Université de Montpellier

M1 informatique

M1 Cursus Master en Ingénierie informatique

Modélisation et résolution d'un Rubik's Cube

RubiCube

Rapport de projet

Étudiants :
Julie CAILLER
Thomas GEORGES
Amandine PAILLARD
Florent TORNIL

Encadrant :
Mickaël MONTASSIER

Rédigé le :
30 avril 2021

Remerciements

Avant d'entamer ce rapport, nous souhaiterions remercier quelques personnes, sans elles ce projet ne serait pas dans son état actuel.

En premier lieu, nous remercions notre encadrant Mickaël Montassier pour avoir accepté de s'occuper de notre projet et de nous avoir guidé tout au long de cette année de travail. Nous remercions également Anne-Élisabeth Baert ainsi que Mathieu Lafourcade pour nous avoir encadré dans les cadres respectifs du projet annuel CMI puis du TER. Nous pensons également à Éric Bourreau qui nous a accompagné dans nos démarches auprès de la DSI. Enfin de manière plus globale, nous souhaiterions adresser nos remerciements à l'équipe pédagogique du département informatique de la Faculté des Sciences de l'Université de Montpellier.

Enfin, pour leurs contributions à la thématique de recherche du Rubik's Cube nous souhaitons remercier Herbert Kociemba, Daniel Walton, Daniel Shiffman, Ben Katz, Jared et bien d'autres. Leur travail a pu nous permettre de mieux comprendre certaines notions et d'avancer dans notre projet.

Résumé

Dans le cadre du projet Travaux d'Étude et de Recherche de la première année de master informatique et du projet annuel du Cursus Master en Ingénierie (CMI), nous avons décidé de travailler sur le Rubik's Cube. Ce rapport présente l'ensemble du travail réalisé.

Il explicite ce qu'est un Rubik's Cube et quelles sont les notations en rigueur pour cet objet avant d'étudier les thématiques de recherche qui y sont adossées. Une fois les notions nécessaires présentées, la solution apportée est décrite : il s'agit d'une application web permettant aux utilisateurs de résoudre leur Rubik's Cube quelque soit la taille de ces derniers. Pour réaliser cela, la répartition suivante a été décidée : les interactions avec l'utilisateur et la résolution du Rubik's Cube.

Concernant les interactions avec l'utilisateur, il a fallu se concentrer sur l'acquisition du cube et sa représentation auprès de ce dernier. L'acquisition choisie est composée de plusieurs étapes : calibration, détection et classification des couleurs du cube. La présentation du cube ainsi que celle des mouvements de résolution se font part une représentation tridimensionnelle dont le modèle s'anime et se met à jour selon les mouvements de résolution du cube.

Concernant la partie de résolution du cube, après avoir présenté un bref état de l'art, le rapport présente les méthodes choisies pour ce projet. Le couche par couche a été implémenté tandis que des algorithmes libres de droits pour les méthodes de *Kociemba*, *Iterative Deepening A* et de $N \times N \times N$ ont été récupérés et ajoutés à l'API.

Ainsi ce TER a permis la réalisation d'une application web permettant aux utilisateurs de résoudre leur Rubik's Cube. Cette application propose différentes méthodes de résolution et prend en compte différentes tailles de Rubik's Cube. Par ailleurs, de part la séparation dans le développement des parties *interface* et *algorithmique*, l'application peut être facilement enrichie de nouvelles méthodes de résolution. Enfin, l'accent a été mis sur la facilitation pour l'utilisateur de représenter son cube. Pour cela une méthode de détection des couleurs du cube couplée à une classification ont été utilisées. Enfin, la reconnaissance vocale autorise l'utilisateur de naviguer plus aisément dans l'application.

Table des matières

Partie 1

Présentation du projet

1.1 Présentation générale

Ce projet, réalisé dans le cadre du projet Travaux d'Étude et de Recherche de la première année de Master Informatique et du projet annuel du Cursus Master en Ingénierie (CMI) a pour but la mise en place d'une application permettant l'acquisition, la modélisation et la résolution de Rubik's Cubes physiques.

Dans sa forme classique, le Rubik's Cube est un casse-tête géométrique à trois dimensions composé extérieurement de vingt-six éléments et de six couleurs prenant la forme d'un cube de dimensions $3 \times 3 \times 3$. À première vue, chacun de ces éléments semble être lui-même un cube pouvant se déplacer sur toutes les faces du Rubik's Cube et parait libre de toute fixation sans pour autant se détacher de l'ensemble. Au centre du cube, se cache un système d'axes permettant de déplacer les éléments, dont le mécanisme a été breveté par son auteur, Ernő Rubik.

L'intérêt d'un tel casse-tête est sa résolution. Initialement, chacune des six faces du cube est identifiée par sa couleur unie et l'objectif est, après avoir mélangé les six faces, de manipuler le cube afin de lui rendre son apparence d'origine, c'est à dire avec les six faces de couleurs unies. Il existe de nombreuses façons de résoudre un Rubik's Cube. Les algorithmes sont variés, allant de la résolution classique et mécanique à la résolution optimale en 20 coups ou moins.

FIGURE 1.1 – Un Rubik's Cube $3 \times 3 \times 3$

De plus amples informations sur les Rubik's Cube sont disponibles dans la partie 2, dédiée à ce sujet.

1.2 Objectifs

La finalité de ce projet est la création d'une application permettant à un utilisateur de résoudre son propre Rubik's Cube quelle que soit sa taille. Cette application doit être accessible à tous et ne

pas nécessiter de connaissances approfondies sur le Rubik's Cube. Ce projet peut ainsi se diviser en deux parties : les interactions avec l'utilisateur et la résolution du Rubik's Cube.

1.2.1 Interactions avec l'utilisateur

Cette partie du projet se concentre sur le passage de Rubik's Cube physiques à un modèle de cube virtuel ainsi que sur la représentation d'une solution permettant la résolution du Rubik's Cube donné. Les efforts de cette partie s'organisent autour de la facilité d'acquisition des Rubik's Cubes en tenant compte des diverses tailles existantes et met l'accent sur sa simplification et son caractère intuitif pour l'utilisateur.

De fait, la numérisation de Rubik's Cubes doit se faire de la manière la plus simple et intuitive possible. Une fois cette étape accomplie, et une suite de coups récupérée, l'utilisateur pourra lancer une résolution guidée.

1.2.2 Résolution

L'objectif principal de cette partie est la réalisation d'une API permettant de résoudre un cube donné par la partie interaction avec l'utilisateur.

Cela passe par le recensement des principaux algorithmes de résolution du Rubik's Cube existants, la comparaison de leur efficacité et la sélection de ceux qui sembleront les plus pertinents à incorporer à l'API.

Il est important de noter que de nombreux critères peuvent être pris en compte pour choisir une méthode de résolution la plus efficace possible. Une concentration particulière sera apportée sur le fait d'avoir le moins de coups possibles tout en gardant un temps de calcul raisonnable du point de vue utilisateur.

Dans la suite de ce rapport seront décrits les algorithmes existants pour résoudre un Rubik's Cube ainsi que les implémentations les plus efficaces connues et utilisées à ce jour avant de présenter ceux qui ont été ajoutés à l'API.

1.3 Organisation

1.3.1 Planning prévisionnel et répartition des tâches

Pour mener à bien ce projet, deux équipes différentes ont travaillé sur les aspects principaux de l'application :

— Interactions avec l'utilisateur : Amandine Paillard et Florent Tornil.
— Résolution : Julie Cailler et Thomas Georges

La répartition des tâches ainsi que leur ordonnancement dans le temps peuvent être consultés en figure 1.2.

1.3.2 Outils utilisés

Dans le cadre de la gestion d'équipe, des outils tels que Trello ou GitLab ont pu être utilisés. Le lien du git est le suivant : `https://gitlab.com/amapai/rubik`. Des langages de programmation et API différents ont été utilisés selon la partie du projet. Leurs choix sont expliqués dans la suite de ce rapport.

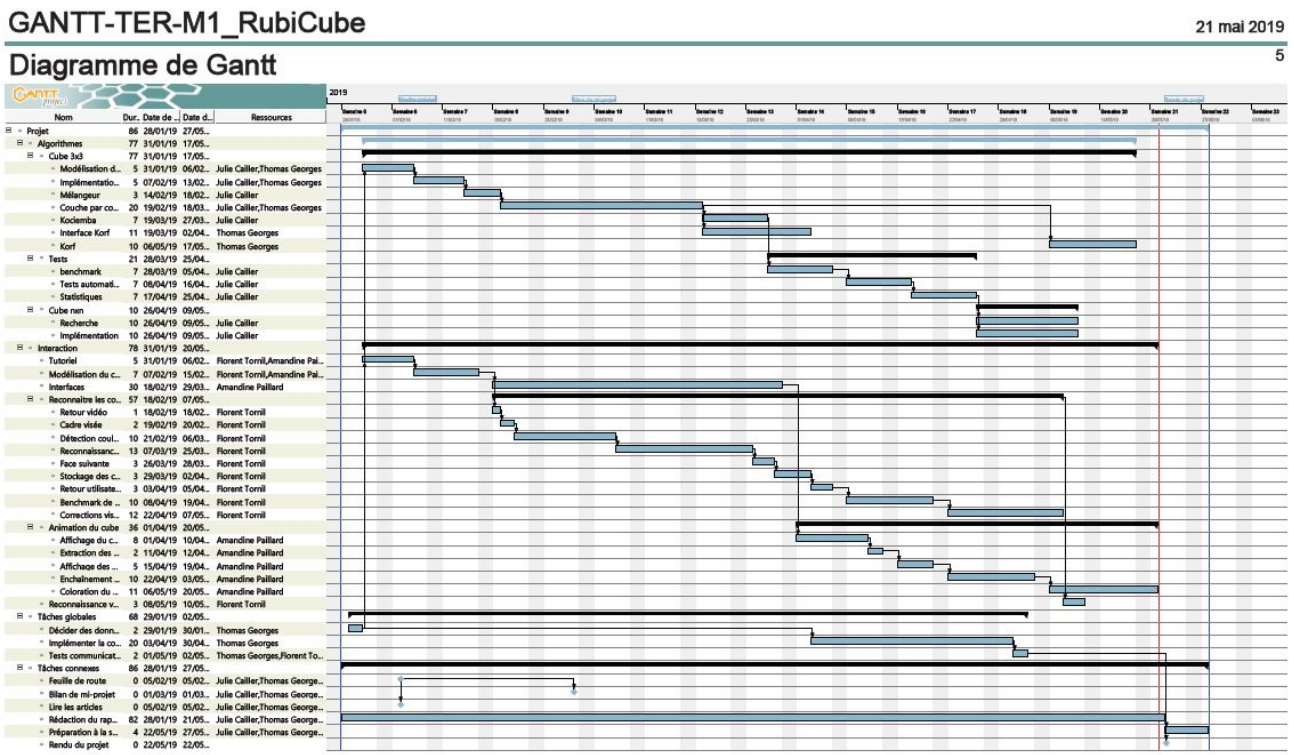

FIGURE 1.2 – Diagramme de Gantt effectif

Partie 2

Introduction au domaine

Avant d'aller plus loin, une présentation du Rubik's Cube et des notions le concernant semble importante.

Cette partie du rapport va présenter plus en détail le Rubik's Cube ainsi que l'univers qui l'entoure, afin de pouvoir mieux comprendre les tenants et aboutissants de ce projet.

2.1 Le Rubik's cube

2.1.1 Présentation du casse-tête

Le Rubik's Cube est un casse-tête tridimensionnel originellement composé de vingt-six petits cubes assemblés de façon à former un cube plus gros. Il possède six faces possédant chacune une couleur différente : blanche, verte, rouge, bleue, orange et jaune.

Le Rubik's Cube peut être manipulé grâce à un mécanisme interne permettant de faire bouger les rangées et les colonnes. Le but est, une fois le Rubik's Cube mélangé, de le ramener à son état initial, à savoir une seule couleur sur chaque face.

Depuis sa création, de nombreuses déclinaisons ont vu le jour, tant au niveau des couleurs (par exemple, le cube dit « japonais »), que de la taille ou de la forme.

FIGURE 2.1 – Un Rubik's Cube $3 \times 3 \times 3$ dans un état mélangé

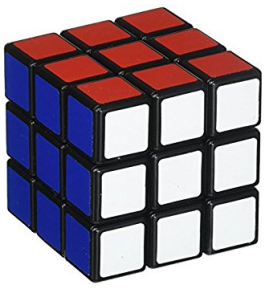

FIGURE 2.2 – Un Rubik's Cube $3 \times 3 \times 3$ dans l'état résolu

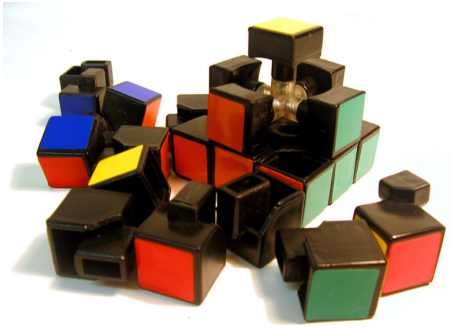

FIGURE 2.3 – Mécanisme interne d'un Rubik's Cube $3 \times 3 \times 3$

FIGURE 2.4 – Un Rubik's Cube $4 \times 4 \times 4$

FIGURE 2.5 – Des Rubik's Cubes de toutes formes

2.1.2 Historique

Inventé en 1974 par Ernő Rubik, architecte et professeur de design hongrois, le Rubik's Cube est un casse-tête cubique qui s'est répandu à travers le monde au cours des années 1980, avec plus de cent milions d'exemplaires vendus entre 1980 et 1982.

