Projet Python : Flip

Introduction et Plan

- Presentation du projet
- Choix adoptés
- Structure du cube
- La résolution classique
- La résolution Petrus
- Résultats
- Optimisation
- Difficultés
- Conclusion

1

Présentation

Objectif, délai,compétences

2

Choix adoptés

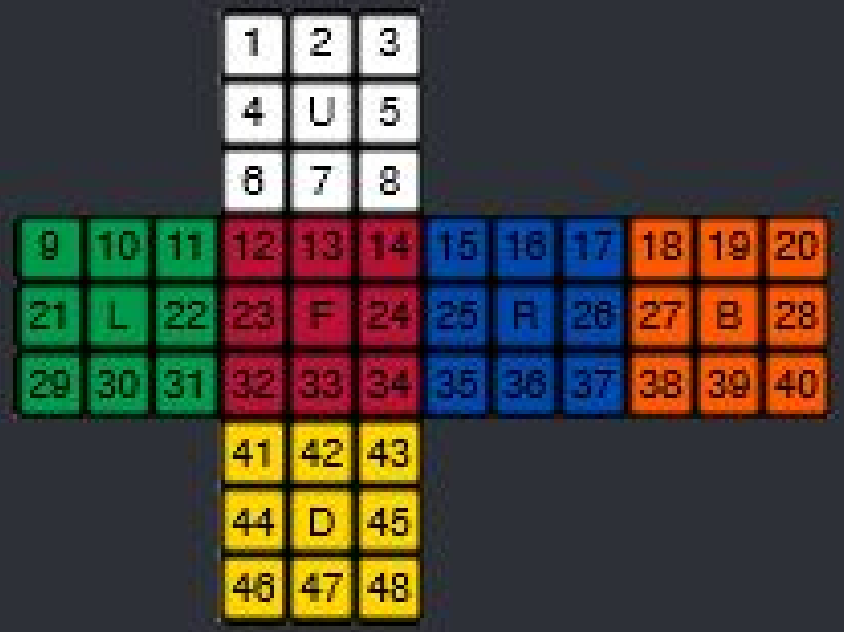
Prendre un bon départ

Le choix du dictionnaire

Simple a créer et manipuler.

Utilisation de compétences connues

Se prête bien à la situation



Commencer par la résolution classique

- Résolution simple à comprendre
- Très méthodique

- Nombre de mouvements élevé

Puis implémenter la résolution de Pétrus

- Nombre de coups peu élevé
- Résolution demandant plus de reflexion
- Solution plus “visuelle” sur les premières étapes

En paralléle : La résolution de Fridrich

- Assez proche de la Resolution de Petrus
- Très bons résultats en terme de mouvements
- Plus gourmand en temps et mémoire

3

La structure du cube

Rubik.py

Les méthodes

```
    _(self,cube=''):
   .solve = [] #On initialise la liste des mouveme
 elf.solution = ''
self.orientation =''
self.mvt = ['U','L','F','R','B','D']# la liste des mouveme
self.Dcube =  {}
#La fonction prend en paramètre une chaîne de 54 caractère et
if self.estCube(cube):
    self.chaine=cube
    #On ajoute tout d'abord les pièces qui ne bougent pas (index
    self.Dcube["U"] = cube[4]
    self.Dcube["L"] = cube[22]
    self.Dcube["F"] = cube[25]
    self.Dcube["R"] = cube[28]
    self.Dcube["B"] = cube[31]
    self.Dcube["D"] = cube[49]
    #On recompose la chaîne en supprimant les pièces que l'on vient
    cube = cube[0:4]+cube[5:22]+cube[23:25]+cube[26:28]+cube[29:31]+
    for i in range(0,48):
        self.Dcube[i+1] = cube[i] #On ajoute chaque autre pièce ave

else :
    self.MAZ()

solChaine(self):
for i in range (0,len(self.solve)):
    self.solution= self.solution+self.solve[i]


estCube(self,chaine):
precaution supplémentaire pour s'assurer que notre in
  len(chaine) == 54: #Si la longueur est bonne
    r i in range(0,54): #Et que chaque caractèr
        chaine[i] != 'O' and chaine[i] !=
            +(chaine[i])
```

Initialisation

Mouvements du cube

Affichage

Scramble

Le dictionnaire

- Simple a mettre en place
- Chaque changement d'état facile a implémenter
- Facile à comprendre
- On modélise des étiquettes et pas des pièces

4 La résolution classique

Un premier jet

- Répartition des taches par étapes (découpage en 7)
- Vite implémentée
- La solution est composée de nombreux mouvement.

