ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

RAPPORT FINAL PTS

Majeure : DATA & INTELLIGENCE ARTIFICIELLE

Intitulé du sujet : Exploration exhaustive du Redi Cube

Nom de l'encadrant : Maxime BERRIOT

Numéro de l'équipe : DIA - 17

Date de rédaction : 22/03/2022

Membres de l'équipe

Nom	Prénom
NENEZ	Owen
PETILLION	Pierre
NADESU	Pratheswar

SOMMAIRE

1 Introduction

N'auriez-vous jamais souhaité savoir en combien de coût le casse-tête que vous aviez entre les mains pouvait être résolu ? En l'occurrence, notre casse-tête s'avère être le Redi Cube. Celui-ci est un cube, variant du Rubik's Cube, dont chaque sommet d'une face est relié à ses arêtes adjacentes. L'objectif de ce projet est de déterminer le nombre de Dieu ou du moins d'essayer de s'en approcher. Définissons tout d'abord ce qu'est le nombre de Dieu, prenons par exemple celui du Rubik's cube, qui est 20. Cela signifie que tout mélange d'un Rubik's cube peut être résolu en 20 mouvements au maximum.

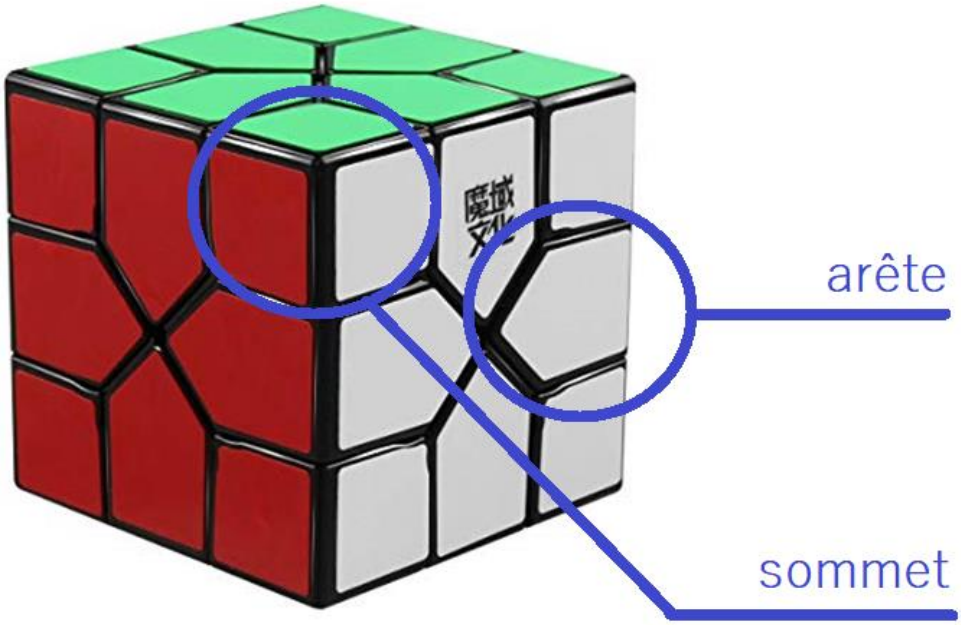
Notre stratégie est semblable aux travaux dans la résolution du nombre de Dieu pour le Rubik's Cube entamés en 2010, à l'issue de laquelle, l'équipe de chercheurs se sont servis des machines mises à disposition par Google afin de calculer le nombre de Dieu. Le projet rassemble à la fois un aspect mathématique, un aspect informatique et un aspect de modélisation qui ouvrent la voie à une pluralité de réflexions. Nous pouvons ainsi nous demander quel est le nombre de Dieu d'un Redi Cube ? En partant de là, une grande partie de notre travail a été de comprendre et de modéliser comment s'articulent les mouvements du Redi Cube, le fonctionnement étant largement différent du modèle original. À la suite de ça, s'est posé la question de comment atteindre notre objectif, et plusieurs pistes et approches ont été expérimentées afin de travailler en ce sens, toujours avec le même fil conducteur : résoudre un Redi cube, de plus en plus rapidement, tout en travaillant sur le nombre de coût utilisés.