Des compétitions ont alors commencé à voir le jour un peu partout dans le monde, ce qui amena en 2004 à la création de la World Cube Association.

Lors de ces compétitions, des cubistes (pratiquants du Rubik's Cube) s'affrontent pour résoudre le Rubik's Cube le plus rapidement possible.

Les compétitions peuvent concerner tous les types de Rubik's Cube. Il existe également des variantes, comme les compétitions de résolutions de Rubik's Cube à l'aveugle ou avec les pieds.

À ce jour, le record de vitesse de résolution d'un Rubik's Cube 3x3x3 est détenu depuis le 24 novembre 2018 par Yusheng Du, qui l'a résolu en 3'47 secondes.

2.1.3 Vocabulaire

Un Rubik's Cube est composé de :

Cubics : au nombre de vingt-six, ils regroupent la notion de coin, d'arête et de centre. Ils sont ce qui compose le Rubik's Cube.

Faces : au nombre de six, elles sont définies par la couleur de leur centre. Elles sont composées de trois rangées de trois cubics.

Centre : au nombre de six, il s'agit des cubics centraux de chaque face. Il est impossible de les changer de place.

Coin : au nombre de huit, ce sont des cubics définis par leurs trois couleurs.

Arêtes : au nombre de douze, ce sont des cubics définis par leurs deux couleurs.

Dans la suite du rapport, par **modèle d'un Rubik's Cube**, il est sous-entendu une représentation numérique du cube et de son espace dans une configuration précise.

2.2 Concepts mathématiques

2.2.1 Nombre de combinaisons

Il y a plus de quarante-trois trillions de positions différentes pour le Rubik's Cube.

- Il y a douze arêtes. Chacune des arêtes possède deux orientations possibles. De plus, les orientations des onze autres arêtes définissent l'orientation de la dernière. Il y a donc 2^{11} possibilités d'orientation des arêtes.
- Il y a huit coins. Chaque coin possède trois orientations possibles. Comme précédemment, l'orientation du dernier coin est déterminée par les orientations des sept autres. Il y a donc 3^7 possibilités.
- Il y a douze arêtes. Les arêtes pouvant s'inverser mutuellement, il y a donc 12! possibilités de positionnement.
- Les huit coins étant également interchangeables, cela donne 8! possibilités.
- Cependant, il peut y avoir à un problème dit « de parité » : il est impossible d'échanger seulement deux coins ou deux arêtes. La position des arêtes et des premiers coins fixe donc la position des deux derniers coins et il faut alors diviser le résultat par deux.

Ce qui donne $\frac{2^{11}\times 3^7\times 12!\times 8!}{2} = 8!\times 3^7\times 12!\times 2^{10} = 43\ 252\ 003\ 274\ 489\ 856\ 000$ de combinaisons possibles pour le Rubik's Cube.

Distance	Nombre de positions
0	1
1	18
2	243
3	3,240
4	43,239
5	574,908
6	7,618,438
7	100,803,036
8	1,332,343,288
9	17,596,479,795
10	232,248,063,316
11	3,063,288,809,012
12	40,374,425,656,248
13	531,653,418,284,628
14	6,989,320,578,825,358
15	91,365,146,187,124,313
16	environ 1,100,000,000,000,000,000
17	environ 12,000,000,000,000,000,000
18	environ 29,000,000,000,000,000,000
19	environ 1,500,000,000,000,000,000
20	environ 490,000,000

Nombre de positions atteignables en fonction du nombre de mouvements.

2.2.2 Nombre de Dieu

Le « *God's number* » ou *nombre de dieu* est le nombre de mouvements maximum pour résoudre le Rubik's Cube depuis n'importe quelle position. Il est le résultat de l'*algorithme de Dieu*.

En 1980, une première estimation basse fût trouvée pour le Rubik's Cube en comptant le nombre de combinaisons possibles en effectuant dix-sept mouvements. Ce nombre étant inférieur

au nombre de combinaisons possibles du cube, il a ainsi été démontré qu'il fallait au minimum dix-huit coups pour résoudre le cube.
Ce nombre fût augmenté à vingt en 1995 par Michael Reid qui prouva que la position du *superflip* nécessitait au moins vingt mouvements.

Pour ce qui est de la majoration, le nombre de cinquante-deux fût établi par Morwen Thistlethwaite grâce à une résolution informatique en 1981. Il n'a depuis cessé de décroître jusqu'en 2010, lorsque Tomas Rokicki, Herbert Kociemba, Morley Davidson et John Dethridge ont prouvé que le nombre de Dieu est au plus vingt. [1]

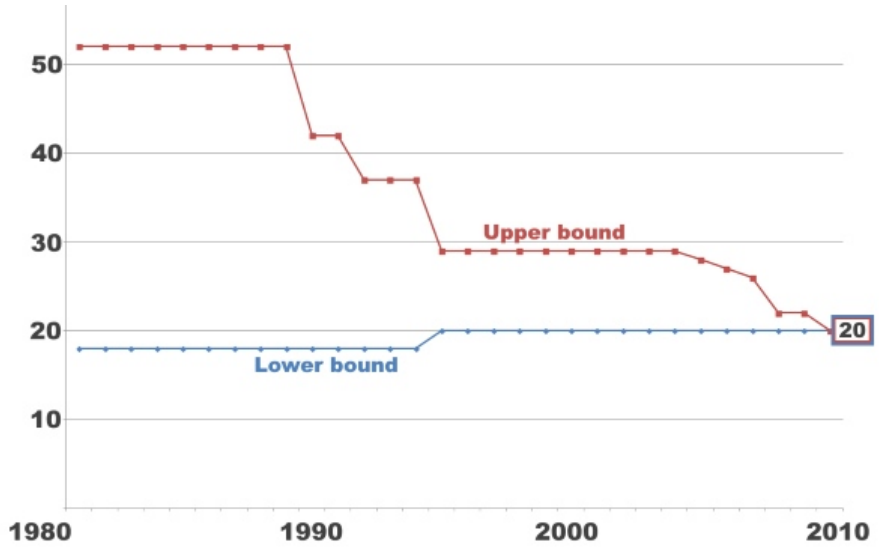

FIGURE 2.6 – Historique du nombre de coups minimum et maximum pour résoudre un Rubik's Cube

2.2.3 Notations

Il existe des notations particulières en matière d'algorithmes de Rubik's Cube. Ces notations sont à considérer tenant compte du fait que la face avant est celle que l'utilisateur a en face de lui lorsqu'il regarde le Rubik's Cube.

2x2x2 et 3x3x3

Pour les cubes 2x2x2 et 3x3x3, la liste des mouvements basiques est la suivante :

— **F (Front)** – Avant
— **R (Right)** – Droite
— **U (Up)** – Dessus
— **B (Back)** – Arrière
— **L (Left)** – Gauche
— **D (Down)** – Dessous

Effectuer l'un de ces mouvements correspond à tourner la face correspondante d'un quart de tour dans le sens horaire.

Une apostrophe (') peut être ajoutée après une lettre pour signifier une rotation dans le sens anti-horaire.
Le chiffre deux (2) peut être ajouté après une lettre pour signifier que l'on effectue deux fois le mouvement.

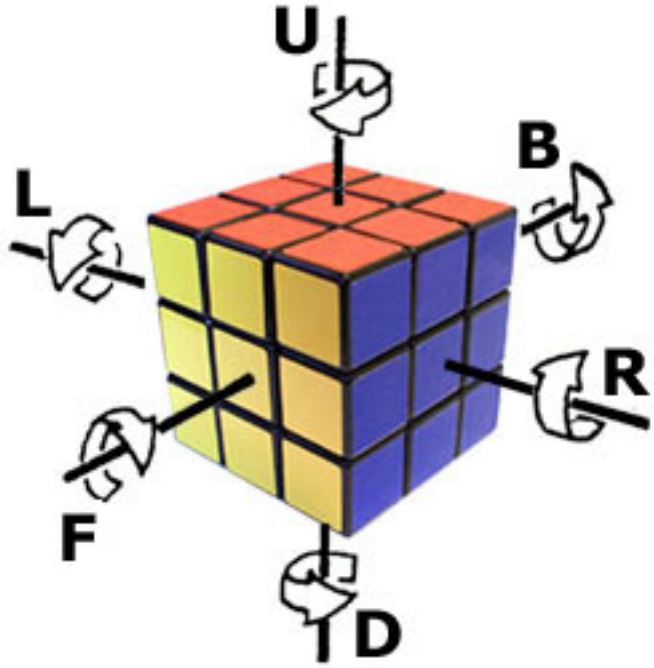

FIGURE 2.7 – Schéma représentant les coups basiques du Rubik's Cube3x3x3, en considérant que la face jaune est face à l'utilisateur

NxNxN

Pour les Rubik's Cubes d'arité supérieure au 3x3x3 (4x4x4, 5x5x5, etc.), la notation diffère quelque peu.
En effet, il faut dès lors indiquer combien de couches l'on souhaite bouger et la face à faire pivoter. On ajoute de plus la lettre w à la fin du coup pour signifier que l'on prend au moins les deux première couches.
Ainsi, les mouvements sont les mêmes que pour un cube 2x2x2 ou 3x3x3 si l'on souhaite uniquement faire pivoter la face extérieure.
En revanche, pour en faire pivoter deux, il faut écrire $2 + Face + w$ (ou simplement $Face + w$ pour le cas 2)
Pour en faire pivoter 3, il faut écrire $3 + Face + w$, et ainsi jusqu'à l'arité maximum du cube (même si dans les faits on ne dépasse jamais $\frac{arite}{2}$).
(Pour $Face$ dans $[U, R, F, D, L, B]$).

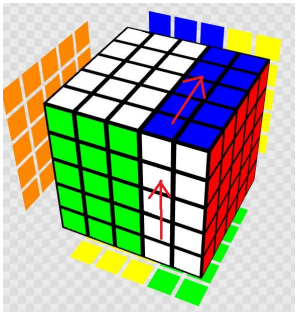

FIGURE 2.8 – Ce mouvement correspond à Rw (ou $2Rw$)

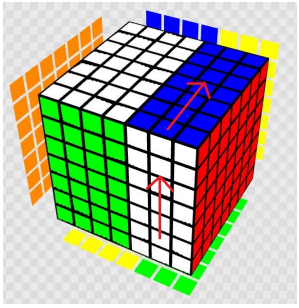

FIGURE 2.9 – Ce mouvement correspond à $3Rw$

Partie 3

Premier objectif : Acquisition de Rubik's Cubes et présentation de solutions

Afin de proposer à l'utilisateur une résolution guidée de son Rubik's Cube, il est nécessaire d'obtenir les informations concernant l'état actuel de ce cube.

Cette partie du rapport s'intéresse tout d'abord aux méthodes existantes afin d'obtenir de telles informations couplées à la présentation à l'utilisateur de solutions à son Rubik's Cube. Les sous-parties *acquisition du modèle* ainsi que la *présentation à l'utilisateur de la résolution* de ce même Rubik's Cube permettront de présenter les choix effectués et leurs apports en ce qui concerne le passage de Rubik's Cubes physiques à leur modèle numérique.

3.1 Positionnement par rapport à l'état de l'art

3.1.1 Analyse de l'existant

Lorsqu'une personne se retrouve face à un Rubik's Cube mélangé, sa première idée est de tourner les faces de façon plus ou moins aléatoire en espérant arriver à la configuration initiale. Au bout d'un certain temps, en se rendant compte que cela ne fonctionne pas, cette personne va très probablement abandonner en jetant son cube ou bien se diriger vers son ordinateur et utiliser son moteur de recherche favori afin de rechercher les termes « Comment résoudre son Rubik's Cube ». Les résultats obtenus sont alors nombreux, allant des guides écrits aux vidéos. Si la personne est déterminée, elle ira suivre un de ces tutoriels afin de résoudre son cube. Cependant, avec une recherche plus poussée, il est possible de trouver de nombreux solveurs de Rubik's Cube en ligne. Une analyse plus détaillée de ces tutoriels et solveurs est disponible en annexe A.

Bien que les tutoriels permettent d'apprendre à résoudre un Rubik's Cube, ils ne proposent qu'une solution générale et ne tiennent pas compte de la configuration initiale du cube de l'utilisateur. Ils ne seront donc pas plus détaillés dans ce rapport.

Les solveurs en ligne existants présentent un problème majeur : l'acquisition du cube. En effet, colorer chaque face de cubic une par une demande un minimum de soixante clics sur un Rubik's Cube 3x3x3 (cinquante-quatre faces de cubic et six changements de couleur), en suivant précisément son cube physique sans se tromper. Cette solution est loin d'être efficace et de ce fait cette partie du projet met l'accent sur cette acquisition et sa simplification. L'aspect 3D du cube lors de son affichage à l'écran est également un avantage considérable dans la facilité de lecture des différents mouvements qui n'est pas à négliger.

