);
129 v.(18) <- v.(24);
130 v.(24) <- v.(26);
131 v.(26) <- v.(20);
132 v.(20) <- v.(22)
133 ;;
134
135 let aretes_fB v =
136 v.(22) <- v.(19);
137 v.(19) <- v.(21);
138 v.(21) <- v.(25);

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```
139 v.(25) <- v.(23);
140 v.(23) <- v.(22)
141 ;;
142
143 let coins_a1B v =
144 v.(22) <- v.(9);
145 v.(9) <- v.(45);
146 v.(45) <- v.(36);
147 v.(36) <- v.(0);
148 v.(0) <- v.(22)
149 ;;
150
151 let coins_a2B v =
152 v.(22) <- v.(15);
153 v.(15) <- v.(51);
154 v.(51) <- v.(42);
155 v.(42) <- v.(6);
156 v.(6) <- v.(22)
157 ;;
158
159 let aretes_aB v =
160 v.(22) <- v.(12);
161 v.(12) <- v.(48);
162 v.(48) <- v.(39);
163 v.(39) <- v.(3);
164 v.(3) <- v.(22)
165 ;;
166
167 let b v =
168 coins_fB v;
169 aretes_fB v;
170 coins_a1B v;
171 coins_a2B v;
172 aretes_aB v;
173 v.(22) <- "B"
174 ;;
175
176
177 let coins_fG v =
178 v.(31) <- v.(27);
179 v.(27) <- v.(33);
180 v.(33) <- v.(35);
181 v.(35) <- v.(29);
182 v.(29) <- v.(31)
183 ;;
184
185 let aretes_fG v =
186 v.(31) <- v.(28);
187 v.(28) <- v.(30);
188 v.(30) <- v.(34);
189 v.(34) <- v.(32);
190 v.(32) <- v.(31)
191 ;;
```

```

192
193 let coins_a1G v =
194 v.(31) <- v.(17);
195 v.(17) <- v.(8);
196 v.(8) <- v.(44);
197 v.(44) <- v.(53);
198 v.(53) <- v.(31)
199 ;;
200
201 let coins_a2G v =
202 v.(31) <- v.(11);
203 v.(11) <- v.(2);
204 v.(2) <- v.(38);
205 v.(38) <- v.(47);
206 v.(47) <- v.(31)
207 ;;
208
209 let aretes_aG v =
210 v.(31) <- v.(14);
211 v.(14) <- v.(5);
212 v.(5) <- v.(41);
213 v.(41) <- v.(50);
214 v.(50) <- v.(31)
215 ;;
216
217 let g v =
218 coins_fG v;
219 aretes_fG v;
220 coins_a1G v;
221 coins_a2G v;
222 aretes_aG v;
223 v.(31) <- "G"
224 ;;
225
226
227 let coins_f0 v =
228 v.(40) <- v.(36);
229 v.(36) <- v.(42);
230 v.(42) <- v.(44);
231 v.(44) <- v.(38);
232 v.(38) <- v.(40)
233 ;;
234
235 let aretes_f0 v =
236 v.(40) <- v.(37);
237 v.(37) <- v.(39);
238 v.(39) <- v.(43);
239 v.(43) <- v.(41);
240 v.(41) <- v.(40)
241 ;;
242
243 let coins_a10 v =
244 v.(40) <- v.(6);

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```
245 v.(6) <- v.(24);
246 v.(24) <- v.(47);
247 v.(47) <- v.(33);
248 v.(33) <- v.(40)
249 ;;
250
251 let coins_a20 v =
252 v.(40) <- v.(8);
253 v.(8) <- v.(26);
254 v.(26) <- v.(45);
255 v.(45) <- v.(35);
256 v.(35) <- v.(40)
257 ;;
258
259 let aretes_a0 v =
260 v.(40) <- v.(7);
261 v.(7) <- v.(25);
262 v.(25) <- v.(46);
263 v.(46) <- v.(34);
264 v.(34) <- v.(40)
265 ;;
266
267 let o v =
268 coins_f0 v;
269 aretes_f0 v;
270 coins_a10 v;
271 coins_a20 v;
272 aretes_a0 v;
273 v.(40) <- "0"
274 ;;
275
276
277 let coins_fY v =
278 v.(49) <- v.(45);
279 v.(45) <- v.(51);
280 v.(51) <- v.(53);
281 v.(53) <- v.(47);
282 v.(47) <- v.(49)
283 ;;
284
285 let aretes_fY v =
286 v.(49) <- v.(46);
287 v.(46) <- v.(48);
288 v.(48) <- v.(52);
289 v.(52) <- v.(50);
290 v.(50) <- v.(49)
291 ;;
292
293 let coins_a1Y v =
294 v.(49) <- v.(42);
295 v.(42) <- v.(18);
296 v.(18) <- v.(11);
297 v.(11) <- v.(35);
```