2 Objectifs et livrables du projet

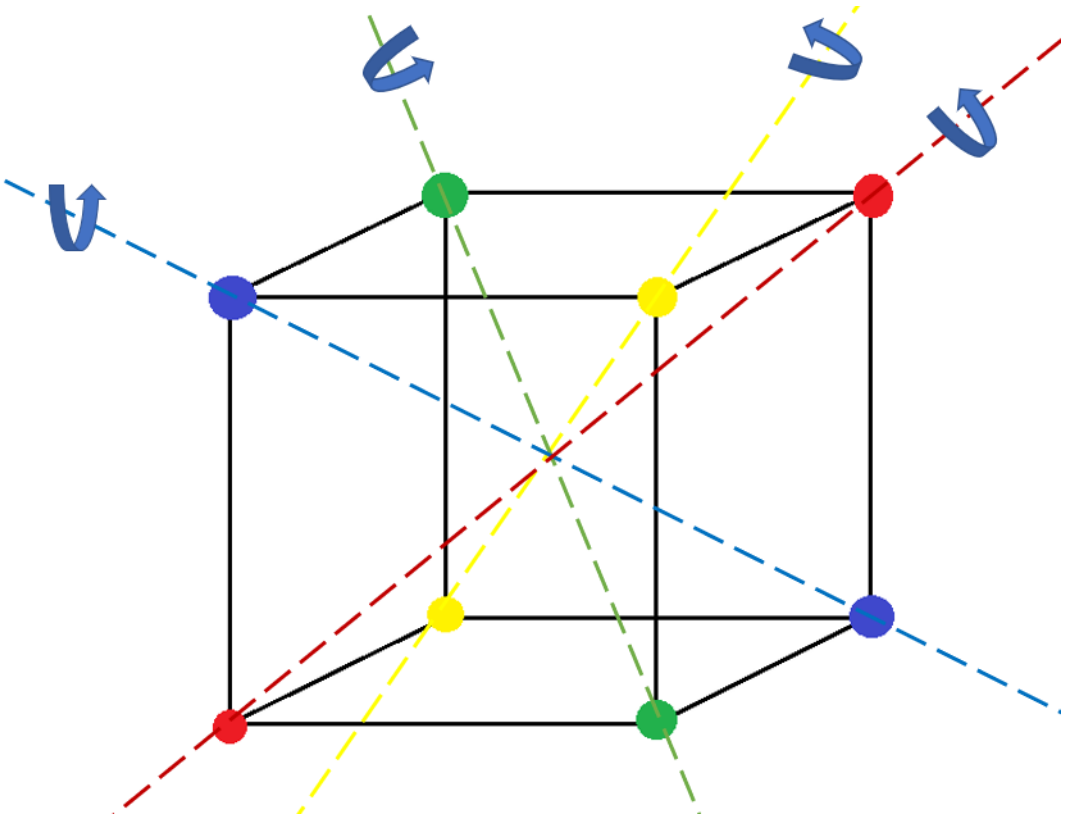
Les livrables de ce projet sont les suivants :

- Ce rapport sous format PDF
- Le Code source
- Une vidéo de démonstration de la résolution d'un Redi cube
- Un manuel d'utilisation de notre interface graphique
- Une approximation du nombre de dieux

3 Définitions



Dans le Redi Cube, on compte 8 **sommets** et 12 **arêtes**.



Le Redi Cube comporte 4 **axes de rotation** selon lesquels les pièces tournent.

4 Contraintes

Modélisation du Redi Cube

Le retard dans la livraison des Redi Cube nous a amené à modéliser le cube dans un premier temps via du code sous le langage Python à l'aide de la librairie VPython. Cependant, un Redi Cube physique aurait permis une meilleure fiabilité pour la modélisation, une meilleure compréhension du fonctionnement et de la stratégie de résolution du cube.

Compréhension des algorithmes de résolution

Au début du projet, aucun des membres du groupe ne savait résoudre un Redi Cube. Nous devions ainsi apprendre à se familiariser avec le cube : savoir comment il pouvait être résolu. Pour ce faire, nous avons effectué des recherches sur son fonctionnement en suivant, notamment, des tutoriels de résolution sur YouTube [1].