Définir un cube virtuel et exécuter des algorithmes de résolution n'est pas réservé exclusivement aux ordinateurs. Un appareil mobile comme un téléphone portable peut également satisfaire ces objectifs. Une analyse détaillée des applications mobiles existantes est disponible en annexe B. La plupart des applications étudiées permettent d'interagir avec le modèle virtuel en touchant l'écran ; il faut cependant être délicat car le modèle est très souvent sensible. Peu d'entre elles présentent un guidage « pas-à-pas » de la résolution du Rubik's Cube mais plutôt des modes d'emploi ou la théorie des méthodes existantes. Seule une application parmi celles étudiées permet de photographier son Rubik's Cube. Enfin, une question demeure quant à la façon dont l'utilisateur va pouvoir photographier son Rubik's Cube et tenir son appareil mobile pendant qu'il le résout.

3.1.2 Définition des besoins

L'analyse de l'existant a permis de définir quels sont les éléments importants pour l'application. Pour rappel, cette application doit permettre à un utilisateur d'acquérir son cube de la façon la plus simple possible afin de pouvoir suivre une suite de coups résolvant son cube.

Pour l'acquisition du cube, en tenant compte de ce qui se fait et de ce qui semble le plus pratique, il a été décidé que l'application prendrait des photos des différentes faces du Rubik's Cube de l'utilisateur afin de détecter automatiquement les couleurs de chaque face de cubic. Une phase de calibration des couleurs est également envisagée afin d'améliorer cette détection.

En ce qui concerne le problème soulevé de tenir le cube tout en prenant une photo, la solution retenue dans le cadre de ce projet est d'utiliser des commandes vocales permettant d'effectuer la prise de vue sans avoir besoin d'appuyer sur un bouton.

Pour la présentation d'une solution, il est souhaitable que l'application puisse afficher une représentation en trois dimensions du Rubik's Cube de l'utilisateur afin que ce dernier puisse suivre les mouvements à effectuer selon l'animation d'un cube semblable au sien en perspective et couleurs. La solution de commandes vocales est également envisagée sur ce dernier point afin de contrôler l'animation.

Un dernier problème se pose alors : que faire si l'utilisateur se trompe de mouvement malgré les précautions prises ? Dans ce cas, utiliser un traitement vidéo peut permettre de détecter le mouvement effectué par l'utilisateur et lui proposer une correction sans avoir à tout recommencer.

3.1.3 Choix de l'outil

Afin de réaliser cette application, il a été nécessaire d'étudier les différentes technologies (application mobile, logiciel (`Java`, `C++`, etc ...), application web, etc ...) et d'en choisir une selon les besoins précédemment définis.

Le premier critère de recherche fut l'accessibilité de l'application. En effet, il est souhaitable d'avoir une application facile d'accès et d'utilisation pour un utilisateur non-formé en informatique. Pour cela, en tenant compte des solutions existantes étudiées lors de l'analyse de l'existant, il a été décidé que l'application réalisée serait une application web, trouvable et utilisable facilement sur un navigateur de recherche. Afin que l'application soit la plus facile possible à prendre en main pour les utilisateurs le choix de faire une *Single Page Application* s'est vite imposé : de cette manière les utilisateurs n'ont qu'à se laisser guider par l'application, sans possibilité de se tromper.

De nombreuses technologies existent pour réaliser une application web. En plus d'utiliser `jQuery`, `Ajax` et `Bootstrap`, les recherches se sont alors tournées vers des solutions permettant

d'accéder facilement à la webcam d'un ordinateur, présenter un retour 3D à l'utilisateur ainsi que la possibilité d'intégrer des commandes vocales facilement.

`Processing` est un logiciel et un langage éducatif dans le contexte des arts visuels. Publié en 2001, `Processing` est gratuit, *open-source* et disponible sur GNU/Linux, Mac OS X, Windows et même Android ou ARM. Depuis sa création, de nombreuses librairies étendant le programme principal ont vu le jour permettant de réaliser facilement des programmes interactifs en utilisant des technologies 2D, 3D, PDF ou encore SVG. Les traitements 2D et 3D sont optimisés grâce à une intégration de la bibliothèque OpenGL dans le cœur de `Processing`. Ce logiciel ayant été publié il y a maintenant dix-sept ans, sa documentation est complète et de nombreux ouvrages existent à son sujet publiés aussi bien par des chercheurs que des artistes ou des étudiants.

L'outil choisi : `P5.js` est une librairie `Javascript` reprenant les objectifs principaux de `Processing` tels que la facilité d'accès à la programmation aux artistes ou designers et de moderniser la programmation web.

En plus de l'affichage 3D géré par la librairie, cette dernière permet d'accéder facilement à la webcam de l'utilisateur afin d'obtenir les images nécessaires à l'acquisition du modèle d'un Rubik's Cube. De plus, de nombreuses librairies supplémentaires permettent de traiter les images obtenues, mais également de faire appel à l'API Google concernant la reconnaissance vocale.

3.2 Acquisition de modèle

L'objectif de cette partie est de passer du Rubik's Cube physique que possède l'utilisateur à un modèle numérique représentant la même configuration.

3.2.1 Informations concernant la stratégie adoptée

Avant de détailler la stratégie adoptée, il est nécessaire de préciser certaines notions relatives à la reconnaissance de couleurs sur un ordinateur :

- Une image numérique est composée de pixels, qui sont la plus petite unité visualisable sur un écran. Un élément d'une image, dans le cadre de ce projet : une face de cubic, est donc représenté par un ensemble de pixels.
- Chaque pixel est représenté par une valeur dite « RGB », comprenant trois composantes, qui représentent les intensités (entre 0 et 255) des trois couleurs principales : R pour la composante rouge (red en anglais), G pour la composante verte (green en anglais) et B pour la composante bleue (blue en anglais). Le pixel (255,0,0) apparaît donc rouge, tandis que le pixel (255,204,153) apparaît de couleur saumon.
- La distance entre deux couleurs correspond à leur distance dans un espace 3D pour lequel chaque couleur est représentée par un point de l'espace. Ce point a pour coordonnées les valeurs des composantes RGB (R pour la valeur en X, G pour la valeur en Y et B pour la valeur en Z).
- Un retour caméra désigne l'affichage à l'écran de l'image captée par la webcam de l'utilisateur. Ce retour présente un problème majeur désigné comme « effet miroir ». En effet, l'image captée par un appareil et affichée à l'identique représente la façon dont une personne perçoit la scène de face. Les notions de « gauche » et de « droite » sont donc inversées. Ceci possède à la fois des avantages et des inconvénients, selon l'utilisation de l'image. Il est cependant possible de conserver les directions en appliquant une transformation à l'image. Il suffit dans ce cas-là d'afficher l'image comme s'il s'agissait d'un miroir.

La stratégie adoptée afin d'obtenir une représentation numérique du cube possédé par l'utilisateur en conservant sa configuration existante est inspirée de l'implémentation réalisée par Ben Katz et Jared (`http://build-its-inprogress.blogspot.com/2018/03/the-rubiks-contraption.html`) dont le code source est disponible et open source sur GitHub (`https://github.com/dicarlo236/cube-solver`).

Cette stratégie consiste, pour chaque face de cubic, à déterminer une couleur « perçue » par la caméra, en faisant la moyenne de chaque composante RGB, pour chaque pixel représentant cette face de cubic.

Sur un Rubik's Cube $3 \times 3 \times 3$, il résulte de cette première étape un ensemble de cinquante-quatre couleurs sous la forme de composantes RGB. La seconde étape consiste, à partir de ces valeurs, à déterminer à quelle couleur du cube physique correspond la couleur perçue. Pour cela, la distance entre chaque couleur est calculée en utilisant la formule de la distance euclidienne $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. En fonction de cette distance, il est alors possible de déterminer quelles sont les neuf faces de cubic les plus proches de la couleur de référence afin de les interpréter comme même couleur physique.

La stratégie finalement implémentée est légèrement différente de celle présentée. En effet, sur un Rubik's Cube $3 \times 3 \times 3$, cette stratégie fonctionne très bien car il est possible de prendre comme couleur de référence la couleur des centres de chaque face qui est fixe. Cependant, en étendant la stratégie à des cubes de taille N, et notamment lorsque N est pair, il n'y a pas de centre de référence pour chaque couleur. C'est pour cela que dans ce projet, une phase de calibrage a été ajoutée afin d'obtenir ces couleurs de référence. Ces dernières seront classées lors de la phase de détection des couleurs des cubics.

Chacune de ces étapes est présentée par la suite, avec pour chacune, sa place et son utilité dans la stratégie globale, un rappel des données disponibles au départ, ainsi que les actions à effectuer par l'utilisateur et un algorithme détaillant la réalisation de l'étape et un détail des spécificités d'implémentation. À la fin de ces trois étapes, les résultats obtenus, leur fiabilité ainsi que certaines spécificités sont détaillés.

Afin de simplifier les actions de l'utilisateur, chaque clic nécessaire pour appliquer les actions de l'étape en cours est remplaçable par une commande vocale utilisant le mot clé « Suivant ». Ceci est possible grâce à l'API de reconnaissance vocale mise en place par Google.

3.2.2 Première étape : Calibration

L'étape de calibration est la première de la stratégie. Elle consiste à récupérer les couleurs de référence sur le cube de l'utilisateur. Pour cela, ce dernier se trouve face au retour caméra, affiché en mode miroir afin de conserver les notions de « gauche » et de « droite ». Un rectangle, qui sera par la suite appelé « cadre de visée » est superposé à ce retour caméra et des instructions lui indique quoi faire. Il s'agit à cette étape de trouver une face de cubic de la couleur demandée sur le cube physique et de la présenter à la caméra pour qu'elle prenne l'espace désigné par le cadre de visée. L'utilisateur n'a alors plus qu'à appuyer sur le bouton « Suivant » (ou dire « Suivant ») pour passer à la couleur suivante. Une fois les six couleurs calibrées, l'étape est terminée et l'utilisateur passe automatiquement à l'étape de détection.

À chaque fois que l'utilisateur appuie sur le bouton suivant, l'application détermine la couleur perçue dans l'espace désigné par le cadre de visée et conserve la valeur. La détermination de la couleur se fait selon l'algorithme 1 ci-contre.

Algorithme : Calibration d'une couleur

Données : img : l'image captée par la caméra de l'utilisateur
Résultat : la couleur perçue au format RGB
début

```
pixels = transformer img en tableau de pixels // fonction fournie par P5.js
moyenne_r = 0
moyenne_g = 0
moyenne_b = 0
compteur_pixel = 0
// Calcul de la somme des composantes RGB
pour chaque pixel de pixels faire
    si pixel est dans le cadre de visée alors
        moyenne_r += composante_rouge(pixel)
        moyenne_g += composante_verte(pixel)
        moyenne_b += composante_bleue(pixel)
        compteur_pixel ++
// Calcul de la moyenne
moyenne_r /= compteur_pixel
moyenne_g /= compteur_pixel
moyenne_b /= compteur_pixel
// Retour du résutat : nouvelle couleur
Retourner couleur(moyenne_r, moyenne_g, moyenne_b)
```

Algorithme 1 : Calibration d'une couleur

Lors de l'implémentation, le tableau de pixels n'a pas la forme d'un tableau contenant dans chaque case un pixel. Il prend la forme d'un tableau dans lequel un pixel est représenté par 4 cases, les trois premières représentant les trois composantes RGB et la quatrième une valeur de transparence qui n'est pas utile dans ce traitement. Une formule est alors disponible dans la documentation afin de déterminer un décalage dans le tableau pour obtenir les pixels correspondant à un certain point de l'image. De plus, la nouvelle couleur n'est pas retournée, mais directement stockée dans un objet de référence afin d'être accessible par la suite.

3.2.3 Deuxième étape : Détection

L'étape de détection est la deuxième de la stratégie. Elle consiste à récupérer les couleurs de chaque face de cubic du cube de l'utilisateur. Pour cela, ce dernier se trouve toujours face au retour caméra, affiché en mode miroir afin de conserver les notions de « gauche » et de « droite ». Une grille, qui sera par la suite appelée « grille de visée » est superposée à ce retour caméra et des instructions lui indiquent quoi faire. Il s'agit à cette étape de trouver la face dont le centre est de la couleur demandée sur le cube physique et de la présenter à la caméra pour qu'elle prenne l'espace désigné par la grille de visée avec chaque face de cubic dans la case correspondante sur cette grille. L'orientation de la face a alors une importance. Celle-ci est indiquée par les couleurs de la grille de visée, qui indique de quelle couleur doit être la face du côté correspondant. Pour les cubes pairs, il est indiqué à l'utilisateur comment « fixer » pour sa propre représentation les couleurs des faces. Ce dernier n'a alors plus qu'à appuyer sur le bouton « Suivant » pour passer à la face suivante. Une fois les couleurs de toutes les faces de cubic détectées, l'étape est terminée et l'utilisateur passe automatiquement à l'étape de classification.

À chaque fois que l'utilisateur appuie sur le bouton suivant, l'application détermine la couleur perçue dans chaque case de la grille de visée et conserve les valeurs calculées. La détection des couleurs se fait selon l'algorithme 2 ci-contre.

