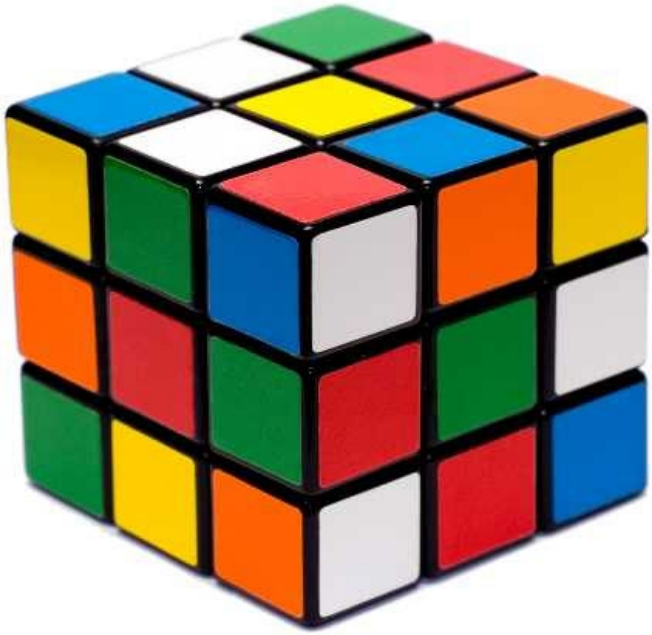
Viki's Cube

version 1.0.0

DIVERSITY by EPITECH

I. Introduction

En 1989, Viki est une jeune hongroise passionné de Rubik's cube. Depuis quelques années, elle se lasse des tournois, survolant l'adversité à chaque fois. Elle a alors l'idée de créer un programme qui résout les fameux cubes contre lequel elle pourrait s'entrainer. En effet, Viki a découvert la programmation quelques semaines auparavant et elle aura besoin de ton aide pour réussir son projet et faire fonctionner son code.



Un rubik's cube non résolu !

II. Récupère le cube !

Commence par regarder les fichiers « .txt ». C'est comme cela que serons stockés les faces des Rubik's Cubes. Pour récupérer leur contenu, il faut que tu ouvres le fichier « rubic.rb ». Dans la fonction `parse()`, rajoute :

```
file = File.open(file_name)
```

Ensuite, lis le contenu du fichier. Pour cela il suffit de faire :

```
content = file.read
```

Pour finir, pense bien à fermer le fichier, comme cela :

```
file.close
```

Le code qui suit ce que tu viens de coder sert à stocker chaque face du Rubik's cube dans un tableau. C'est ce tableau que tu modifieras par la suite pour résoudre le Rubik's cube.

III. Crée le mouvement U et U'.

a. Crée le mouvement U

Le mouvement U consiste à déplacer la ligne supérieure du cube sur la droite. Regarde ici si tu souhaites une représentation visuelle de ce mouvement. Dans le code, la ligne 0 de la face 0 prendra la valeur de la ligne 0 de la face 1, la ligne 0 de la face 1 prendra la valeur de la ligne 0 de la face 2 etc.

b. Crée le mouvement U'

Maintenant que tu as réussi la rotation U, tu peux faire le mouvement inverse, le mouvement U'. Tu as deux manières de le faire, soit tu fais l'inverse de ce que tu as fait précédemment, soit tu fais 3 fois la rotation U. Et oui, faire quatre fois la même rotation c'est comme ne rien faire !

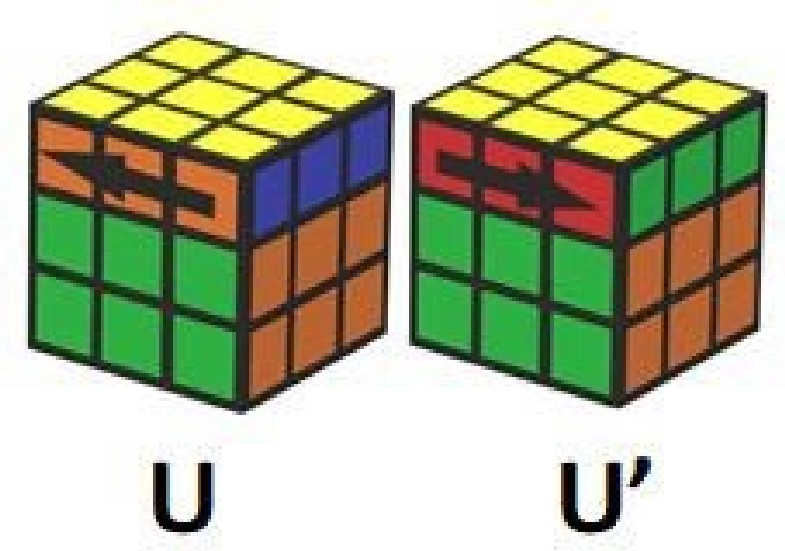


Schéma des rotations U et U'

IV. Crée la transformation Gauche et Droite.

a. Crée la transformation gauche

La transformation gauche consiste à tourner la totalité du cube sur la gauche. Regarde ici si tu souhaites une représentation visuelle de ce mouvement. Dans le code, la ligne 1 de la face 3 prendra la valeur de la ligne 1 de la face 2, la ligne 1 de la face 2 prendra la valeur de la ligne 1 de la face 1 etc.