Optimisation d'un algorithme de résolution

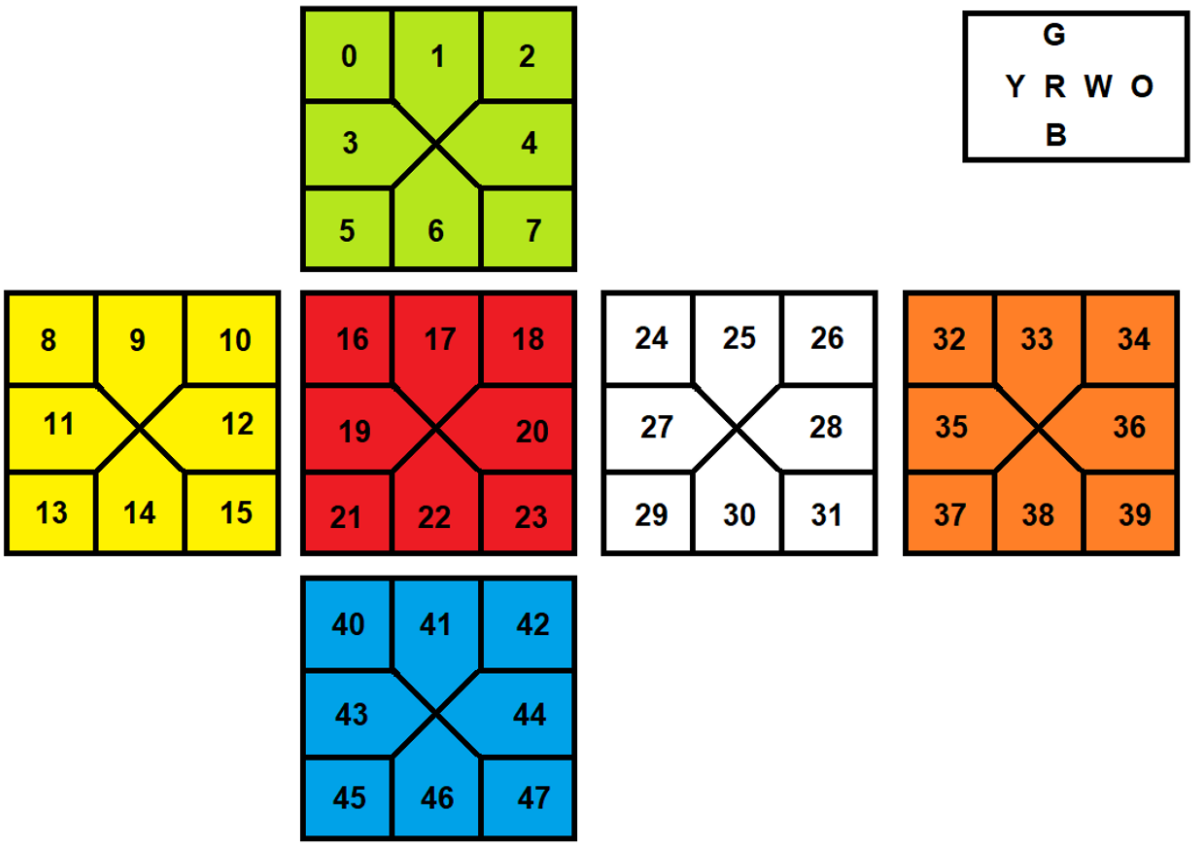
Durant la phase de recherche pour un algorithme de résolution, nous étions passés par plusieurs stratégies. Et pour chacune des stratégies énoncées par la suite (cf. Résolution d'un Redi cube), nous devions peser les avantages et inconvénients des algorithmes et de leur efficacité de résolution en termes de complexité au niveau du temps et de mémoire. De plus, les ressources des machines des membres du groupe n'étaient pas équitablement réparties. Ainsi, nous devions nous répartir l'exécution des algorithmes selon le niveau de consommation des ressources.