Algorithme : Détection des couleurs
Données : img : l'image captée par la caméra de l'utilisateur
Résultat : les couleurs perçues au format RGB pour chaque face de cubic de la face actuelle
début
    pixels = transformer img en tableau de pixels `// fonction fournie par P5.js`
    pour *chaque face de cubic f de la face actuelle* **faire**
        moyenne_r = 0
        moyenne_g = 0
        moyenne_b = 0
        compteur_pixel = 0
        `// Calcul de la somme des composantes RGB`
        pour *chaque pixel de pixels* **faire**
            si *pixel est dans la grille de visée et correspond à la case représentant f* **alors**
                moyenne_r += composante_rouge(pixel)
                moyenne_g += composante_verte(pixel)
                moyenne_b += composante_bleue(pixel)
                compteur_pixel ++
        `// Calcul de la moyenne`
        moyenne_r /= compteur_pixel
        moyenne_g /= compteur_pixel
        moyenne_b /= compteur_pixel
        couleur_percue = couleur(moyenne_r, moyenne_g, moyenne_b)
        `// Affectation de la valeur à la face de cubic correspondante`
        set_couleur_rgb(f, couleur_percue)

Algorithme 2 : Détection des couleurs de toutes les faces de cubic de la face actuelle

L'implémentation de cette phase est très similaire à celle donnée lors de l'étape de calibration. La différence entre les deux vient du fait que lors de la calibration, la couleur réelle du cubic présenté est connue et servira par la suite de repère tandis que lors de la phase de détection, les couleurs réelles des cubics préentés ne sont pas connues et seront classifiées lors de l'étape suivante à partir des couleurs de la calibration.

3.2.4 Troisème étape : Classification

L'étape de classification est la troisième et dernière étape de la stratégie. Elle consiste à passer des différentes couleurs perçues par la caméra à un ensemble de six couleurs réelles, correspondant aux couleurs du cube physique. L'utilisateur n'a rien à faire lors de cette étape. Elle se lance automatiquement lorsque les couleurs ont été détectées sur les six faces du Rubik's Cube. Il s'agit à cette étape de trouver de quelle couleur perçue lors de la calibration chaque couleur perçue lors de la phase de détection est la plus proche. Dans le cadre de ce projet, plusieurs idées ont été testées pour effectuer cette tâche, et la plus efficace au vu des données disponibles s'approche de la méthode des k plus proches voisins (`https://fr.wikipedia.org/wiki/Méthode_des_k_plus_proches_voisins`). Cela consiste à trouver quelles sont les k couleurs détectées les plus proches de chaque couleur récupérée lors de la calibration. Ceci se fait selon l'algorithme 3 ci-contre.

Algorithme : Classification des couleurs

Données : c_refs : couleurs de référence
c_dets : couleurs détectées, associées à la face de cubic correspondante
Résultat : la multitude de couleurs détectées est associée aux six couleurs de référence
début

```
// Calcul des distances
all_colors = []
pour chaque couleur détectée c_det faire
    pour chaque couleur de référence c_ref faire
        distance = calculer_distance(c_det, c_ref)
        obj = {d : distance, c : c_ref, o : c_det}
        ajouter obj dans all_colors
// Tri du tableau selon le critère fourni
trier(all_colors, distance croissante)
// Attribution des couleurs
pour chaque objet obj de all_colors dans l'ordre faire
    // attribuer à la couleur détectée la couleur de référence
       correspondante
    attribuer_couleur(obj['o'], obj['c'])
    // la couleur détectée a désormais une couleur de référence associée.
       Il est inutile de conserver les comparaisons aux autres couleurs de
       référence dans le tableau
    pour chaque objet obj2 de all_colors faire
        si obj['o'] == obj2['o'] alors
            supprimer obj2 dans all_colors
    // Limite à un certain k du nombre de couleurs détectées associées à
       la même couleur de référence
    compteur(obj['c']) ++
    si compteur(obj['c']) > k alors
        pour chaque objet obj2 de all_colors faire
            si obj['c'] == obj2['c'] alors
                supprimer obj2 dans all_colors
```

Algorithme 3 : Classification des couleurs

Cet algorithme a subi de nombreuses modifications lors du développement. Son implémentation diffère dans le cas d'un cube impair, où la couleur des faces de cubic situées au centre de chaque face est connue à l'avance et n'est donc pas classifiée. Le k utilisé dépend de la taille du cube et correspond au nombre de cubic de chaque face (moins le cubic central dans le cas de Rubik's Cube possédant un tel centre).

À la fin de cette étape, l'utilisateur se voit présenté le résumé de son cube avec les couleurs correspondantes à chaque face de cubic.

3.2.5 Résultats obtenus et modifications manuelles

Les résultats obtenus par la stratégie précédemment présentée sont prometteurs. Pour ce projet, une fois la stratégie implémentée, les algorithmes ont été testés sur des Rubik's Cube de taille différente (allant de $2 \times 2 \times 2$ à $6 \times 6 \times 6$) ainsi que sur plusieurs modèles de Rubik's Cube $3 \times 3 \times 3$, présentant des couleurs différentes. En moyenne, le taux d'erreur est inférieur à 10%, ce qui représente 4 erreurs sur un Rubik's Cube $3 \times 3 \times 3$. Ces erreurs se font principalement à cause de la proximité des couleurs orange et rouge sur certains cubes, où il n'est pas évident de faire la distinction à l'oeil nu.

Suite à ces résultats, une étape supplémentaire est présente. Elle consiste à proposer à l'utilisateur une palette de couleur et lui laisser la possibilité de modifier manuellement les couleurs de chaque face de cubic, afin de corriger les erreurs de classification en suivant son Rubik's Cube physique.

3.3 Présentation de la résolution à l'utilisateur

À cette étape de l'application, le modèle du cube a été calculé et envoyé aux algorithmes de résolution suivant la méthode décrite en 4.5. Ainsi la liste de mouvements à faire pour résoudre le cube a été calculée. L'enjeu de cette partie est donc de trouver une manière adéquate de présenter les mouvements de résolution.

Pour que l'utilisateur suive la résolution calculée par l'application, il convient de faire une représentation du cube et des mouvements simple. C'est pourquoi un modèle standard de cube en trois dimensions a été choisi. Ce modèle de cube doit s'adapter à celui de l'utilisateur (couleurs et dimensions) pour qu'il ne soit pas perturbé. En ce qui concerne l'illustration des mouvements de résolution à faire pour résoudre le cube, la solution retenue est de montrer l'exemple grâce à une animation dudit mouvement.

3.3.1 Représentation tridimensionnelle du cube

Un des critères qui a poussé le choix de `P5.js` est la possibilité qu'il offre d'interagir avec des objets 3D. Cela s'explique notamment car `P5.js` utilise WebGL. La représentation tridimensionnelle du cube en a donc été facilitée.

Structure interne de Rubik's Cube

Le modèle de cube qui apparaît à l'écran est en fait une structure composée de cubics. Un cubic est une boîte (objet `box`) contenant six faces dans un ordre particulier. Une face a une couleur et une position dans l'espace. Le diagramme de classe en figure 3.1 reprend cette explication.

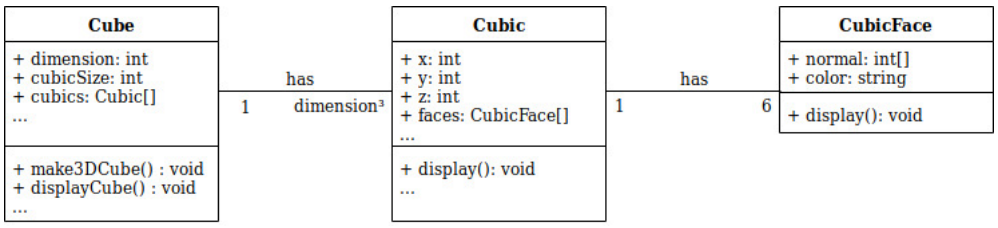

FIGURE 3.1 – Diagramme de classe simplifié de l'architecture du cube

Ainsi lors de la création d'un `Cube`, on va créer $N \times N \times N$ cubics, positionnés à intervalle régulier dans l'espace. Enfin, pour faciliter les calculs, les coordonnées de chacun des objets constituant le `Cube` sont mises à jour afin que le centre du `Cube` soit aux coordonnées `(0,0,0)`.

Adaptation avec les Rubik's Cube $N \times N \times N$

La solution présentée dans la sous-partie précédente s'adapte parfaitement au cube de l'utilisateur. En plus de créer le nombre exact de cubics constituant le cube, la taille de ces derniers est également paramétrée en fonction de la dimension du cube : $\frac{150}{N}$. La valeur 150 représente en fait la taille maximum d'un cube. Il s'agit d'une valeur choisie arbitrairement pour que quelque soit la dimension du cube, le modèle reste aux mêmes dimensions.

Coloration du cube

À ce stade, l'application est capable de montrer à l'utilisateur un cube au format $N \times N \times N$ résolu. Le cube est dans l'état résolu car les faces de ses cubics sont initialisées dans le même ordre, ce qui fait que chacun des cubics est orienté de la même façon. Le seul bémol à cette solution est que l'utilisateur n'a pas totalement une représentation de *son cube*. Pour que ce soit vraiment le cas, il faut que le modèle de cube présenté soit dans le même état, c'est à dire que la coloration des cubics soit identique.

Pour réaliser cela, la solution trouvée est de partir de l'état neutre du cube et de lui appliquer l'inverse des mouvements de résolution disponibles. En quelque sorte le cube est mélangé pour arriver à l'état exact de celui de l'utilisateur. Ceci est possible car à ce stade de l'application, les mouvements que l'utilisateur doit réaliser pour résoudre son cube sont connus. Il faut donc parcourir à l'envers la liste de mouvements de résolution et pour chacun, appliquer l'inverse.

Par exemple si les mouvements pour résoudre le cube sont $FUL'D2$, le cube neutre va se mélanger en réalisant les mouvements $D2LU'F'$. Pour inverser un mouvement, il y a trois cas de figure différents :

— Le mouvement est à effectuer dans le sens horaire : il faut alors l'effectuer dans le sens anti-horaire.
— La réciproque : le mouvement est à effectuer dans le sens anti-horaire : il faut alors l'effectuer dans le horaire.
— Le mouvement est de la forme $M2$ avec M la lettre désignant un mouvement : le mouvement n'a pas à être inversé.

3.3.2 Animation des mouvements de résolution

Les mouvements de résolution changent l'état du cube : certains cubics vont changer de place tandis que d'autres ne bougent pas. L'application doit pouvoir montrer à l'utilisateur comment réaliser ce mouvement et prendre en compte ce changement d'état au fur et à mesure que les mouvements s'enchaînent.

Représentation des mouvements

Une classe `Move` a été introduite et désormais un `Cube` connaît six mouvements : `F`, `B`, `R`, `L`, `U`, et `D`. Un mouvement a notamment des coordonnées, un angle et une direction d'angle et un nombre d'itération comme on peut le voir sur la figure 3.2.

Les coordonnées (x, y, z) permettent de cibler la zone d'action du mouvement. Cette donnée est particulièrement utile pour la sélection des cubics concernés par le mouvement. À noter qu'une valeur aberrante pour ces coordonnées permet de montrer que le mouvement n'intervient pas sur cet axe.

L'attribut angle de la classe `Move` est utile lors de l'animation. L'animation consiste à appliquer des rotations sur le bon axe avec une valeur d'angle de plus en plus grande jusqu'à ce qu'elle atteigne $\frac{\pi}{2}$ (ou π si le mouvement est de type $M2$ avec M le nom du mouvement).

La direction de l'angle est utile pour prendre en compte les différences de sens dans la rotation des mouvements. Une valeur positive est associée à une rotation dans le sens horaire alors qu'une valeur négative est associée à une rotation dans le sens anti-horaire.

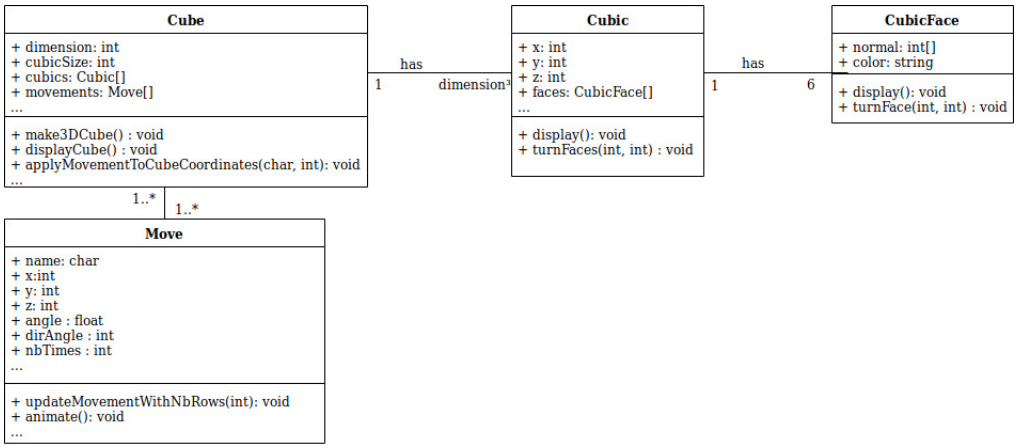

FIGURE 3.2 – Diagramme de classe simplifié de l'architecture du cube avec les mouvements

Enfin, connaître le nombre d'itérations d'un mouvement permet de différencier les mouvements M des mouvements $M2$ avec M un nom de mouvement. Cela est utile notamment pour savoir quand l'angle doit être égal à π ou non.

Sélection des cubics concernés par le mouvement

Chacun des mouvements n'implique qu'une seule face du cube. La sélection des cubics concernés se fait donc en fonction du mouvement à faire. Ainsi pour chaque mouvement, un cubic est concerné si la *coordonnée correspondante à l'axe de rotation du mouvement* est égale à celle dudit mouvement. Par exemple, pour le mouvement `F` qui consiste à faire tourner la face frontale du cube dans le sens horaire, c'est la première rangée de cubics qui va être atteinte. La coordonnée z des cubics est alors comparée avec celle du mouvement. Si les deux correspondent, alors ce cubic est atteint par le mouvement.