```

298 v.(35) <- v.(49)
299 ;;
300
301 let coins_a2Y v =
302 v.(49) <- v.(44);
303 v.(44) <- v.(24);
304 v.(24) <- v.(9);
305 v.(9) <- v.(29);
306 v.(29) <- v.(49)
307 ;;
308
309 let aretes_aY v =
310 v.(49) <- v.(43);
311 v.(43) <- v.(21);
312 v.(21) <- v.(10);
313 v.(10) <- v.(32);
314 v.(32) <- v.(49)
315 ;;
316
317 let y v =
318 coins_fY v;
319 aretes_fY v;
320 coins_a1Y v;
321 coins_a2Y v;
322 aretes_aY v;
323 v.(49) <- "Y"
324 ;;
325
326
327 let mouv_aux x a l v =
328 for i = 1 to x do
329 match a with
330 | "W" -> w v; l := "W"::!l
331 | "R" -> r v; l := "R"::!l
332 | "B" -> b v; l := "B"::!l
333 | "G" -> g v; l := "G"::!l
334 | "O" -> o v; l := "O"::!l
335 | "Y" -> y v; l := "Y"::!l
336 | _ -> failwith "Corrupted Data"
337 done;;
338
339 let mouv x a v = mouv_aux x a solution v;;

```

## E.2 Les booléens

```

1 let orange v = (v.(43)="O") && (v.(46)<>"Y");;
2 let blue v = ((v.(21)="B") && (v.(48)<>"Y"));;
3 let green v = ((v.(32)="G") && (v.(50)<>"Y"));;
4 let red v = ((v.(10)="R") && (v.(52)<>"Y"));;
5
6 let rg v = ((v.(14)="R") && (v.(28)="G"));;
7 let go v = ((v.(34)="G") && (v.(41)="O"));;
8 let ob v = ((v.(39)="O") && (v.(25)="B"));;

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```

9 let br v = ((v.(19)="B") && (v.(12)="R"));;
10
11 let rby v_t = ((v_t.(4)=("R","B","Y")) || (v_t.(4)=("R","Y","B"))
12           )) || (v_t.(4)=("B","Y","R")) || (v_t.(4)=("B","R","Y"
13           ))
14 || (v_t.(4)=("Y","R","B")) || (v_t.(4)=("Y","B","R"
15           )));;
14 let rgy v_t = ((v_t.(5)=("R","G","Y")) || (v_t.(5)=("R","Y","G"
16           ))
17           || (v_t.(5)=("G","R","Y")) || (v_t.(5)=("G","Y","R"
18           ))
19           || (v_t.(5)=("Y","R","G")) || (v_t.(5)=("Y","G","R"
19           )));;
17 let boy v_t = ((v_t.(6)=("B","O","Y")) || (v_t.(6)=("B","Y","O"
20           ))
21           || (v_t.(6)=("O","B","Y")) || (v_t.(6)=("O","Y","B"
22           ))
23           || (v_t.(6)=("Y","O","B")) || (v_t.(6)=("Y","B","O"
22           )));;
23
24 let oriente_rby v_t = (v_t.(4)<>("R","B","Y"));;
25 let oriente_boy v_t = (v_t.(4)<>("B","O","Y"));;
26 let oriente_ogy v_t = (v_t.(4)<>("O","G","Y"));;
27 let oriente_gry v_t = (v_t.(4)<>("G","R","Y"));;
```

### E.3 Les macros