5 Gestion et organisation du travail

a. Création de la classe Redi cube

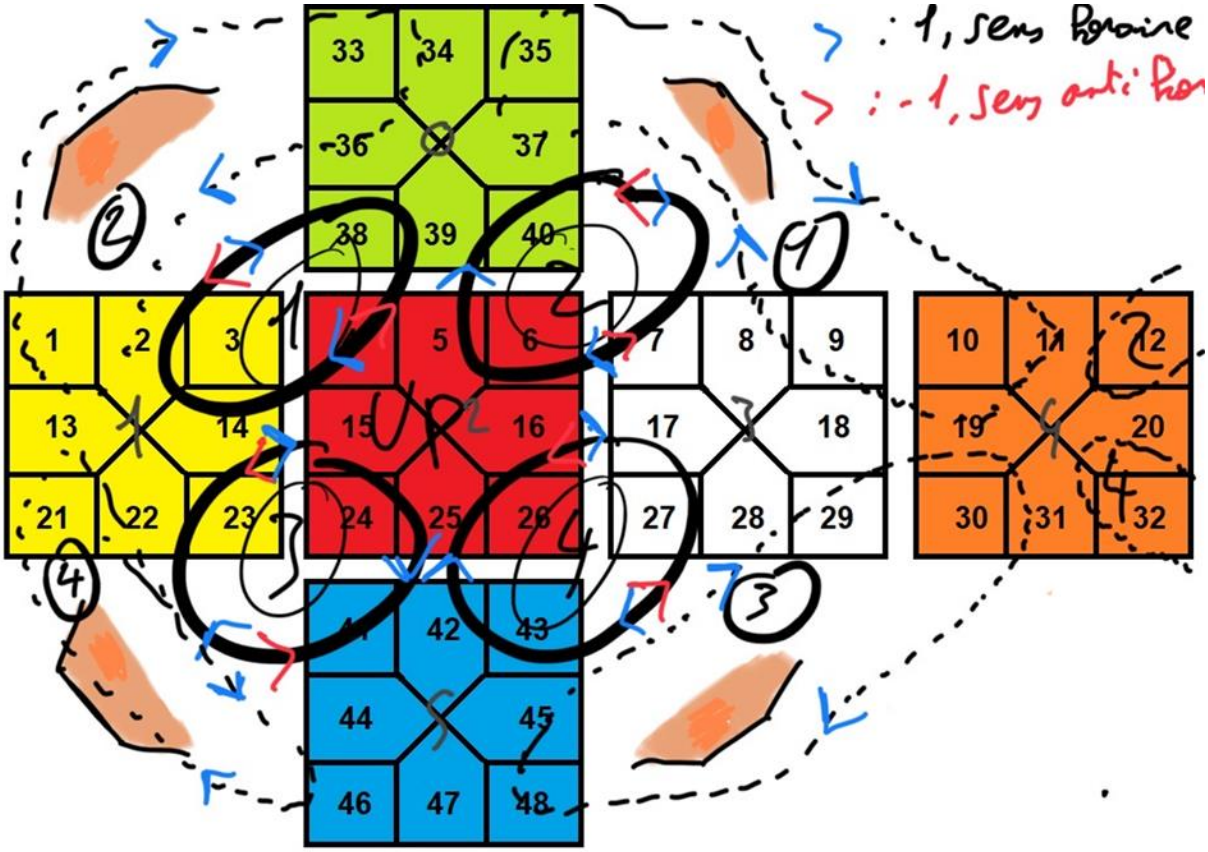
Au préalable de la création de notre classe, on a déjà réalisé un travail d'analyse du sujet, à travers les études déjà réalisées sur le Rubik's cube : la manière de représenter le cube, ses méthodes de résolution…

Une fois ce travail fait, et en attendant la réception de nos Redis cubes, on a démarré par la création d'une classe **RediCube** sur Python, afin de pouvoir créer un Redi cube et toutes les propriétés et fonctionnalités qui en découlent : Réalisation des mouvements, affichage en 2D sur la console Python, mélanger aléatoirement le Redi cube…



Pour l'affichage en 2D du Redi Cube sur Python, nous nous sommes basés sur cette représentation du cube.

Afin de pouvoir créer les mouvements sans avoir encore reçu nos exemplaires, nous nous sommes appuyés sur des représentations plan, afin de chercher à comprendre comment les couronnes, arrêtes et sommets s'articulaient comme le montre le schéma suivant.



Une fois notre matériel arrivé, nous avons dû procéder à quelques correctifs sur comment s'effectuaient les mouvements, avant de pouvoir s'appuyer sur une classe Redi cube fonctionnelle et fidèle à la réalité.

À partir de là, nous avons travaillé sur 2 aspects en parallèle, et ce jusqu'à la fin du projet : La résolution du Redi cube et l'interface graphique.

b. Résolution d'un Redi cube

Avant d'élaborer nos premiers algorithmes de résolution, on a d'abord dû créer une fonction devant pondérer le niveau de résolubilité d'un Redi cube, indiquer si le cube est plus ou moins compliqué à résoudre, c'est notre fonction de coût. Cette fonction attribue un score à un Redi cube, plus le score étant élevée, plus le Redi cube étant proche d'être résolu. Basique au départ, elle donnait un point par arête ou coin de la bonne couleur, puis elle s'est au fur et à mesure complexifiée afin d'apporter en précision dans nos prises de décision sur les procédés de résolution.

Afin de réussir à résoudre un Redi cube, on se dirigeait logiquement sur un algorithme d'arbre, devant vérifier plusieurs possibilités à la chaîne, jusqu'à la résolution complète. D'abord un arbre en profondeur, puis au vu du nombre de configurations à explorer (16 coûts possibles, puissance n) nous sommes vite passés sur un arbre en largeur. Ce type d'arbre nous permettait de naviguer profondeur par profondeur, et éviter de s'enfoncer systématiquement à des grandes profondeurs de l'arbre et donc beaucoup de possibilités à explorer.