Une fois les cubics concernés sélectionnés, ils pivotent d'un angle compris entre 0 (l'animation n'a pas commencé) et $\frac{\pi}{2}$ ou π selon le mouvement. Une fois cet angle atteint et l'animation finie, les coordonnées de ces cubics sont mises à jour.

Calcul des nouvelles coordonnées des cubics après application d'un mouvement

Comme il a été précisé plus tôt chacun des mouvements n'intervient que sur un seul des trois axes du cube :

— `F` et `B` tournent les cubics sur l'axe Z.
— `R` et `L` tournent les cubics sur l'axe Y.
— `U` et `D` tournent les cubics sur l'axe X.

Selon l'axe autour duquel la rotation est effectuée, les nouvelles coordonnées des cubics se calculent de manière différente. Pour les équations suivantes x', y' et z' représentent les coordonnées d'un cubic après application de la rotation. De fait, x, y et z représentent les coordonnées d'un cubic avant la rotation. Enfin, α représente l'angle de rotation : l'angle avec lequel on pivote autour de l'axe désiré.

— Équation de rotation autour de l'axe X :
  — $x' = x$
  — $y' = (y \times cos(\alpha)) - (z \times sin(\alpha))$
  — $z' = (y \times sin(\alpha)) + (z \times cos(\alpha))$
— Équation de rotation autour de l'axe Y :
  — $x' = (z \times sin(\alpha) + (x \times cos(\alpha))$
  — $y' = y$
  — $z' = (z \times cos(\alpha) - (x \times sin(\alpha))$
— Équation de rotation autour de l'axe Z :
  — $x' = (x \times cos(\alpha) - (y \times sin(\alpha))$
  — $y' = (x \times sin(\alpha) + (y \times cos(\alpha))$
  — $z' = z$

Concernant les équations précédentes, il est à noter que l'angle α reste le même pour les mouvements dans le même sens, c'est à dire que les mouvements de type `F` ou `L` auront un angle égal à $\frac{\pi}{2}$. En revanche leurs mouvements inverses, à savoir `F'` ou `L'` auront un angle égal à $-\frac{\pi}{2}$. Enfin, si nous avons un mouvement de la forme `F2` ou `L2`, l'angle sera égal à π.

Adaptation avec les cubes N×N×N

Avoir un cube de dimension $N \times N \times N$ introduit de nouveaux types de mouvements : les mouvements peuvent désormais être préfixés par un chiffre représentant le nombre de rangées concernées par le mouvement. La solution développée prend ces mouvements en compte en mettant à jour les coordonnées des mouvements. Ainsi pour le mouvement `2F` par exemple, l'utilisateur va devoir tourner les deux premières faces frontales du cube dans le sens horaire. En arrivant à ce mouvement, l'application va d'abord créer un mouvement de type `F` avant de modifier la coordonnée z du mouvement pour faire en sorte de prendre en compte deux rangées de cubics. De ce fait, la manière de sélectionner les cubics demeure inchangée.

Partie 4

Second objectif : analyse et implémentation des méthodes de résolution

Dans le but de permettre à l'utilisateur de résoudre son Rubik's Cube, il faut un système calculant les mouvements permettant de le terminer.

Cette partie du rapport s'intéresse d'abord aux différents algorithmes de résolution avant de présenter la mise en place de notre application et la communication avec l'interface.

4.1 Positionnement par rapport à l'état de l'art

4.1.1 Analyse des méthodes de résolution existantes

Méthodes de résolution humaines

Il existe de nombreuses méthodes réalisables par un homme, certaines sont plus optimisées que d'autres, certaines ont pour but de réduire le nombre de coups, d'autres de résoudre un Rubik's Cube le plus rapidement possible (*speedcubing*).

La méthode de résolution dite « couche par couche » est une méthode qui permet de résoudre étape par étape le Rubik's Cube tout en s'aidant de diverses méthodes mnémotechniques.
Les couches sont en fait appelées plus généralement des couronnes, un Rubik's Cube classique en possédant trois.
C'est cette méthode qui a été choisie pour être implémentée de A à Z afin de se familiariser à la manipulation du Rubik's Cube.

Les compétiteurs de résolution de Rubik's Cube utilisent des variantes d'algorithmes afin d'optimiser le temps d'exécution et le nombre de mouvements. Il existe par exemple des moyens d'optimisation de la première couronne, comme la méthode *Jessica Fridrich* par exemple, qui permet de la faire en huit mouvements maximum avec une moyenne de six.[2]

Méthodes de résolution à l'aide de la théorie des groupes

Les concepts mathématiques derrière le Rubik's Cube étant assez forts, il n'est pas étonnant de trouver des méthodes de résolution se basant principalement sur ceux-ci. Parmi ces derniers,

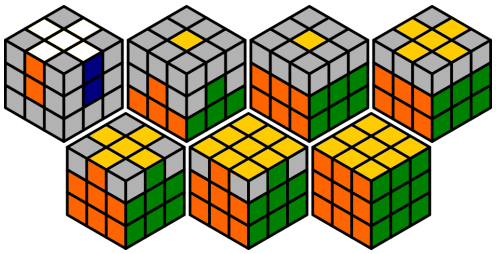

FIGURE 4.1 – Schématisation du couche par couche

celui de la théorie des groupes est régulièrement cité, du fait que l'ensemble des mouvements du Rubik's Cube forme un groupe.

Les méthodes suivantes utilisent la théorie des groupes pour représenter le Rubik's Cube et emploient ses mécanismes afin le résoudre.

Il est à noter que ces méthodes sont purement informatiques. Elles sont conçues pour fonctionner sur un ordinateur et ne sont pas applicables à une résolution manuelle.

Le but est de trouver une suite de mouvements permettant de résoudre un Rubik's Cube donné en un minimum de coups.

Pour rappel, un groupe (au sens mathématique du terme) est un ensemble d'éléments muni d'une loi de composition interne associative admettant un élément neutre et, pour chaque élément de l'ensemble, un élément symétrique.

Voici les caractéristiques d'un groupe appliqué au Rubik's Cube :

— Le groupe G représente l'ensemble de positions du Rubik's Cube.
— Il est engendré par les mouvements de base U,R,F,D,L et B. De ce fait, on note $G = <U,R,F,D,L,B>$
— Soit M l'ensemble des mouvements du cube et soient m,n et $o \in M$, et soit « suivi de » une opération traduisant le fait qu'un mouvement est suivi d'un autre.
— m « suivi de » $n \in M$: un mouvement suivi d'une autre est toujours un mouvement. M est donc stable par l'opération « suivi de ».
— m « suivi de » (n « suivi de » o) $\Leftrightarrow$ (m « suivi de » n) « suivi de » o : l'opération « suivi de » est associative.
— L'élément neutre est la permutation consistant à ne rien faire.
— On définit des sous-groupes du groupe G, notés G_1, G_2, ..., G_n, avec $G \supset G_1 \supset G_2 \supset ... \supset G_n = \{ I \}$ (à savoir, le cube résolu).
— Chaque sous-groupe restreint le groupe précédent au niveau des mouvements. Ainsi un groupe plus petit contiendra moins de positions et certains mouvements ne seront plus possibles.
— Le Rubik's Cube se résout en navigant de sous-groupe en sous-groupe en utilisant seulement les mouvements du sous-groupe courant jusqu'à pouvoir passer au suivant. Une fois dans un sous-groupe, l'algorithme ne revient jamais à un groupe antérieur.

Ces algorithmes sont basés sur les principes suivants :
Soit un ensemble de n groupes, noté $G_0, G_1, ..., G_n$. Le Rubik's Cube est analysé pour déterminer à quel sous-groupe il appartient. Supposons qu'il appartienne à G_0. Grâce à une suite de quarts de tours (les mouvements autorisés de G_0), il atteint une position de G_1. Dès lors, certains mouvements ne seront plus possibles (les seuls autorisés seront ceux de G_1). L'opération recommence pour

passer de G_1 à G_2, puis de G_i à G_{i+1}, et ainsi de suite jusqu'à G_n, le groupe contenant le Rubik's Cube résolu.

Morwen Thistlethwaite est le premier à avoir trouvé une solution efficace au Rubik's Cube, en utilisant cinq groupes. Il a par ailleurs permis de majorer le nombre maximum de coups nécessaires (à savoir 52) pour résoudre un cube depuis n'importe quelle position. [3]

Son algorithme fut amélioré par Herbert Kociemba en 1992 en réduisant le nombre de groupes de cinq à trois. Son logiciel, Cube Explorer, implémente cet algorithme. [4]

En 1997, Michael Reid améliora l'algorithme de Kociemba en utilisant la réduction de l'espace de recherche par utilisation de classes d'équivalence et la symétrie.

Méthodes de résolution à l'aide du Filtrage par motifs

En 1997, Richard Korf réalise un algorithme permettant de résoudre des instances aléatoires d'un Rubik's Cube avec un faible nombre de coups. Il se base sur un parcours de graphe appelé IDA* (*Iterative Deepening A**, ou Profondeur Itérative A*) qu'il a décrit la première fois en 1985. Ce parcours, mis en lien avec le filtrage par motifs, permet des résolutions qui obtiennent de très bons résultats mais avec un temps de résolution moyen plus grand que les autres algorithmes.[5]
Le filtrage par motifs, ou *pattern matching* en anglais, est un type d'algorithme se basant sur la reconnaissance de motifs.
Un motif est un état du cube. Dans notre cas, il s'agit d'une position spécifique du cube.[6]

Comparatif

Ici sont résumés les principaux algorithmes vus précédemment.
Les données présentes dans ce tableau sont issues de recherches, d'expérimentations et d'estimations.

Algorithme	**Nombre de coups en moyenne**	**Temps de calcul**	**Place en mémoire**
Kociemba	25	Rapide	Importante (stockage des groupes)
IDA*	20	Peut devenir très long au-delà d'une profondeur de 10	Importante (stockage des motifs)
Méthode couche par couche	250	Très rapide	Faible

Ces trois approches permettront de comparer l'efficacité des algorithmes selon les cas et d'offrir à l'utilisateur un choix de résolution assez large.

4.1.2 Définition du besoin

L'analyse de l'existant a permis de définir quels sont les éléments importants pour l'application. Pour rappel, cette application doit être capable de communiquer avec la partie *Acquisition et modélisation* afin de prendre un Rubik's Cube de taille quelconque et de renvoyer la suite de coups permettant sa résolution.

4.1.3 Choix des algorithmes

À la vue des différentes méthodes étudiées, il a été décidé d'implémenter la méthode couche par couche afin de comprendre le fonctionnement d'une méthode de résolution et de proposer une version « didactique » de la résolution du Rubik's Cube.

Dans le but de de fournir une application qui optimise le nombre de coups, l'algorithme de Kociemba a également été sélectionné, pour ses bons résultats et sa rapidité d'exécution.

4.1.4 Choix de l'outil

L'analyse de l'existant a permis d'essayer plusieurs solveurs, de les comparer et de voir leur implémentation.

En se basant sur les solveurs existant et sur ceux à incorporer à l'application, le langage Python a semblé être le plus adapté pour cette tâche.

4.1.5 Présentation de la solution

L'apport qu'offrirait cette application par rapport aux applications existantes serait sa modularité. En effet, elle est portable d'une application de modélisation à l'autre et peut ainsi facilement s'intégrer dans d'autres projets.

De plus, elle permettrait de rassembler différentes méthodes de résolution, allant de la plus optimisée à la plus didactique. En outre, elle offrirait la possibilité de résoudre des cubes de toutes tailles.

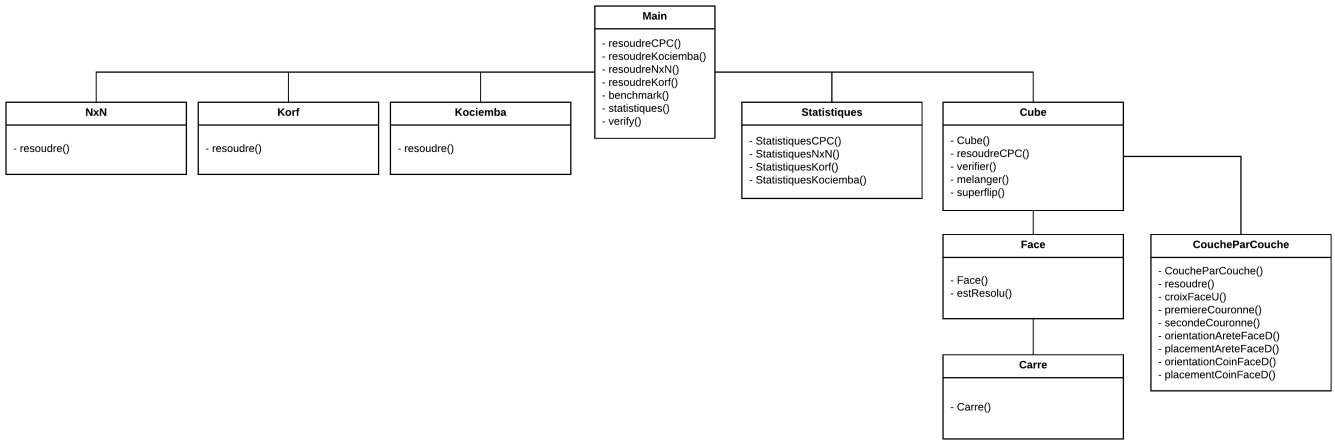

FIGURE 4.2 – Schématisation de l'architecture utilisée

Ce rapport va également présenter une vulgarisation des algorithmes étudiés et implémentés afin de permettre au lecteur de comprendre les mécanismes sous-jacents au Rubik's Cube et peut-être lui donner l'envie d'apprendre à le résoudre.