5 La résolution de Petrus

Etape 1 : Assemblage du cube 2x2x2

1) Reconnaissance de motif

2) Construction de paires

3) Assemblage de blocs

Problème complexe : nécessite une solution alternative

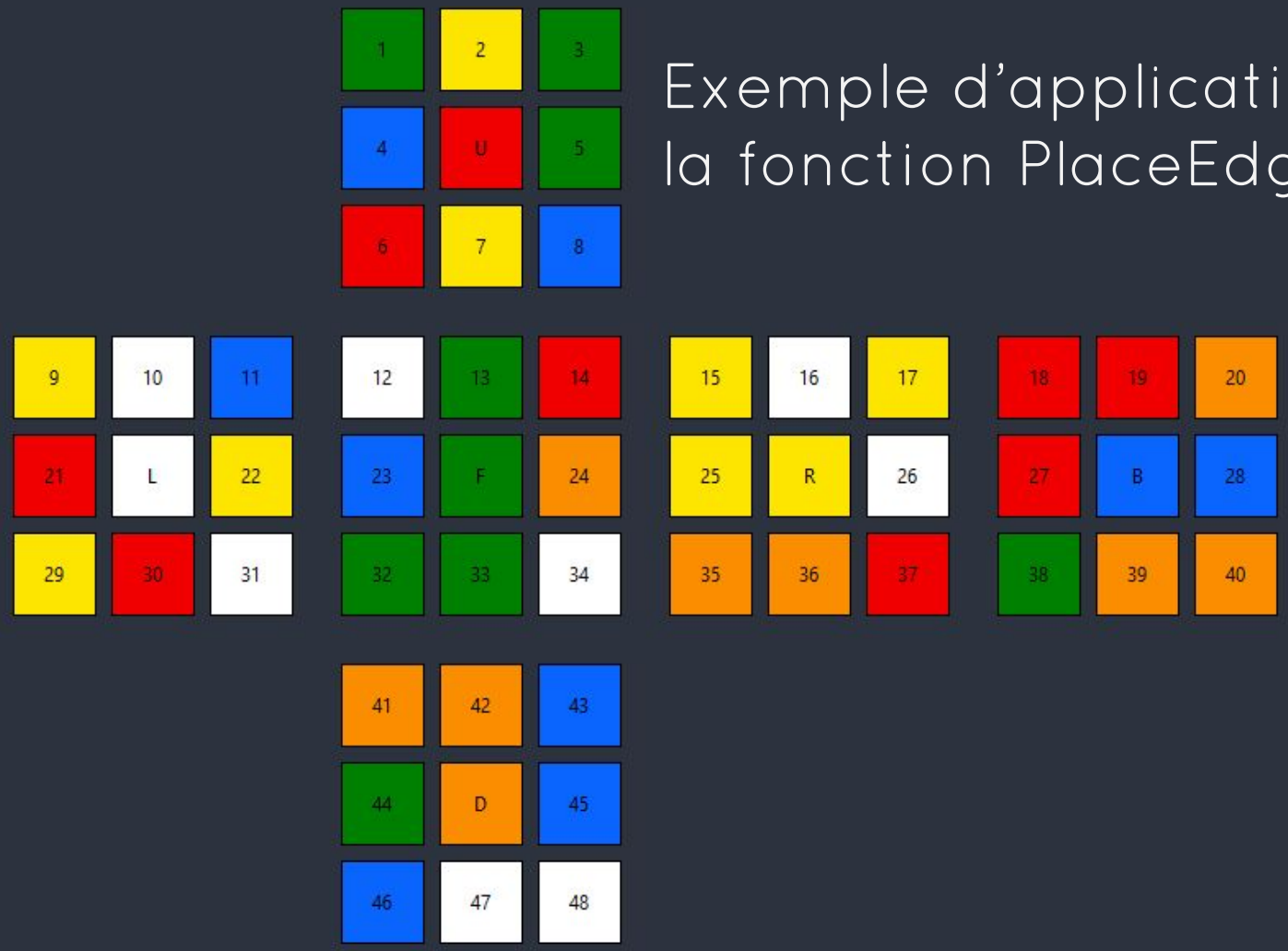