```

1 #open "mouv";;
2 #open "booleen";;
3
4 (* etape 1 *)
5
6 let scan_double_aux v_d a b =
7   let position = ref 0 in
8     while ((v_d.(!position) <> (a,b))
9           && (v_d.(!position) <> (b,a))) do
10       incr position done;
11     !position
12 ;;
13
14 let scan_double v a b =
15   scan_double_aux (vect_double v) a b
16 ;;
17
18
19 (* etape 2 *)
```

```

20
21 let position_coins_aux v a b =
22 mouv 3 a v;
23 mouv 3 b v;
24 mouv 1 a v;
25 mouv 1 b v
26 ;;
27
28 let position_coins_FW v a =
29   position_coins_aux v a "Y"
30 ;;
31
32 let scan_triple_aux v_t a b c =
33   let pos = ref 0 in
34     while ( (v_t.(!pos) <> (a,b,c))
35           && (v_t.(!pos) <> (a,c,b))
36           && (v_t.(!pos) <> (b,a,c))
37           && (v_t.(!pos) <> (b,c,a))
38           && (v_t.(!pos) <> (c,a,b))
39           && (v_t.(!pos) <> (c,b,a))
40           && (!pos <= 8) ) do
41     incr pos done;
42     if (!pos <8) then !pos
43     else failwith "erreur scan_triple"
44 ;;
45
46 let scan_triple v a b c =
47   scan_triple_aux (vect_triple v) a b c
48 ;;
49
50 let sortie_coins v a b =
51   mouv 3 a v;
52   mouv 1 b v;
53   mouv 1 a v
54 ;;
55
56
57 (* etape 3 *)
58
59 type orientation = gauche | droite ;;
60
61 let valeur a =
62   match a with
63   | "R" -> 1
64   | "B" -> 2
65   | "O" -> 3
66   | "G" -> 4
67   | _    -> failwith "erreur val"
68 ;;
69
70 let oriente a b =
71   let n = ((valeur a) - (valeur b)) in
72     match n with

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```

73      | 1 -> gauche
74      | (-3) -> gauche
75      | 3 -> droite
76      | (-1) -> droite
77      | 2 -> failwith "2"
78      | (-2) -> failwith "-2"
79      | 0 -> failwith "louche 0"
80      | _ -> failwith "erreur ori"
81 ;;
82
83 let manchot_aux v a b c oriente =
84 if ((oriente a b) = gauche)
85 then begin mouv 1 c v;
86         mouv 1 b v;
87         mouv 3 c v;
88         mouv 3 b v;
89         mouv 3 c v;
90         mouv 3 a v;
91         mouv 1 c v;
92         mouv 1 a v
93     end
94
95 else begin mouv 3 c v;
96         mouv 3 b v;
97         mouv 1 c v;
98         mouv 1 b v;
99         mouv 1 c v;
100        mouv 1 a v;
101        mouv 3 c v;
102        mouv 3 a v
103    end
104 ;;
105
106 let manchot v a b =
107   manchot_aux v a b "Y" oriente
108 ;;
109
110
111 (* etape 4 *)
112
113 let croix_jaune_aux v a b c =
114   mouv 3 a v;
115   mouv 3 c v;
116   mouv 3 b v;
117   mouv 1 c v;
118   mouv 1 b v;
119   mouv 1 a v
120 ;;
121
122 let croix_jaune v a b =
123   croix_jaune_aux v a b "Y"
124 ;;
125

```