4.2 Implémentation du couche par couche

4.2.1 Modélisation

L'une des premières étapes a été de choisir une modélisation pour le cube. Cette étape s'est faite conjointement avec la partie *Acquisition et modélisation* afin de faciliter la communication et le voyage du cube au travers de l'application.

Pour décider de cela, une étude des formats de représentation utilisés par les autres algorithmes a été effectuée. En effet, le but étant de regrouper plusieurs solveurs en une seule application, il fallait que le cube soit facilement transférable d'un solveur à l'autre.

Au final, la modélisation suivante a été conservée :

```
             |************|
             |*U1**U2**U3*|
             |************|
             |*U4**U5**U6*|
             |************|
             |*U7**U8**U9*|
             |************|
************|************|************|************
*L1**L2**L3*|*F1**F2**F3*|*R1**R2**R3*|*B1**B2**B3*
************|************|************|************
*L4**L5**L6*|*F4**F5**F6*|*R4**R5**R6*|*B4**B5**B6*
************|************|************|************
*L7**L8**L9*|*F7**F8**F9*|*R7**R8**R9*|*B7**B8**B9*
************|************|************|************
             |************|
             |*D1**D2**D3*|
             |************|
             |*D4**D5**D6*|
             |************|
             |*D7**D8**D9*|
             |************|
```

La chaîne de caractère pour représenter un cube se compose de cinquante-quatre caractères parmi les lettres U, R, F, D, L et B. Ces lettres correspondent respectivement aux couleur de la face du haut (up), de droite (right), avant (face), du bas (down), et gauche (left) et arrière (back). La chaîne de caractères possède l'ordre suivant :

```
U1, U2, U3, U4, U5, U6, U7, U8, U9, R1, R2, R3, R4, R5, R6, R7, R8, R9, F1, F2, F3, F4,
    F5, F6, F7, F8, F9, D1, D2, D3, D4, D5, D6, D7, D8, D9, L1, L2, L3, L4, L5, L6, L7,
    L8, L9, B1, B2, B3, B4, B5, B6, B7, B8, B9.
```

Par exemple, la définition d'un cube résolu est :

```
UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB
```

L'objet `Cube` se compose des spécificités suivantes :

Chaîne de caractères : Le cube est représenté par une chaîne de caractères contenant cinquante-quatre caractères parmi U, R, F, B, L, et D.

Faces : Un cube est composé de six faces.

- **Chaîne de caractère** : La face est représentée par une chaîne de caractère contenant neuf caractères parmi U, R, F, B, L et D.

Couleur : Une face possède un attribut *couleur* représentant la couleur de son centre.

Cubics : Une face est composée de neuf cubics. Chaque cubic est identifié par son numéro, correspondant à son placement sur la face, et sa couleur, parmi U, R, F, B, L et D.

Mouvements : Les mouvements modifient la chaîne de caractères représentant le cube. Les six mouvements de base ainsi que leurs versions anti-horaires ont été implémentés. Chaque mouvement correspond à une permutation des couleurs des cubics concernés.

4.2.2 Implémentation des différentes étapes de résolution

La méthode couche par couche est la méthode que chaque membre du groupe a appris pour ce projet. C'est une méthode simple mais efficace, qui est associée à des moyens mnémotechniques. La méthode couche par couche permet de résoudre le cube couronne par couronne.

Première couronne : Correspond à la première rangée résolue. « Faire la première couronne » revient donc à réaliser la première face et à bien positionner les arêtes et les coins.

Deuxième couronne : Correspond à la deuxième rangée résolue. « Faire la deuxième couronne » revient donc à positionner les arêtes entre les cubics centraux des faces F, R, L et B.

Troisième couronne : Correspond à la troisième rangée résolue. « Faire la troisième couronne » revient donc à réaliser la dernière face et à bien positionner les arêtes et les coins.

Ici la résolution se fait en partant de la face blanche puis en remontant jusqu'à la face bleue sur la figure ci-dessous.

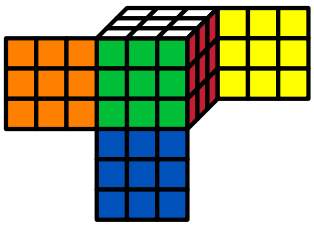

FIGURE 4.3 – Cube résolu mis à plat

L'algorithme de couche par couche s'applique comme ceci :

— La croix blanche est la première étape à réaliser. Pour ce faire, le centre et les quatre autres côtés des arêtes doivent être placés en face des centres de même couleur.
— Ensuite il faut finir la face blanche en plaçant les quatre coins blancs tout en respectant les couleurs des deux autres faces.
— Grâce à ces manipulations, la première couronne est faite.
— Pour la seconde couronne, le premier mouvement complexe est à effectuer : *le belge*, qui permet de placer les arêtes de la seconde couronne.
— La troisième et dernière couronne se fait en même temps que la face inférieure. Pour cela, il faut commencer en réalisant la croix.
— Une fois la croix en place, c'est au tour des arêtes. Pour les positionner de la bonne façon, il faut utiliser *la chaise simple*.

— Maintenant que les arêtes sont bien positionnées, il faut bien placer les coins sans forcément avoir une orientation correcte. Ici c'est la méthode *des voisins* qu'il convient d'utiliser.
— Et enfin, pour tourner correctement les coins il faut utiliser la méthode de *la double chaise*.
— Ainsi le Rubik's Cube est résolu.

L'implémentation de l'algorithme du couche par couche s'est résumé à de nombreuses disjonctions de cas.

Les détails des algorithmes composant le couche par couche se trouvent dans l'annexe C.

Algorithme : MÉTHODE COUCHE PAR COUCHE
Données : cube : une chaîne de caractères correspondant au cube mélangé
Résultat : suiteCoups : la suite de coups à effectuer pour résoudre le cube
début
  si *!EstCorrect(cube)* **alors**
    Retourner "Le cube n'est pas correct" ;
  sinon
    suiteCoups = "" ;
    suiteCoups+=croixFaceU(*cube*) ;
    suiteCoups+=premiereCouronne(*cube*) ;
    suiteCoups+=deuxiemeCouronne(*cube*) ;
    suiteCoups+=orientationAretesFaceD(*cube*) ;
    suiteCoups+=placementAretesFaceD(*cube*) ;
    suiteCoups+=placementCoinFaceD(*cube*) ;
    suiteCoups+=orientationCoinsFaceD(*cube*) ;
    Retourner suiteCoups ;

Algorithme 4 : Méthode couche par couche

Cet algorithme résout le cube en cent quarante-deux coups en moyenne, pour un temps moyen de recherche de solution de 0.002 secondes.

4.3 Autres algorithmes de résolution

4.3.1 Kociemba

Comme vu précédemment, la méthode de H. Kociemba utilise la théorie des groupes. Son algorithme, le *Two-Phase-Algorithm* (Algorithme en deux phases), permet de résoudre le cube de façon relativement optimisée, en obtenant un score proche du nombre de Dieu. Pour ce faire, il commence par analyser à quel sous-groupe de $G = <U,R,F,B,L,D>$ appartient le Rubik's Cube donné en entrée. Ensuite, à l'aide des mouvements autorisés par ce sous-groupe et d'heuristiques adaptées, il va passer au sous-groupe suivant. Il effectue ces opérations jusqu'à arriver au groupe contenant uniquement le cube résolu.

Cet algorithme effectue plusieurs passes afin de trouver la meilleure suite de coups tout en restant dans un temps d'attente raisonnable pour l'utilisateur.

Les sous-groupes utilisés par l'algorithme de Kociemba sont les suivants :

— $G_0 =< L, R, F, B, U, D >$
Ce sous-groupe autorise tous les mouvements et contient toutes les positions possibles du Rubik's Cube.

— $G_2 =< L2, R2, F2, B2, U, D >$
Ce sous-groupe contient toutes les positions qui peuvent être atteintes (depuis le cube résolu) par des quarts de tour à gauche et à droite mais uniquement des doubles tours sur les faces avant, arrière, haut et bas.
— $G_4 = \{I\}$
Ce groupe contient uniquement le cube résolu.

Algorithme : KOCIEMBA
Données : cube : une chaîne de caractères correspondant au cube mélangé
Résultat : suiteCoups : la suite de coups à effectuer pour résoudre le cube
début
  si *!EstCorrect(cube)* **alors**
    Retourner "Le cube n'est pas correct" ;
  sinon
    sousGroupeCourant = verifierSousGroupe(*cube*) ;
    tant que *sousGroupeCourant* $\neq G_4$ **faire**
      sousGroupeCourant = passerSousGroupeSuivant(*cube*,*suiteCoups*) ;
    Retourner suiteCoups ;

Algorithme 5 : Algorithme de Kociemba

Cet algorithme résout le cube en vingt coups en moyenne, pour un temps moyen de recherche de solution de 0.0486 secondes.

Le code source ainsi que de plus amples informations peuvent être trouvés à l'adresse suivante : `http://kociemba.org/`

4.3.2 Profondeur Itérative A*

*IDA** est un algorithme de parcours en profondeur de graphes utilisant des heuristiques permettant de rechercher le plus court chemin entre deux états. Cet algorithme est ici lié à la recherche de motifs dans le but de passer de motif en motif jusqu'à l'état résolu.

Dans cette configuration, les parcours sont faits en vue d'atteindre les objectifs suivants :

1. Chercher le bon placement des coins indépendamment du reste du cube.
2. Chercher le bon placement de six arêtes indépendamment des coins et des autres arêtes.
3. Chercher le bon placement des six autres arêtes.
4. Obtenir un cube résolu.

Cela permet d'obtenir un résultat très proche du nombre de coups minimum nécessaire à la résolution d'un Rubik's Cube , mais pour un temps bien plus important que les méthodes précédentes. Une fois implémenté et testé, les instances prenaient entre une et dix minutes à se résoudre, mais offraient de bien meilleurs résultats, avec dix à vingt coups en moyenne.

4.3.3 NxNxN

Il existe des méthodes permettant de résoudre un Rubik's Cube quelque soit sa taille. L'idée générale choisie repose sur la réduction d'un Rubik's Cube de dimension n à un cube de dimension

inférieure et compatible, c'est à dire « simuler » un Rubik's Cube de dimension n avec un Rubik's Cube de dimension strictement supérieur à n. Par exemple, on peut réduire un Rubik's Cube 4x4x4 à un cube 2x2x2 (en bougeant toujours les arêtes du 4x4x4 par deux, le Rubik's Cube pourra être utilisé comme un 2x2x2, et il pourra se résoudre à l'aide des solution adaptées).

Le but est, pour un Rubik's Cube de dimension quelconque, de le réduire à un 3x3x3 pour appliquer l'algorithme de Kociemba. Cependant, les réductions ont des limites : on ne pourra pas simuler un Rubik's Cube 2x2x2 avec un Rubik's Cube 3x3x3 par exemple. C'est pour cela qu'il faut faire une disjonction de cas selon si le Rubik's Cube est de dimension paire ou impaire.

Pour passer d'une dimension à une dimension inférieure, l'algorithme utilise, pour les Rubik's Cube de 4x4x4 à 7x7x7, des tables de correspondances. Cela revient à faire de la recherche de motifs pour optimiser la réduction.

Le principe de fonctionnement général de l'algorithme est le suivant : Tout d'abord, il faut analyser la dimension du Rubik's Cube (paire ou impaire). Ensuite, l'algorithme va former les centres du Rubik's Cube à l'aide d'un algorithme générique. Par la suite, il va vouloir reformer les arêtes. Pour ce faire, et selon la dimension du cube, le Rubik's Cube sera réduit à un 7x7x7 puis à un 5x5x5 (cas impair) ou à un 6x6x6 à un 4x4x4 (pair) avant d'être réduit à un 3x3x3. Une fois réduit à un 3x3x3, l'algorithme applique l'algorithme de Kociemba.

Le code source ainsi que de plus amples informations peuvent être trouvés à l'adresse suivante : `https://github.com/dwalton76/rubiks-cube-NxNxN-solver`

```
Algorithme : SolveurNxNxN
Données : cube : une chaîne de caractères correspondant au Rubik's Cube mélangé
Résultat : suiteCoups : la suite de coups à effectuer pour résoudre le Rubik's Cube
début
    si !EstCorrect(cube, dimension) alors
        Retourner "Le cube n'est pas correct" ;
    sinon
        dimension = analyseDimension(cube) ;
        suiteCoups = formerCentre(cube, dimension)
        si dimension > 3 alors
            si (dimension%2 == 0) alors
                suiteCoups = reduction6x6x6(cube) // Réduit les arêtes du Rubik's
                    Cube à une configuration de 6x6x6
                suiteCoups = reduction4x4x4(cube) // Réduit les arête du Rubik's
                    Cube à une configuration de 4x4x4
            sinon
                suiteCoups = reduction7x7x7(cube) // Réduit les arêtes du Rubik's
                    Cube à une configuration de 7x7x7
                suiteCoups = reduction5x5x5(cube) // Réduit les arête du Rubik's
                    Cube à une configuration de 5x5x5

        suiteCoups = reduction3x3x3(cube) // Réduit le cube à un 3x3x3
        suiteCoups = kociemba(cube) // Applique Kociemba au Rubik's Cube réduit à
            un 3x3x3
        Retourner suiteCoups ;
```

Algorithme 6 : Méthode de résolution d'un cube N x N x N.