Etape 1 : Assemblage du cube 2x2x2

1) Placer le meilleur coin

2) Orienter le coin

3) Placer les arêtes

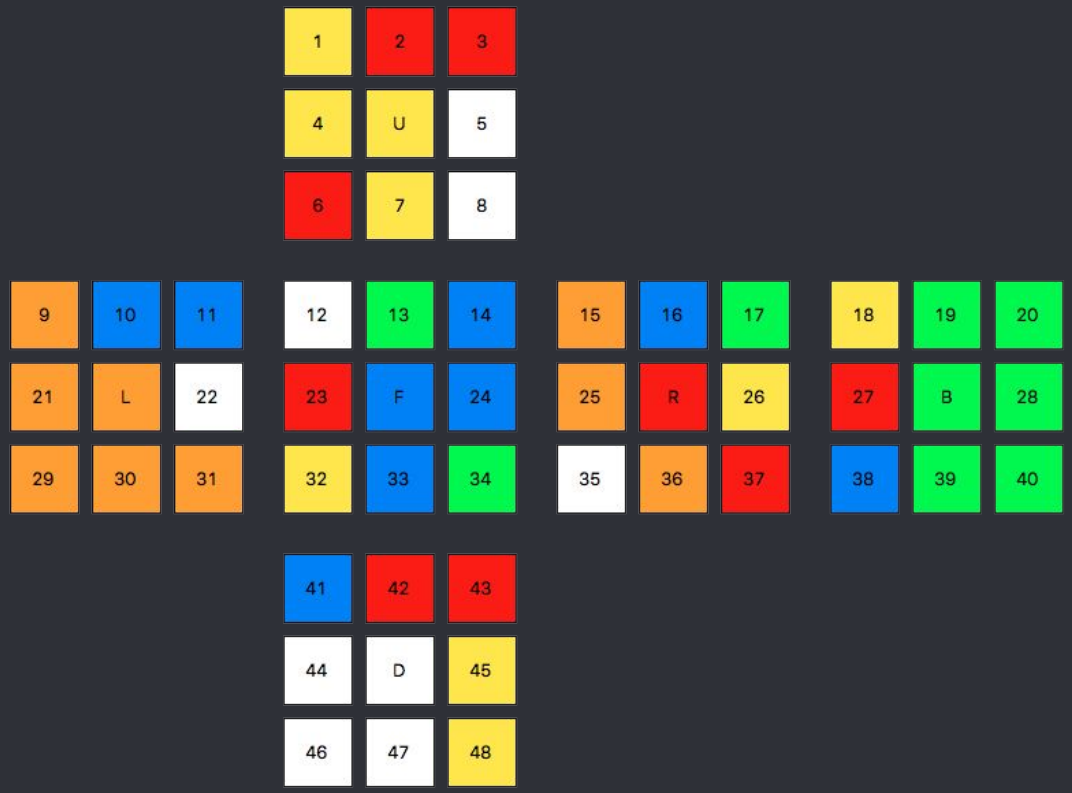
→ Méthode plus simple au détriment du nombre de coups (~+10 coups)