```

126 let chaise_aux v a b =
127 while ([|v.(10);v.(21);v.(32);v.(43)|] <> [|"R";"B";"G";"O"|])
128   do
129     mouv 1 a v;
130     mouv 2 b v;
131     mouv 3 a v;
132     mouv 3 b v;
133     mouv 1 a v;
134     mouv 3 b v;
135     mouv 3 a v
136   done
137 ;;
138 let chaise v a = chaise_aux v a "Y" ;;
139
140
141 (* etape 5 *)
142
143 let rotation_cY_aux v a b c =
144   mouv 1 a v;
145   mouv 3 b v;
146   mouv 1 c v;
147   mouv 3 a v;
148   mouv 3 c v;
149   mouv 1 b v;
150   mouv 1 c v;
151   mouv 1 a v;
152   mouv 3 c v;
153   mouv 3 a v
154 ;;
155
156 let rotation_cY v a b = rotation_cY_aux v a b "Y" ;;
157
158
159 let position_coins_bool_aux v a b bool =
160   let booleen = ref (bool (vect_triple v)) in
161     while !booleen do
162       mouv 3 a v;
163       mouv 3 b v;
164       mouv 1 a v;
165       mouv 1 b v;
166       booleen := (bool (vect_triple v))
167     done
168 ;;
169
170 let position_coins_bool v bool =
171   position_coins_bool_aux v "R" "W" bool
172 ;;

```

## E.4 Les étapes

### E.4.1 Étape 1

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4
5 let etape_1_placement_r v =
6   match (scan_double v "R" "W") with
7     |0 -> ()
8     |1 -> mouv 1 "W" v
9     |2 -> mouv 3 "W" v
10    |3 -> mouv 2 "W" v
11    |4 -> mouv 3 "R" v
12    |5 -> mouv 1 "R" v
13    |6 -> mouv 2 "R" v
14    |7 -> begin mouv 1 "O" v ; mouv 2 "W" v end
15    |8 -> begin mouv 2 "B" v ; mouv 1 "W" v end
16    |9 -> begin mouv 1 "G" v ; mouv 3 "W" v end
17    |10 -> begin mouv 2 "G" v ; mouv 3 "W" v end
18    |11 -> begin mouv 2 "O" v ; mouv 2 "W" v end
19    |_ -> failwith "Corrupted data etape 1"
20 ;;
21
22 let etape_1_placement_b v =
23   match (scan_double v "B" "W") with
24     |0 -> failwith "ERREUR ETAPE 1"
25     |1 -> ()
26     |2 -> begin mouv 1 "R" v; mouv 2 "W" v; mouv 3 "R" v end
27     |3 -> begin mouv 1 "R" v; mouv 1 "W" v; mouv 3 "R" v end
28     |4 -> mouv 1 "B" v
29     |5 -> begin mouv 1 "R" v; mouv 3 "W" v; mouv 3 "R" v end
30     |6 -> begin mouv 1 "Y" v; mouv 2 "B" v end
31     |7 -> mouv 3 "B" v
32     |8 -> mouv 2 "B" v
33     |9 -> begin mouv 2 "O" v ; mouv 3 "B" v end
34     |10 -> begin mouv 2 "Y" v ; mouv 2 "B" v end
35     |11 -> begin mouv 3 "Y" v ; mouv 2 "B" v end
36     |_ -> failwith "Corrupted data etape 1"
37 ;;
38
39 let etape_1_placement_g v =
40   match (scan_double v "W" "G") with
41     |0 -> failwith "ERREUR ETAPE 1"
42     |1 -> failwith "ERREUR ETAPE 1"
43     |2 -> ()
44     |3 -> begin mouv 1 "O" v; mouv 1 "G" v end
45     |4 -> begin mouv 3 "B" v; mouv 2 "Y" v; mouv 2 "G" v; mouv
46       1 "B" v end
47     |5 -> mouv 3 "G" v
48     |6 -> begin mouv 3 "Y" v; mouv 2 "G" v end
49     |7 -> begin mouv 2 "O" v; mouv 1 "G" v end
50     |8 -> begin mouv 2 "Y" v; mouv 2 "G" v end
51     |9 -> mouv 1 "G" v
52     |10 -> mouv 2 "G" v
53     |11 -> begin mouv 1 "Y" v; mouv 2 "G" v end