Malgré ce nouveau système de parcours de l'arbre, beaucoup trop de nœuds sont explorés, c'est là qu'intervient les méthodes d'élagage, s'appuyant sur la fonction de coût développée précédemment. Afin d'aller plus vite, nous sélectionnons un nombre **n** de configurations à chaque profondeur, s'appuyant par exemple sur le top **n** de ces configurations (les n meilleurs

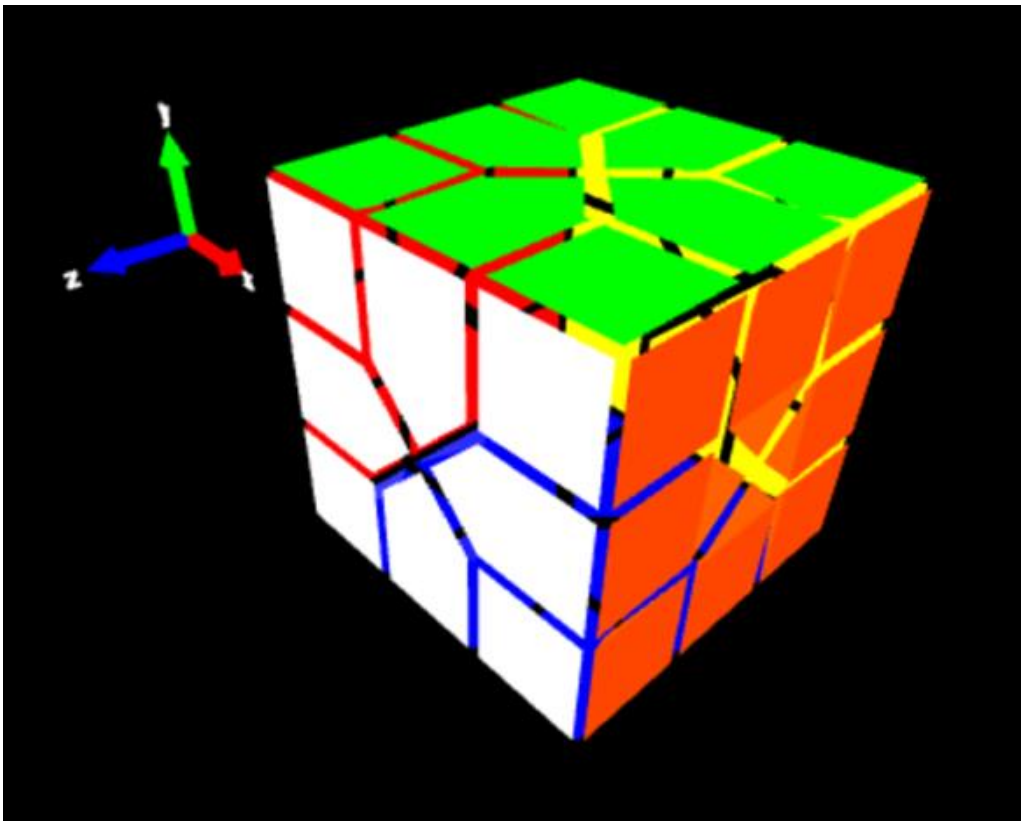
d’après notre fonction de coût). Des premiers résultats concluants sont déjà atteignables, mais très vite l’algorithme tourne en rond dans la résolution de Redi cubes plus difficiles. En effet, en sélectionnant un nombre de configurations par paliers, il est possible de couper les branches, menant à la résolution du cube. La résolution peut passer par des stagnations voire des baisses du score du cube, « reculer pour ensuite mieux avancer ».

C’est là enfin que la logique du Rollback se développe dans nos algorithmes. Afin de par ne pas couper des branches qui sont potentiellement les bonnes, on s’appuie sur cette pensée : On élague notre arbre à chaque niveau de profondeur en gardant toujours un top n des configurations, mais, si par exemple 4 niveaux plus bas, un nœud fils n’a pas de meilleur score que son nœud père, on peut alors considérer qu’il y a une stagnation, c’est-à-dire que cette branche ne mène à rien. On décide donc de l’élaguer et de revenir au niveau du nœud père, afin de tester un autre nœud qu’on avait élagué précédemment. On revient donc sur nos pas, et cette méthode permet de limiter donc un arbre qui s’enfoncerait indéfiniment sans jamais trouver de solutions.