4x4x4	5x5x5	6x6x6	7x7x7	8x8x8	9x9x9	10x10x10
50	88	156	218	344	430	601

FIGURE 4.4 – Nombre de coups moyen en fonction de la dimension du Rubik's Cube

4.4 Fonctionnalités connexes

4.4.1 Statistiques, benchmark et mélangeur

Différents algorithmes connexes ont été également implémentés dans cette application. Ils ont pour but de permettre d'obtenir des statistiques ou d'effectuer des tests sur les solveurs.

Mélangeur : Afin de pouvoir vérifier les algorithmes de résolution de l'application, il a fallu un ensemble de Rubik's Cube de test. La mise en place d'un mélangeur a donc été décidée, afin de pouvoir générer des cubes valides mélangés.

Benchmark : Pour pouvoir obtenir des statistiques sur les algorithmes de résolution, il a fallu tester ceux-ci sur un grand nombre de Rubik's Cube valides. Pour ce faire, un générateur de benchmark a été mis en place. Il permet de créer n instances valides et mélangés de cube. À noter que ce benchmark inclus également la position du *superflip*, qui est la solution nécessitant le plus grand nombre de coups pour résoudre un cube (20 coups).

Statistiques : Pour finir, afin de chiffrer et comparer l'efficacité des algorithmes, des statistiques sur le temps moyen de calcul d'une solution et le nombre moyen de coups à effectuer pour chacun des algorithmes ont été effectuées en se basant sur le benchmark.

4.4.2 Validité

Il est utile d'avoir une méthode de vérification de la validité d'un Rubik's Cube pour être certain de résoudre un cube solvable. Un cube invalide pourrait faire boucler certains algorithmes ou renvoyer des résultats incorrects.

Afin de vérifier qu'un Rubik's Cube est valide différents critères sont à vérifier :

— Il faut qu'il y ait neuf fois six couleurs différentes, cela se vérifie en calculant les nombres maximum et minimum de couleurs égales ;
— Que chaque coin soit unique puis dans une orientation valide ;
— Que chaque angle soit unique puis dans une orientation valide ;
— Si tout cela est correct, il ne reste plus qu'à vérifier la parité. Il est nécessaire que la somme des parités des coins et des angles soit paire.

4.5 Mise en place de l'API

Une fois les algorithmes étant prêts à être développés ou étant déjà implémentés, il a fallu les rendre disponibles et permettre l'intégration avec la partie Visualisation.
En effet, les deux parties sont dans des langages différents, `Python` pour les solveurs et principalement `JavaScript` pour l'application.

Algorithme : Vérification de la validité d'un Rubik's Cube

Données : cube : une chaîne de caractères correspondant à un cube
Résultat : Vrai si le cube est valide, faux non
début
    #Vérification du bon nombre de couleurs sur le cube $(6 * 9)$
    si *MaxNombreCouleursEgales(cube) !=6 || MinNombreCouleursEgales(cube) !=6* **alors**
        Retourner faux ;
    #Vérification des coins, il en existe douze différents
    si *!uniciteDesCoins(cube)* **alors**
        Retourner faux ;
    #Vérification que chaque coin est dans le bon sens
    si *!bonneOrientationsCoins(cube)* **alors**
        Retourner faux ;
    #Vérification des arêtes, il en existe huit différents
    si *!uniciteDesAretes(cube)* **alors**
        Retourner faux ;
    #Vérification que chaque arête est dans le bon sens
    si *!bonneOrientationsAretes(cube)* **alors**
        Retourner faux ;
    #Vérification de la parité
    si *!(pariteCoins(cube) + pariteAretes(cube) % 2 == 0)* **alors**
        Retourner faux ;
    Retourner vrai ;

Algorithme 7 : Validité d'un Rubik's Cube

Pour cela, une API en `Python` regroupant tous les algorithmes a été développée. Elle est accessible via des commandes sur un terminal. (Voir annexe D).

Une interface en `JavaScript` a alors été faite afin que la partie Visualisation puisse utiliser sans problèmes les solveurs.

L'interface utilise un script `PHP` qui exécute une commande `Python` via l'environnement courant.

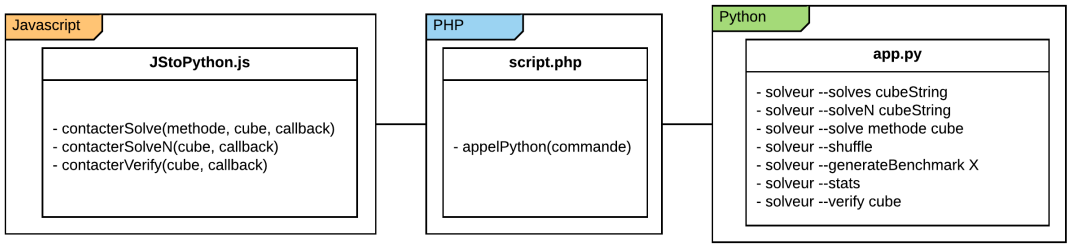

FIGURE 4.5 – Communication entre les langages

Partie 5

Conclusion

5.1 Avancement du projet

Ce rapport aborde ce qu'est un Rubik's Cube et présente quelles sont les notations en rigueur pour cet objet avant d'étudier les thématiques de recherche qui y sont adossées.

Après avoir présenté les aspects du Rubik's Cube nécessaires au projet, il a fallu se concentrer sur l'acquisition du cube et sa représentation auprès de l'utilisateur. L'acquisition choisie est composée de plusieurs étapes : calibration, détection et classification des couleurs du cube. La présentation du cube ainsi que celle des mouvements de résolution se font part une représentation tridimensionnelle dont le modèle s'anime et se met à jour selon les mouvements de résolution du cube.

Concernant la partie algorithmique, après avoir présenté un bref état de l'art, le rapport présente les méthodes choisies pour ce projet. Le couche par couche a été implémenté tandis que des algorithmes libres de droits pour les méthodes de *Kociemba*, *Iterative Deepening A* et de $N \times N \times N$ ont été récupérés et ajoutés à l'API.

Ce TER a permis la réalisation d'une application web permettant aux utilisateurs de résoudre leur Rubik's Cube. Cette application propose différentes méthodes de résolution et prend en compte différentes tailles de Rubik's Cube. Par ailleurs, de part la séparation dans le développement des parties *interface* et *algorithmique*, l'application peut être facilement enrichie de nouvelles méthodes de résolution. Enfin, l'accent a été mis sur la facilitation pour l'utilisateur de représenter son cube. Pour cela une méthode de détection des couleurs du cube couplée à une classification ont été utilisées. Enfin, la reconnaissance vocale autorise l'utilisateur à naviguer plus aisément dans l'application.

À ce jour si le projet était à refaire, nous changerions la façon dont nous nous sommes organisés en tant qu'équipe. Par exemple, nous aurions souhaité avoir des réunions entre nous plus fréquentes pour un suivi plus constant du travail de chacun ou instaurer plus de jalons pour pallier aux éventuels retards.

5.2 Difficultés rencontrés

Ce projet n'échappant à la règle, tout ne s'est pas déroulé comme prévu initialement. Il a fallu être souple, faire des compromis et des concessions à certains moments.

En effet, parmi les problèmes rencontrés, il était prévu d'héberger l'application finale sur un serveur afin de la rendre accessible à tous. La Direction du Système d'Information et du Numérique (DSIN) aurait pu fournir ce serveur. Cependant après de multiples échanges avec ce service pendant plus d'un mois, il a été décidé d'abandonner l'idée et de garder l'application en local, le

serveur proposé n'étant pas adapté à nos besoins.
Les débuts de notre collaboration pour cette partie du projet ont également été quelques peu mouvementés. Chacun étant très indépendant dans son travail, un manque de communication s'est très vite fait ressentir. Afin d'y remédier, plusieurs outils ont été mis en place tels que Trello, qui permet de suivre l'avancement des tâches de chacun, ainsi que Discord, qui donne accès à un serveur de discussions pour tous les membres de l'équipe.

La partie Visualisation a connu quelques problèmes spécifiques. Par exemple, le fait que la reconnaissance des couleurs ne soit pas une science exacte a apporté certaines difficultés. En effet, selon l'environnement, la caméra ou la luminosité, les couleurs peuvent avoir une teinte différente. À cause de cela, certaines couleurs sont mal reconnues et demanderont à l'utilisateur de les modifier manuellement. Un autre problème est qu'une image capturée via la caméra est inversée par rapport à l'utilisateur. Pour remédier à cela, il a fallu modifier l'image afin que l'utilisateur puisse capturer le Rubik's Cube plus simplement.

La partie Résolution a également rencontré ses propres problèmes. Les tâches ont été réparties afin que chacun avance à son rythme. Cependant cela peut parfois être compliqué lorsqu'une tâche à réaliser est dépendante d'autres tâches en cours ou pas encore implémentées. Pour cela, il a été nécessaire de prévoir à l'avance quelles étaient les entrées et sorties attendues ainsi que de faire des tests avec des objets simulant une exécution passée.
L'utilisation de bibliothèques externes a également compliqué les choses. Il a fallu créer des interfaces afin de formater les entrées pour correspondre aux modèles requis, ainsi que pour décoder les valeurs de retours. Cela n'est jamais simple, en particulier si la bibliothèque utilisée est très peu documentée.

5.3 Apports personnels

Un tel projet est riche en apports. La diversité des domaines étudiés et les nombreux aspects du projet étaient très propices à l'apprentissage et l'exploitation de toute cette connaissance.

D'une façon générale, un projet sur deux semestres avec un état de l'art, puis sa réalisation en équipe, est forcément riche en enseignements. Le travail en équipe entraîne des situations dans lesquelles il faut faire preuve de souplesse et d'écoute. Chacun est responsable de la réussite du projet et il faut donc gérer cette pression tout en réalisant les nombreux autres travaux au fil de l'année. De nombreuses API ont été utilisées. Cela nécessite une certaine flexibilité, mais permet en contre partie de comprendre comment sont conçues la plupart des applications de nos jours, ce qui est très intéressant. Cela permet également de se rendre compte des choses à faire et à ne pas faire lorsque l'on réalise quelque chose de semblable.

Au lancement du projet, seul l'un des quatre membres savait résoudre un Rubik's Cube . Il a très vite partagé ses connaissances au reste de l'équipe. Lors de l'état de l'art au premier semestre, l'étude des domaines connexes au Rubik's Cube a permis de se familiariser avec le fonctionnement de la recherche, aidé par la lecture d'un article scientifique.

La réalisation du projet a apporté de nombreuses connaissances techniques, de par l'utilisation du langage `Python` ou `JavaScript`, la communication entre ces langages, ainsi que par l'utilisation

de nombreuses API externes.

Les différents binômes ont appris des choses spécifiques. Celui s'occupant de la visualisation a pu se pencher sur un domaine différent de celui suivi au sein du cursus de leur Master, tandis que celui travaillant à la résolution a pu étudier un mélange entre théorie grâce entre autres aux mathématiques autour du Rubik's Cube et pratique avec l'implémentation de ces méthodes.

5.4 Perspectives

Parmi les idées d'améliorations possibles pour aider l'utilisateur dans sa résolution, il est à noter un suivi en temps réel de cette étape. En effet avec un retour vidéo, il serait possible de surveiller que l'utilisateur ne se trompe pas entre deux mouvements et ainsi éviter qu'il ait à tout recommencer si la résolution échoue. Par ailleurs, une autre idée d'amélioration qui constituerait plus à elle seule un projet entier serait de proposer une version de l'application à l'aide de la réalité mixte comme il est présenté dans [7]. De plus, il est toujours possible d'enrichir l'API avec davantage de méthodes de résolution. Enfin, il pourrait être intéressant de développer un nouvel algorithme de résolution de Rubik's Cube.

Glossaire

API Application Programming Interface. 6, 16, 17

OpenGL (Open Graphical Library) : API multi-langage et multi plateforme pour faire des rendus 3D. 16

WebGL API `JavaScript`, un standard pour faire des rendus 2D et / ou 3D dans un navigateur. Cette API se base sur OpenGL. 22

Bibliographie

[1] T. Rokicki, H. Kociemba, M. Davidson, J. Dethrifge. *The Diameter of the Rubik's Cube Group Is Twenty*, 2010

[2] *Speedcubing* `https://www.worldcubeassociation.org/results/statistics.php`

[3] Morwen Thistlethwaite. *Thistlethwaite's 52-move algorithm.* `https://www.jaapsch.net/puzzles/thistle.htm`, 1981.

[4] H. Kociemba. `http://kociemba.org/cube.htm`, 2018.

[5] Richard E. Korf *Depth-First Iterative-Deepening : An Optimal Admissible Tree Search** `https://cse.sc.edu/~mgv/csce580f09/gradPres/korf_IDAStar_1985.pdf`, 1985.

[6] Richard E. Korf *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases* `https://www.aaai.org/Papers/AAAI/1997/AAAI97-109.pdf`, 1997.