```

```

53     |_ -> failwith "Corrupted data etape 1"
54 ;;
55
56 let etape_1_placement_v =
57   match (scan_double v "0" "W") with
58     |0 -> failwith "ERREUR ETAPE 1"
59     |1 -> failwith "ERREUR ETAPE 1"
60     |2 -> failwith "ERREUR ETAPE 1"
61     |3 -> ()
62     |4 -> begin mouv 3 "B" v; mouv 1 "Y" v;
63           mouv 2 "0" v; mouv 1 "B" v end
64     |5 -> begin mouv 1 "G" v; mouv 3 "Y" v;
65           mouv 2 "0" v; mouv 3 "G" v end
66     |6 -> begin mouv 2 "Y" v; mouv 2 "0" v end
67     |7 -> mouv 1 "0" v
68     |8 -> begin mouv 1 "Y" v; mouv 2 "0" v end
69     |9 -> mouv 3 "0" v
70     |10 -> begin mouv 3 "Y" v; mouv 2 "0" v end
71     |11 -> mouv 2 "0" v
72     |_ -> failwith "Corrupted data etape 1"
73 ;;
74
75
76 let etape_1_replacement_v =
77 if (v.(1)= "W") then ()
78 else begin mouv 1 "R" v;
79           mouv 3 "W" v;
80           mouv 1 "B" v;
81           mouv 1 "W" v
82         end ;
83
84 if (v.(3)= "W") then ()
85 else begin mouv 1 "B" v;
86           mouv 3 "W" v;
87           mouv 1 "0" v;
88           mouv 1 "W" v
89         end ;
90
91 if (v.(5)= "W") then ()
92 else begin mouv 1 "G" v;
93           mouv 3 "W" v;
94           mouv 1 "R" v;
95           mouv 1 "W" v
96         end ;
97
98 if (v.(7)= "W") then ()
99 else begin mouv 1 "0" v;
100          mouv 3 "W" v;
101          mouv 1 "G" v;
102          mouv 1 "W" v
103        end
104 ;;
105

```

```

106 let etape1 v =
107   etape_1_placement_r v;
108   etape_1_placement_b v;
109   etape_1_placement_g v;
110   etape_1_placement_o v;
111   etape_1_replacement v
112 ;;

```

### E.4.2 Étape 2

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4
5 let sortie_coins_0 v = sortie_coins v "B" "Y";;
6 let sortie_coins_1 v = sortie_coins v "R" "Y";;
7 let sortie_coins_2 v = sortie_coins v "O" "Y";;
8 let sortie_coins_3 v = sortie_coins v "G" "Y";;
9
10 let temps6 v a b c place nb=
11   let temp = ref 1 in
12     if (nb = 0) then
13       while (((vect_triple v).(place) <> (a,b,c)) && (!temp <
14         7)) do
15           position_coins_fW v c;
16           incr temp
17         done
18       else
19         while (((vect_triple v).(place) <> (a,b,c)) && (!temp <
20           7)) do
21             position_coins_fW v b;
22             incr temp
23           done
24     done
25   ;;
26
27 let placement_coins_fW v a b c place nb = let test = ref 0 in
28   while (((vect_triple v).(place) <> (a,b,c)) && (!test<50)) do
29     if (!test=50) then failwith "erreur plac_coins"
30     else let temp2 = (place +4) and temp1 = place in
31       match 0 with
32         | _ when ((scan_triple v a b c)=temp1) ->
33           temps6 v a b c place nb
34         | _ when ((scan_triple v a b c)=0) -> sortie_coins_0 v
35         | _ when ((scan_triple v a b c)=1) -> sortie_coins_1 v
36         | _ when ((scan_triple v a b c)=2) -> sortie_coins_2 v
37         | _ when ((scan_triple v a b c)=3) -> sortie_coins_3 v
38         | _ when ((scan_triple v a b c)=temp2) ->
39           temps6 v a b c place nb
40         | _ -> mouv 1 "Y" v
41     done
42   ;;
43
44 let placement_coins_fW_0 v  =

```