Ce dernier algorithme sera celui utilisé pour la recherche du nombre de dieu.

c. Interface graphique

Tout en travaillant sur les méthodes de résolution, il était important de réussir à aller plus loin qu’une représentation 2D des cubes, pas très visuel avec des mouvements difficiles à observer. On a alors créé une interface graphique du Redi cube en 3D, dynamique, l’utilisateur peut bouger directement avec la souris le cube, et à chaque mouvement nécessaire à la résolution, nous pouvons observer que les arêtes et les coins concernés changent de couleur en temps réel ou avec une latence pour permettre à l’utilisateur de bien comprendre les étapes de résolution du cube. Cette interface graphique a été réalisé grâce à la librairie VPython.



Un visuel du Redi Cube à partir de l’interface graphique

d. Recherche du nombre de dieu

La manière la plus intuitive pour trouver ce nombre serait de résoudre toutes les configurations du Redi cubes possibles, et de conclure à la fin, quel serait le nombre de coup maximal utilisé. Cette méthode étant très compliquée à mettre en place et nécessitant d'énormes ressources, il a fallu travailler sur la mise en place d'échantillons représentatifs, afin d'estimer ce nombre.

Résoudre un nombre n de Redi cubes aléatoires, aurait été peu pertinent. En effet, comme observé précédemment, il y a des configurations plus difficiles que d'autres. Les configurations faciles, se résolvent plus rapidement, et en peu de coût. Les plus difficiles, prennent plus de temps de calcul, car elles nécessitent plus de coût à effectuer, et donc plus de nœuds à explorer sur l'arbre. On peut donc améliorer notre échantillon, en s'appuyant davantage sur ces Redi cubes plus difficiles, et qui nécessiteront le nombre de coût maximum pour être résolu, et donc ce fameux nombre de Dieu. On a alors créé plusieurs jeux de données de configurations de Redi cube possibles, variant par la difficulté de ces derniers, et de tenter d'en résoudre un maximum, notamment dans le dataset le plus complexe, afin de tenter de trouver le nombre de Dieu.

6 Résultats

Afin d'approcher avec la plus grande précision possible le nombre de dieux, nous avons créé 3 datasets de difficulté croissante du Redi Cube que nous avons tenté de résoudre en une vingtaine d'heures chacun.

Voici les résultats obtenus :

```
In [20]: AnalyseDataset(2)
Out[20]: (1000, 0, 5, 2.727, 57.175177392244336)

In [21]: AnalyseDataset(1)
Out[21]: (403, 19, 19, 5.950520833333333, 128.8395408745855)

In [22]: AnalyseDataset(0)
Out[22]: (92, 51, 25, 11.341463414634147, 287.54681882625675)
```

Les Tuples sont composés de la manière suivante :

- Nombre de Redi cube analysés
- Nombre de Redi cube non résolus
- Coût maximum obtenu extrait du dataset
- Coût moyen dans le dataset
- Temps moyen de calcul par Redi cube

Les points positifs que nous pouvons noté de ce training :

- Le système de poids mis en place pour résoudre un Redi cube semble cohérent au vu des résultats obtenus, que cela soit en termes de temps, du coût maximum ainsi que du nombre de Redi cube résolus.
- Les deadlocks semblent résolus car les Redi cube non résolus viennent du nombre de nœuds maximum parcourables par Redi cube définis dans le cadre de ce training.

Quelques points négatifs peuvent également être mise en avant :

- Le temp de compilation sans threading est relativement important.
- Le coût maximum d'un Redi cube (25) dépasse le nombre de dieux théoriques (19); on peut donc en déduire que l'algorithme choisi est encore améliorable.

7 Conclusion

Ce projet fut très enrichissant aussi bien sur le plan humain que sur le plan technique. Il fut l'occasion de débats passionnants pour mettre au point la stratégie et l'algorithme de résolution de notre Redi Cube.

Nous sommes toutefois conscients que le nombre de Redi cube résolus pour obtenir nos interprétations est relativement faible, bien que les résultats obtenus semblent globalement satisfaisant. Afin de conforter nos conclusions, nous envisageons de réaliser quelques expérimentations que nous présenterons lors de la soutenance.

8 Bibliographie

[1]. Comment résoudre le Redi Cube (YouTube)
[03/01/2018] https://www.youtube.com/watch?v=zw7UZcqqsgA&ab_channel=ValentinoCube