[7] Zhenliang Zhang, Yue Li, Jie Guo, Dongdong Weng,Yuanan Liu, Yongtian Wang.
Vision-tangible interactive display method for mixed and virtual reality : Toward the human-centered editable reality, November 2018. Journal of the Society for Information Display

Annexe A

Comparatif de sites

Tutoriels

De nombreux tutoriels sont disponibles en ligne, dans la majorité des langues existantes, présentant différentes méthodes permettant de résoudre un Rubik's Cube. On peut par exemple citer les sites RubiksSolver ou Le Rubik's Cube qui proposent des méthodes simples et manuelles de résolution d'un Rubik's Cube à l'aide de texte et d'images, similaire à la notice donnée lors de l'achat d'un Rubik's Cube. Il existe aussi de nombreuses vidéos en ligne expliquant en détail comment résoudre simplement un Rubik's Cube.

Ruwix

Ruwix est un site proposant tout un ensemble d'informations sur les Rubik's Cube, semblable à Wikipédia, mais spécialisé dans les Rubik's Cube. La page permettant de résoudre son Rubik's Cube de ce site propose différentes visions du cube, principalement en 2D avec un léger effet de perspective. Ceci rend la résolution du cube assez difficile à suivre à cause de la différence de point de vue entre le cube physique et le cube virtuel. L'acquisition du Rubik's Cube se fait petit à petit, en choisissant une couleur puis le cube auquel l'associer, avec également une fonction « Cube aléatoire ».

La résolution du cube ne se lance pas avec le solveur de ce site, mais avec celui de Rubiks Cube Solver.

Rubiks Cube Solver

Rubiks Cube Solver est un site proposant de saisir et résoudre des Rubik's Cube de taille et forme différentes. Ce site propose différentes visions du cube, dont une en 3D avec la possibilité de déplacer et tourner le cube dans l'espace. L'acquisition du Rubik's Cube se fait petit à petit, en choisissant une couleur puis le cube auquel l'associer, avec également une fonction « Cube aléatoire ».

La résolution du cube tient compte des erreurs de saisie et affiche un message d'erreur en cas de configuration impossible du Rubik's Cube saisi. La résolution est présentée pas à pas, avec une vitesse suffisamment faible pour pouvoir être suivie par une personne normale, et des animations à l'écran permettant de mieux comprendre les mouvements à effectuer.

Grubiks

Grubiks est un site proposant de saisir et résoudre un Rubik's Cube. Ce site propose une vision 3D du cube avec la possibilité de pivoter le cube dans l'espace. L'acquisition du Rubik's Cube se fait petit à petit, en choisissant une couleur puis le cube auquel l'associer.

La résolution du cube tient compte des erreurs de saisie et affiche un message d'erreur en cas de configuration impossible du Rubik's Cube saisi. La résolution est présentée pas à pas, en attendant un clic entre chaque pas. Bien que cela laisse le temps nécessaire à l'utilisateur, ce dernier est également forcé de poser son cube entre chaque étape pour passer à la suivante.

Annexe B

Comparatif d'applications

Tutoriel pour le Cube de Rubik

L'application est similaire à une notice sur appareil mobile et présente le même contenu que la notice donnée lors d'un achat d'un Rubik's Cube. Cependant, elle ne présente qu'une seule méthode de résolution et aucune forme d'interaction avec l'utilisateur.

Cube de Rubik

L'application présente différents modèles 3D de Rubik's Cube allant du $2\times2\times2$ au $8\times8\times8$. Elle ne propose pas de moyen de résoudre le cube. On déplace le cube par mouvement en le touchant ou en touchant son environnement pour le tourner. Ce qui mène parfois au *problème du gros doigt* : manque de précision et le mouvement qui est effectué n'est pas forcément le bon.

Elle propose aux utilisateurs de se chronométrer, ainsi qu'un mode nuit et la possibilité garder en mémoire ses scores.

Rubik Cube

L'application propose à l'utilisateur d'interagir avec le Rubik's Cube. On déplace le cube par mouvement en le touchant ou en touchant son environnement pour le tourner. On retrouve le *problème du gros doigt*. Cette application ne permet pas à l'utilisateur de résoudre le cube.

Elle met également à disposition des tutoriels pour les débutants ainsi que la méthode Fridrich (sous forme de tutoriels filmés, entraînement et liste des mouvements).

Enfin, l'application permet aux utilisateurs de résoudre un cube après un nombre précis de mouvements pour le mélanger.

RubikSolver

L'application permet aux utilisateurs d'interagir avec le modèle de manière assez simple (mais au moins on n'a pas l'effet « gros doigts »). Une seule des faces est visible à la fois et l'utilisateur peut également faire résoudre son Rubik's Cube. Seulement il devra initialiser le modèle virtuel du Rubik's Cube lui même en définissant les couleurs de chaque carré.

Concernant la résolution, l'application applique « bêtement » un algorithme. Si on effectue un seul mouvement depuis l'état neutre du cube, l'algorithme ne va pas « défaire le mouvement » mais

repartir refaire toutes les étapes, ce qui rend sa résolution assez longue. Seule une animation de la résolution est disponible, avec un retour au mouvement précédent qui ne marche pas... Cependant, la vitesse de résolution est modulable.

CubeX

Cette application est la meilleure parmi celles testées. Elle ne permet pas aux utilisateurs d'interagir avec un cube mais permet de passer d'un modèle physique à un modèle virtuel. L'application permet de faire cela par photographies ou par « *click and touch* ». On peut toutefois corriger l'interprétation des couleurs de l'application en changeant nous-même la couleur d'une case. Une mise au point des couleurs du Rubik's Cube est aussi possible.

Une fois le Rubik's Cube « initialisé », l'application calcule différentes façons de le résoudre et les propose à l'utilisateur en précisant le nombre de mouvements nécessaires pour les finir. Une fois que l'on a choisi une des méthodes de résolution proposées, une animation se déclenche montrant les différents mouvements à effectuer. On peut aussi réaliser un mode « pas à pas » ou revenir au mouvement précédent si on est perdu.

Annexe C

Algorithmes du couches par couche

Algorithme : Croix Face U

Données : cube : une chaîne de caractères correspondant au cube mélangé
Résultat : suiteCoups : la suite de coups à effectuer pour former la croix de la face U
début

- **tant que** *!CroixUFaite(cube)* **faire**
  - `// Afin de placer correctement l'arête voulue, une disjonction de cas est suivie par une suite de mouvements spécifiques qui laissent inchangées les arêtes déjà correctement placées.`
  - **tant que** *!croixPremiereAreteU(cube)* **faire**
    - placerUB(*cube*)
    - `// Recherche de l'arête de côté U et B puis on la place correctement.`
  - **tant que** *!croixSecondeAreteU(cube)* **faire**
    - placerUR(*cube*)
    - `// Recherche de l'arête de côté U et R puis on la place correctement`
  - **tant que** *!croixTroisiemeAreteU(cube)* **faire**
    - placerUF(*cube*)
    - `// Recherche de l'arête de côté U et F puis on la place correctement.`
  - **tant que** *!croixQuatriemeAreteeU(cube)* **faire**
    - placerUL(*cube*)
    - `// Recherche de l'arête de côté U et L puis on la place correctement.`
- **Retourner** suiteCoups ;

Algorithme 8 : Croix face U. Nombre de coups : max : 80, moyen : 45

Algorithme : Première Couronne

Données : cube : une chaîne de caractères correspondant au cube avec la croix U de faite
Résultat : suiteCoups : la suite de coups à effectuer pour placer et orienter les coins afin d'effectuer la première couronne

début

- **tant que** *!coinsUPlacés(cube)* **faire**
  - `// Afin de placer correctement le coin voulu, une disjonction de cas est suivie par une suite de mouvements spécifiques qui laissent inchangés les cubes déjà correctement placés.`
  - **tant que** *!facePremierCoinU(cube)* **faire**
    - placerULB(*cube*)
    - `// Recherche du coin U, L et B puis on le place correctement`
  - **tant que** *!croixSecondCoinU(cube)* **faire**
    - placerUBR(*cube*)
    - `// Recherche du coin U, B et R puis on le place correctement`
  - **tant que** *!croixTroisiemeCoinU(cube)* **faire**
    - placerURG(*cube*)
    - `// Recherche du coin U, R et F puis on le place correctement`
  - **tant que** *!croixQuatriemeCoinU(cube)* **faire**
    - placerUFL(*cube*)
    - `// Recherche du coin U, F et L puis on le place correctement`
- **Retourner** suiteCoups ;

Algorithme 9 : Première Couronne. Nombre de coups : max : 95, moyen : 60

Algorithme : Deuxième couronne

Données : cube : une chaîne de caractères correspondant au cube avec la première couronne de faite
Résultat : suiteCoups : la suite de coups à effectuer pour réaliser la deuxième couronne

début

- **tant que** *!deuxiemeCouronne(cube)* **faire**
  - `// Parcourt les arêtes du haut pour en chercher une qui ne contient pas de jaune (donc une arête à placer). On regarde son orientation, on la place et on renvoie la face depuis laquelle effectuer le belge.`
  - `// Si il n'y en a pas en haut, on cherche une arête sur la deuxième couronne mal placée. On renvoie la face depuis laquelle effectuer le belge.`
  - $face$ = positionnementArête(*suiteCoups*) ;
  - `// On regarde l'arête à placer et on détermine si le belge doit se faire en partant dans le sens horaire ou anti-horaire.`
  - *sens* = choixCoté($face$, *suiteCoups*)
  - `// On effectue le belge sur la face` $face$ `dans le sens` *sens*
  - belge($face$,*sens*,*suiteCoups*)
- **Retourner** suiteCoups ;

Algorithme 10 : Deuxième couronne. Nombre de coups : max - 16 + 64, moyen : 4 + 32

Algorithme : Orientation Arêtes Face D

Données : cube : une chaîne de caractères correspondant au cube avec la deuxième couronne de faite

Résultat : suiteCoups : la suite de coups à effectuer pour réaliser la croix sur la face D

début

- **tant que** *!croixFaceD(cube)* **faire**
  - `// Positionne la face D pour pouvoir appliquer l'algorithme du chinois dessus (effectue une ou deux rotation de la face D dans le sens horaire ou anti-horaire)`
  - positionnementCroix(*suiteCoups*)
  - `// Effectue l'algorithme des chinois pour créer la croix jaune`
  - chinois(*suiteCoups*)
- **Retourner** suiteCoups ;

Algorithme 11 : Orientation Arêtes Face D. Nombre de coups : max - 2 + 18, moyen - 1 + 12

Algorithme : Placement Arêtes Face D

Données : cube : une chaîne de caractères correspondant au cube avec les arêtes de la face D bien orientées

Résultat : suiteCoups : la suite de coups à effectuer pour orienter correction les arêtes de la face D

début

- **tant que** *!arreteFaceDPlacees(cube)* **faire**
  - `// Positionne la face D pour pouvoir appliquer la chaise, et renvoie la face depuis laquelle l'appliquer`
  - $face$ = vérifieCentresD(*suiteCoups*) ;
  - `// applique la chaise depuis la face` $face$ `et dans le sens horaire`
  - chaise($face$,AH, *suiteCoups*)
- **Retourner** suiteCoups ;

Algorithme 12 : Placement Arêtes Face D. Nombre de coups : max - 3 + 14, moyen - 2 + 7

Algorithme : Placement coins Face D

Données : cube : une chaîne de caractères correspondant au cube avec les arêtes de la face D bien orientées et placées

Résultat : suiteCoups : la suite de coups à effectuer pour réaliser la croix sur la face D

début

- **tant que** *!coinsFaceDPlacés(cube)* **faire**
  - `// Sélectionne la face depuis laquelle effectuer l'algorithme des voisins`
  - $face$ = verifierPlacementCoinsD(*suiteCoups*) ;
  - `// Effectue l'algorithme des voisins sur la face` $face$
  - voisins($face$,*suiteCoups*)
- **Retourner** suiteCoups ;

Algorithme 13 : Placement Coins Face D. Nombre de coups : max - 24, moyen - 8

Algorithme : Orientation Coins Face D

Données : cube : une chaîne de caractères correspondant au cube avec les coins de la face D bien placés

Résultat : suiteCoups : la suite de coups à effectuer pour passer le cube à l'état résolu

début

  tant que *!cubeResolu(cube)* **faire**

    `// Sélectionne la face depuis laquelle effectuer le double chaise`

    $face$ = veriferOrientationCoinD(*suiteCoups*)

    `// Effectue la double chaise sur la face` $face$

    doubleChaine($face$)

  Retourner suiteCoups ;

Algorithme 14 : Orientation Arêtes Face D. Nombre de coups : max - 56, moyen - 28

Annexe D

Commandes app.py

```
solveur --shuffle
  -> renvoie le string du cube melange et la suite de coups correspondant

solveur --stats
  -> cree le fichier de statistiques sur les methodes de resolution

solveur --generateBenchmark X
    -> génére un benchmark de x cubes

solveur --solves cubeString
  -> Resout le cube 3x3 avec toutes les methodes dispo

solveur --solvesn cubeString
  -> Resout le cube NxN avec une seule methode, de facon opti


solveur --solve methode cube
  -> Resout le cube a l'aide de la methode methode


Exemple :

python app.py --generateBenchmark 10

python app.py --stats

python app.py --solves BLFUUFLURBLRDRULRDUFDBFLUDDLFFUDBFFLRBFDLDDRBUBULBRBRR
```