```

42   placement_coins_fW v  "W" "R" "B" 0 0
43 ;;
44 let placement_coins_fW_1 v  =
45   placement_coins_fW v  "W" "R" "G" 1 1
46 ;;
47 let placement_coins_fW_2 v  =
48   placement_coins_fW v  "W" "B" "O" 2 0
49 ;;
50 let placement_coins_fW_3 v  =
51   placement_coins_fW v  "W" "G" "O" 3 1
52 ;;
53
54
55
56 let etape2 v =
57   placement_coins_fW_0 v ;
58   placement_coins_fW_1 v ;
59   placement_coins_fW_2 v ;
60   placement_coins_fW_3 v
61 ;;

```

### E.4.3 Étape 3

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4
5 let extraction = [|4;5;7;9|];;
6
7 let recherche_centre v_d = let i = ref 0 in
8 while (!i <= 3) do
9   match v_d.(extraction.(!i)) with
10    | ("Y",_) -> incr i
11    | (_, "Y") -> incr i
12    | ("R", "B") when (!i=0) -> incr i
13    | _ when (!i=0) -> i:=4
14    | ("R", "G") when (!i=1) -> incr i
15    | _ when (!i=1) -> i:=5
16    | ("B", "O") when (!i=2) -> incr i
17    | _ when (!i=2) -> i:=7
18    | ("G", "O") when (!i=3) -> incr i
19    | _ when (!i=3) -> i:=9
20    | _ -> failwith "pb recherche"
21 done;
22 !i
23 ;;
24
25
26 let sortie_centre v n = match n with
27  | 4 -> manchot v "R" "B"
28  | 5 -> manchot v "R" "G"
29  | 7 -> manchot v "B" "O"
30  | 9 -> manchot v "G" "O"

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```

31   | _ -> failwith "pb sortie centre"
32 ;;
33
34
35 let rota_y v = let nb_mouv = ref 0 in
36   while ( ((red v) = false)
37         && ((green v) = false)
38         && ((blue v) = false)
39         && ((orange v) = false)
40         && (!nb_mouv<=3)) do
41     mouv 1 "Y" v; incr nb_mouv
42   done;
43 ;;
44
45 let place_manchot v = match 0 with
46   | _ when (red v)-> manchot v v.(10) v.(52)
47   | _ when (green v)-> manchot v v.(32) v.(50)
48   | _ when (blue v)-> manchot v v.(21) v.(48)
49   | _ when (orange v)-> manchot v v.(43) v.(46)
50   | _ -> sortie_centre v (recherche_centre (vect_double v))
51 ;;
52
53
54
55 let etape3 v =
56   while not ((rg v) && (go v) && (ob v) && (br v)) do
57     rota_y v;
58     place_manchot v
59   done
60 ;;

```

### E.4.4 Étape 4

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4
5 let etape_4_1 v =
6   while ([|v.(52);v.(48);v.(46);v.(50)|]<>[|"Y";"Y";"Y";"Y"|])
7     do
8       match (v.(52),v.(48),v.(46),v.(50)) with
9         | ("Y","Y",_,_) -> croix_jaune v "O" "G"
10        | (_, "Y", "Y", _) -> croix_jaune v "G" "R"
11        | (_, _, "Y", "Y") -> croix_jaune v "R" "B"
12        | ("Y", _, _, "Y") -> croix_jaune v "B" "O"
13        | (_, _, _, _) -> croix_jaune v "O" "G"
14   done
15
16
17
18 let test_orange v =
19   if (v.(43)="0") then 1 else 0

```

```

20 ;;
21
22 let test_blue v =
23   if (v.(21)="B") then 1 else 0
24 ;;
25
26 let test_red v =
27   if (v.(10)="R") then 1 else 0
28 ;;
29
30 let test_green v =
31   if (v.(32)="G") then 1 else 0
32 ;;
33
34
35 let somme_test v = (test_green v)
36           + (test_orange v)
37           + (test_blue v)
38           + (test_red v)
39 ;;
40
41 let etape_4_2 v = let compteur = ref 0 in
42   while ((somme_test v) <> 1) do
43     if (!compteur<=3) then begin mouv 1 "Y" v; incr compteur end
44     else begin
45       mouv 1 "R" v;
46       mouv 2 "Y" v;
47       mouv 3 "R" v;
48       mouv 3 "Y" v;
49       mouv 1 "R" v;
50       mouv 3 "Y" v;
51       mouv 3 "R" v;
52       compteur := 0
53     end
54   done
55 ;;
56
57 let etape_4_3 v = match ((test_orange v),
58                           (test_blue v),
59                           (test_red v),
60                           (test_green v)) with
61   |(1,0,0,0) -> chaise v "B"
62   |(0,1,0,0) -> chaise v "R"
63   |(0,0,1,0) -> chaise v "G"
64   |(0,0,0,1) -> chaise v "O"
65   |_ -> failwith "4_3"
66 ;;
67
68 let etape4 v =
69   etape_4_1 v;
70   etape_4_2 v;
71   etape_4_3 v
72 ;;

```

#### E.4.5 Étape 5

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4
5 let etape_5_1 v =
6   while not ((rby (vect_triple v))
7             && (rgy (vect_triple v))
8             && (boy (vect_triple v))
9             && (goy (vect_triple v))) do
10   match 0 with
11     |_ when rby (vect_triple v) -> rotation_cY v "R" "O"
12     |_ when rgy (vect_triple v) -> rotation_cY v "G" "B"
13     |_ when boy (vect_triple v) -> rotation_cY v "B" "G"
14     |_ when goy (vect_triple v) -> rotation_cY v "O" "R"
15     |_ -> rotation_cY v "R" "O"
16   done
17 ;;
18
19 let etape_5_2 v =
20 position_coins_bool v (orientate_rby); mouv 3 "Y" v;
21 position_coins_bool v (orientate_boy); mouv 3 "Y" v;
22 position_coins_bool v (orientate_ogy); mouv 3 "Y" v;
23 position_coins_bool v (orientate_gry); mouv 3 "Y" v
24 ;;
25
26 let etape5 v =
27   etape_5_1 v;
28   etape_5_2 v
29 ;;