Enfin, il ne faut pas oublier que la face du dessus (4) et du dessous (5) tournent sur eux même, respectivement dans le sens horaire pour la face du dessous et antihoraire pour la face du dessus.

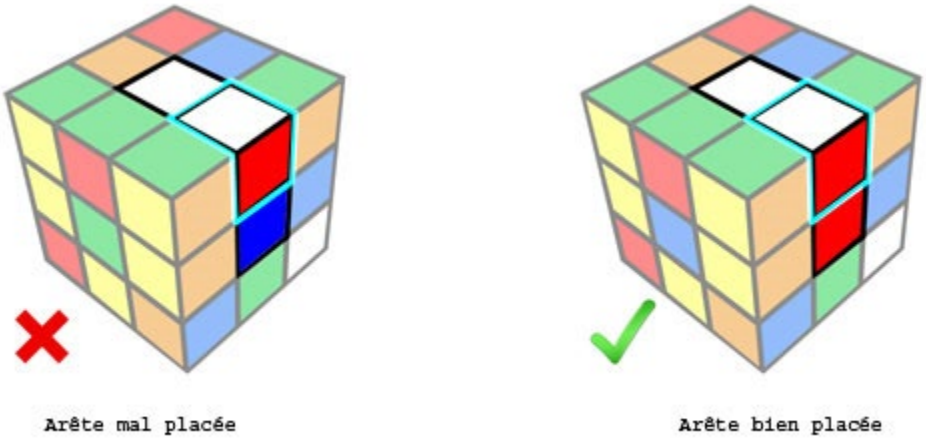
b. Crée la transformation droite

Maintenant que tu as réussi la transformation gauche tu peux faire le mouvement inverse, la transformation droite. Comme pour le mouvement U' et le mouvement U, tu as deux manières de le faire, soit tu fais l'inverse de ce que tu as fait précédemment, soit tu fais 3 fois la transformation gauche.

V. Résous la face blanche du Rubik's Cube.

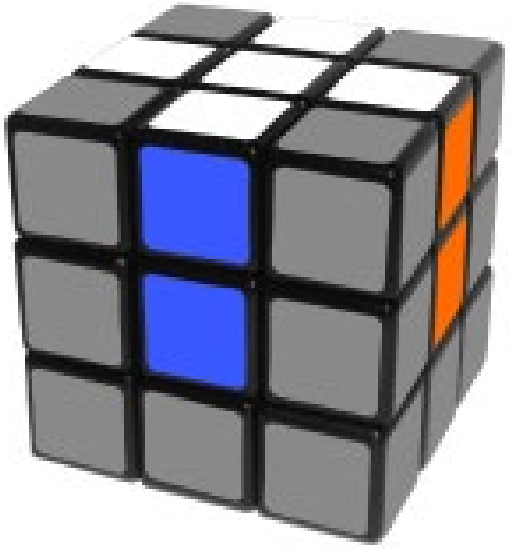
`

A partir d'ici tout ce que tu devras modifier sera dans « algo.rb ». Pour bien comprendre, il faut faire en sorte à ce que les arêtes de chaque face soient de la même couleur.



Exemple de deux Rubik's, l'un avec une arête bien placée, l'autre mal placée

Si les 4 arêtes ne sont pas bien placées à l'origine, au moins 2 d'entre eux le seront forcément. Une fois qu'on les a trouvés il faut appliquer l'une des 2 formules possible en fonction du cas de figure :



Deux arêtes bien placées côte à côte

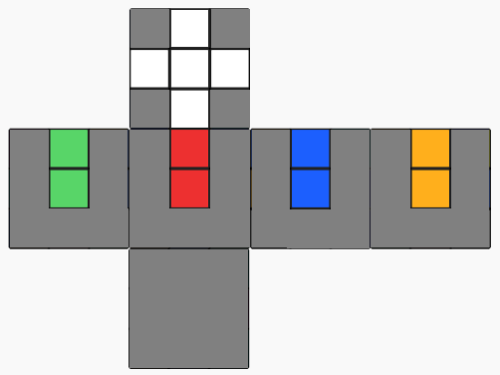
La fonction `def arrete_blanche_formule_cote(ru)` sera appelée si les 2 arêtes bien placées se situent sur des faces adjacentes. Voici la formule à appliquer :

* ru.r
* ru.u
* ru.rp
* ru.u
* ru.r
* ru.u
* ru.u
* ru.rp
* ru.u

La fonction `def arrete_blanche_formule_face(ru)` quant à elle sera appelée si les 2 arêtes sont situées sur des faces opposées. Applique la suite ci-dessous pour résoudre la face.