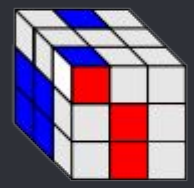


Exemple d'application de la fonction PlaceEdge

Etape 2 : Réalisation du cube 2x2x3

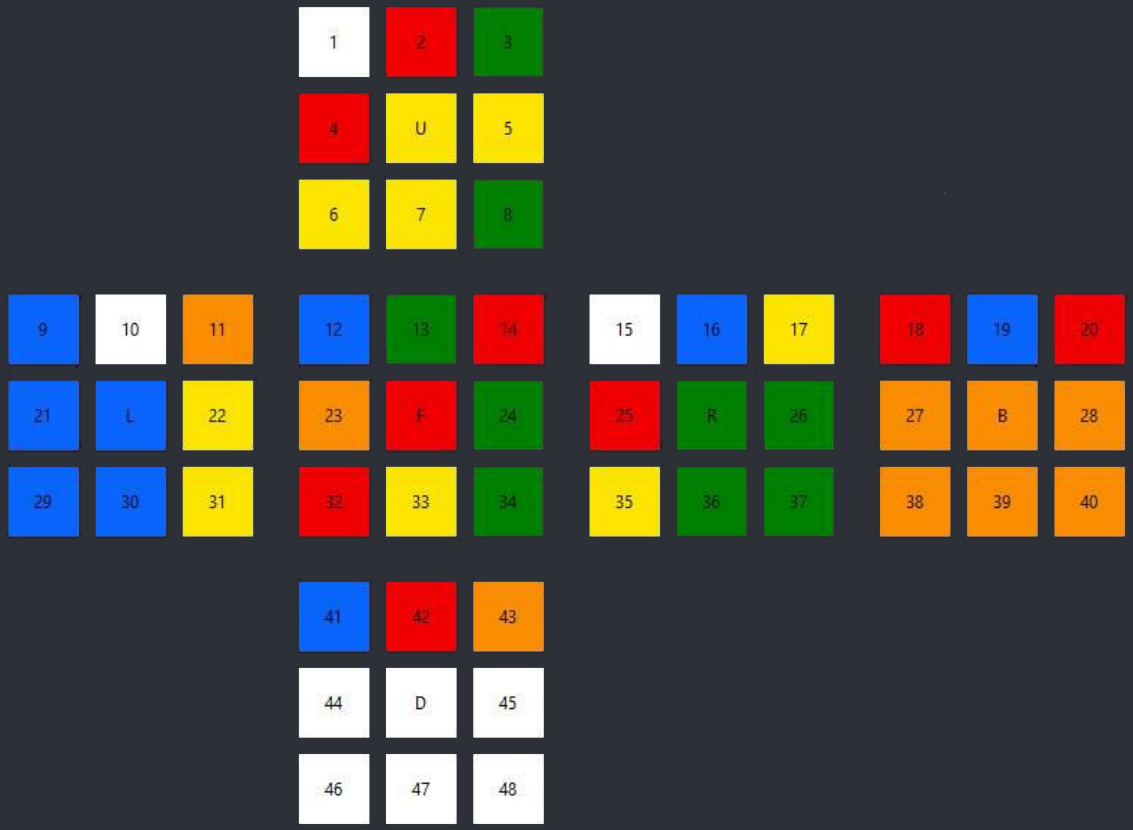
39 combinaisons possibles avec F2L





Etape 3 : Orientation des arêtes

68 combinaisons possibles d'orientation



Etape 4 : Fin des deux premières couronnes

Faire le cube 2*2*1

PF2L 20 combinaisons possibles

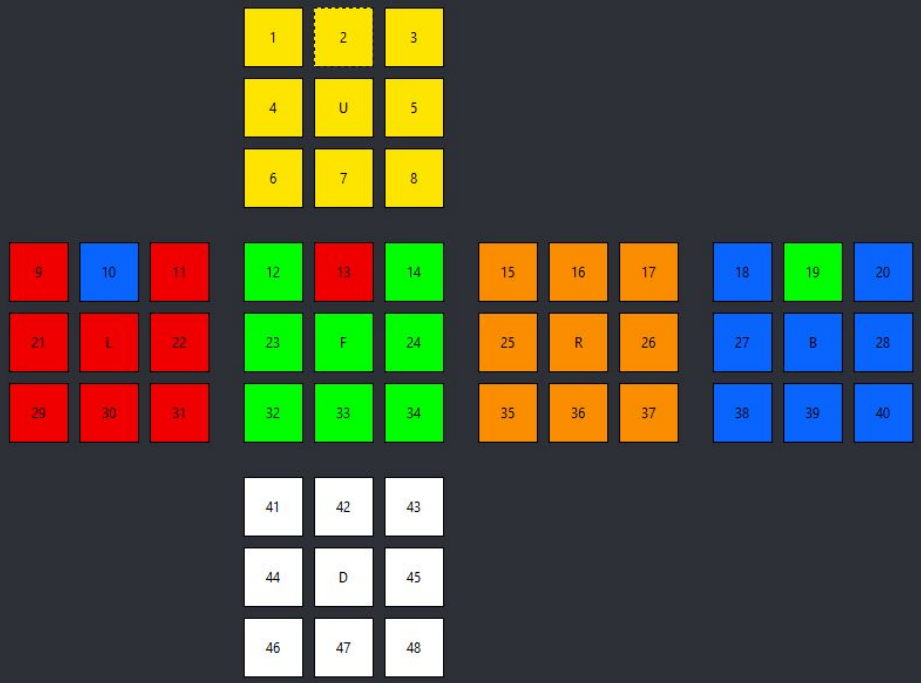
Application des cas COLL

44 combinaisons possibles

Etape 6 : placement des dernières arêtes

Seulement 4 combinaisons possibles

Une réorientation du cube



Modification de la solution

Ajout de l'orientation du cube

Avec les modifications de l'ajout des mouvements adaptés

Simplification de la solution
(ex : RR → R2)

Transformation du tableau en chaîne de caractère

6 Résultats

7

Optimisation

en longueur, en temps et en mémoire

Optimisation

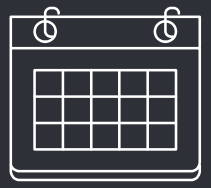
- En terme de longueur : *poqb.py*
- En terme de temps et mémoire : *poqb2.py via la Resolution Classique*
- La fonction *Choix()*

8 Difficultés et apport personnel

pour le groupe

Les difficultés

Organisation



Traduction de methode vers algo

Novice par rapport au sujet

9 Conclusion

Merci de votre attention