```

#### E.5 Exemple de fichier final

```

1 #open "macro";;
2 #open "mouv";;
3 #open "booleen";;
4 #open "etape1";;
5 #open "etape2";;
6 #open "etape3";;
7 #open "etape4";;
8 #open "etape5";;
9
10 let vect_gen = [| "B"; "Y"; "G"; "B"; "W"; "G"; "B"; "B"; "G";
11                  "B"; "O"; "O"; "W"; "R"; "R"; "W"; "B"; "R";
12                  "O"; "O"; "R"; "Y"; "B"; "R"; "O"; "Y"; "R";
13                  "Y"; "W"; "G"; "W"; "G"; "R"; "Y"; "O"; "G";
14                  "Y"; "W"; "O"; "O"; "G"; "B"; "Y"; "R";
15                  "W"; "R"; "W"; "G"; "Y"; "G"; "Y"; "B"; "W" |]
16 ;;
17
18 let reponse = ref "";;
19

```

```

20
21
22 let vect_to_string v = let solution = ref "[]" in
23   for i=53 downto 0 do
24     solution := "\\"^(!solution);
25     solution := v.(i)^(!solution);
26     solution := ";"\\"^(!solution)
27   done;
28   solution := "[|]^(!solution);
29   !solution
30 ;;
31
32 let rec list_to_string l = let solution = ref "" in
33 begin
34   match l with
35   | [] -> ()
36   | a::suite -> reponse := a^(!reponse); list_to_string suite
37 end
38 ;;
39
40 let rec filtre l = match l with
41   | [] -> []
42   | a::b::c::d::suite when ((a=b) && (b=c) && (c=d))
43     -> filtre suite
44   | a::b::c::suite when ((a=b) && (b=c))
45     -> ("-"^a^";")::(filtre suite)
46   | a::b::suite when (a=b)
47     -> ("2"^a^";")::(filtre suite)
48   | a::suite -> (a^";")::(filtre suite)
49 ;;
50
51
52 let kube v =
53   etape1 v;
54   etape2 v;
55   etape3 v;
56   etape4 v;
57   etape5 v
58 ;;
59
60 print_string
61   "Vecteur du kube avant application de l'algorithme :\n"
62 ;;
63 print_newline();;
64 print_string (vect_to_string vect_gen);;
65 print_newline();;
66 print_newline();;
67 print_string
68   "Vecteur du kube apr s application de l'algorithme :\n"
69 ;;
70 print_newline();;
71 print_string (vect_to_string (kube vect_gen; vect_gen));;
72 print_newline();;

```

## E. NOTRE PROGRAMME ENTIER ET NON COMMENTÉ

---

```
73 print_newline();  
74 print_int (list_length !solution);;  
75 print_newline();  
76 print_newline();  
77 list_to_string (filtre !solution);;  
78 print_string !reponse;;  
79 print_newline();  
80 print_newline();;
```