* ru.r
* ru.u
* ru.rp
* ru.u
* ru.r
* ru.u
* ru.u
* ru.rp
* ru.transform_right

Voilà à quoi devrait ressembler les arêtes en question :



Les arêtes des 4 faces chaque sont bien placées

> ⚠ Attention
>
> Si tu n'obtiens pas ce paterne reprend l'étape précédente ou demande l'aide d'un cobra

VI. Maintenant le reste du cube.

a. La deuxième couronne.

Maintenant que tu as réussi à replacer une face, il faut résoudre les autres ! L'étape suivante consiste à replacer la « deuxième couronne », la ligne du milieu du cube. Il faut une nouvelle fois appliquer une suite de mouvement dans la fonction `def deuxieme_couronne(ru)`, mais attention, il y a deux cas différents. A toi de trouver quelle suite va où !

- ru.up
- ru.lp
- ru.u
- ru.l
- ru.u
- ru.f
- ru.up
- ru.fp

- ru.u
- ru.r
- ru.up
- ru.rp
- ru.up
- ru.fp
- ru.u
- ru.f

b. La croix jaune.

L'étape suivante consiste à reformer la croix jaune. Dans la fonction `def croix_jaune_seq(ru)` effectue les mouvements suivants :

- ru.rp
- ru.up
- ru.fp
- ru.u
- ru.f
- ru.r

c. Les arrêtes jaunes.

Dans cette étape, il va falloir placer les arrêtes jaunes sur notre rubik's cube afin de le compléter. La fonction dans laquelle il va falloir écrire coder est : `def placement_coins(ru)`

Dans un premier lieu il va falloir vérifier 4 fois que les coins soient bien placés et s'ils ne sont pas bien placés on va réaliser une `transform.right`.

Une fois cela fait on vérifie si les coins sont bien placés et s'ils sont correct alors on réalise le placement des coins dans une autre fonction que l'on va appeler : `def placement_formule(ru)`

* ru.lp
* ru.u
* ru.r
* ru.up
* ru.l
* ru.u
* ru.rp
* ru.up

Retour dans notre première fonction. Suite à ça, on va vérifier si les coins sont mal placé on réalise une transform.left puis on utilise la fonction "placement_formule" et une transform.right.

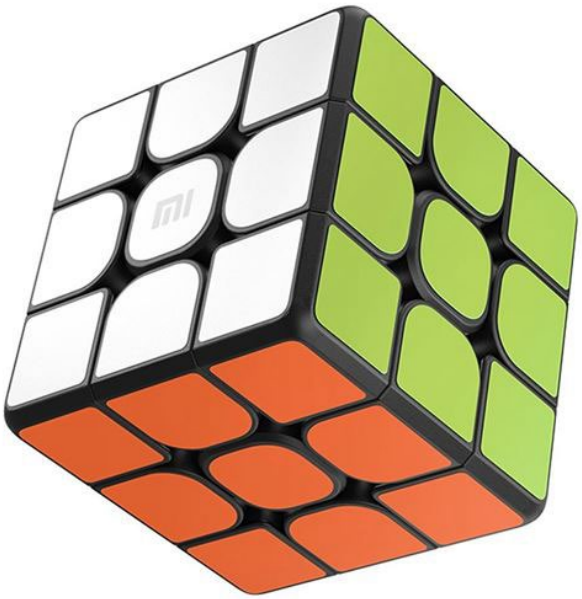
Mais si nos coins n'étaient pas bien placés au départ on appel juste la fonction "placement_formule" et on return notre fonction "placement_coins"

d. Les coins jaunes.

Pour cette fonction qui va s'appeler `def coins_jaunes(ru)` nous allons dire que jusqu'à ce que `is_jaunes_correct?(r)` est faux alors on fait une `transform_right` après cela on vérifie quatre fois que `is_jaunes_correct?(r)` et s'il l'est alors on va faire `coin_blanc_formule(ru)` et on finit par un `ru.u`.

VII. Conclusion

Bravo, grâce à toi Viki à trouver un adversaire à sa hauteur. Mais elle ne va sans doute pas mettre longtemps à surpasser ce programme. Tu peux donc aider Viki à améliorer l'algorithme pour qu'elle continue à progresser !



Un rubik's cube résolu !