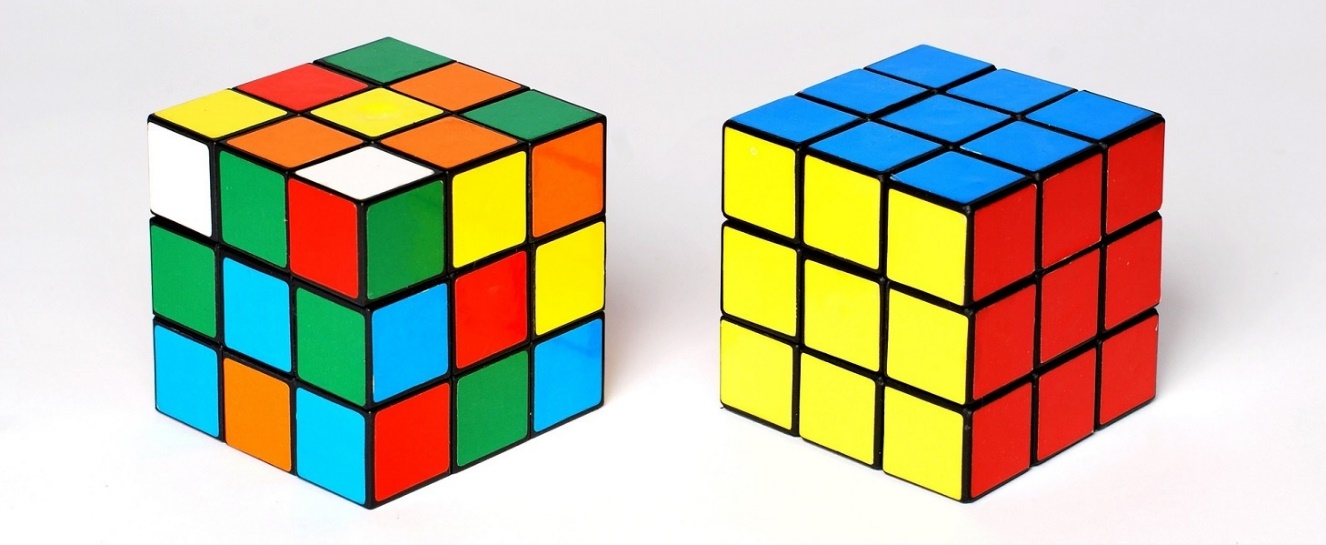
**Document Explicatif**

Cahier des Charges

Le Rubik’s Cube est un cube dont chaque face est divisée en 9 plus petits cubes (car nous ne nous intéresserons dans ce projet qu’au cas du Rubik’s Cube 3X3X3, bien qu’il existe des Rubik’s Cube avec des dimensions différentes) qui peuvent se déplacer indépendamment les uns des autres (voir figure 1). L’objectif du Rubik’s Cube est, après avoir mélangé les six faces, de modifier le cube pour que ces faces soient de couleurs unies, avec chacune une couleur différente (blanc, rouge, bleu, orange, vert, jaune). Ce casse-tête très complexe possède plus de 43 trillions de combinaisons différentes, et il a fallu 1 mois à son inventeur, Ernö Rubik, pour le résoudre. Depuis 1979, plus de 400 millions d’exemplaires du Rubik’s Cube ont été vendu à travers le monde, ce qui en fait un des jouets les plus populaires de tous les temps.

Cela rend donc intéressant l’idée de proposer un programme permettant une résolution guidée, au rythme voulu par l’utilisateur, tant cet objet est à la fois populaire et complexe, voir même fascinant pour un grand nombre.

**

*Figure 1 : Exemple de comparaison entre des Rubik’s Cube 3X3X3 résolus et non résolus*

Cela nous amène donc à poser des objectifs précis concernant les capacités de notre programme. Ce programme devra en effet être en mesure de :

* Afficher un patron de cube qui évolue au fil des mouvements utilisées pour la résolution, dans l’interface.
* Générer un Rubik’s Cube 3X3X3 mélangé de manière totalement aléatoire, en cliquant sur le bouton « Mélanger » de l’interface.
* Résoudre le Rubik’s Cube généré depuis n’importe quelle position, en cliquant sur le bouton « Résoudre » de l’interface.
* Résoudre le Rubik’s Cube uniquement avec les mouvements standards imposés par le produit réel, et indiquer tous les types de mouvements utilisés lors de la résolution sur l’interface.
* Pouvoir afficher les mouvements de résolution un par un, et mettre en pause la résolution depuis l’interface. Pouvoir basculer d’une étape de résolution à une autre depuis l’interface.

Ce sont les objectifs élémentaires de notre projet, mais en fonction de l’avancée des travaux, il peut être également envisagé de :

* Au lieu d’un patron du cube en 2 dimensions, afficher un cube en 3 dimensions, ce qui serait plus agréable pour l’utilisateur.
* Permettre à l’utilisateur de manipuler le Rubik’s Cube directement depuis l’interface, sans passer par la résolution programmée, ce qui permettrait d’apprendre en s’essayant soi-même à la résolution.

Ce programme présente donc naturellement quelques limites.

Comme évoqué précédemment, il ne concerne donc que les Rubik’s Cube de dimensions 3X3X3. De plus, bien que nous soyons en mesure de résoudre le Rubik’s Cube depuis n’importe quelle position, nous ne pouvons garantir le fait que cette résolution utilisera un nombre minimal de mouvements (les algorithmes utilisés pour la résolution du Rubik’s Cube sont très complexes d’un point de vue mathématiques, et certains font même encore l’objet de débats à l’heure actuelle).

Nous avons décidé, dans un objectif de meilleure gestion de l’avancée du projet, de travailler sur l’outil GitHub. C’est un service web d’hébergement et de gestion de développement de programmes, qui permet notamment une meilleure communication dans les projets par le biais de fonctionnalités telles que : le signalement de problèmes dans le code (« issues »), le partage de morceaux de code, la modification de code, …

Notre journal de bord de l’avancée des travaux pourra d’ailleurs également être tenu sur GitHub.

Finalement, l’ensemble de ce projet devra être rendu pour le 13 avril 2020.

Description du problème posé

Comme évoqué dans le cahier des charges, nous cherchons à proposer un programme permettant de résoudre un Rubik’s Cube 3X3X3, généré au préalable de manière aléatoire.

Les principales problématiques envisagées concernaient notamment :

* Créer un Rubik’s Cube 3X3X3 avec ses 6 couleurs différentes, créer les différents mouvements que l’on retrouve dans la réalité, et gérer les conséquences de ces mouvements qui impactent toutes les positions du Rubik’s Cube.
* Proposer un algorithme étant capable, étape par étape, de résoudre un Rubik’s Cube depuis n’importe quelle position. C’est-à-dire, à chaque étape de résolution, de pouvoir repérer les configurations du Rubik’s Cube dans lesquelles on se trouve pour pouvoir lancer les mouvements de résolution adéquats.
* La mise en relation directe des positions successives du Rubik’s Cube au fil des mouvements ordonnés, avec l’affichage proposé à l’utilisateur.
* …

Principe de l'algorithme

Notre programme résout les différents problèmes rencontrés de la manière suivante :

Tout d’abord, au niveau de la résolution pure du Rubik’s Cube, nous avons procédé de la façon suivante : nous avons divisé la résolution en différentes étapes (1ère face-1ère couronne ; 2ème couronne ; OLL-Orientation of Last Layer ; PLL-Permutation of Last Layer). Pour chacune de ces étapes, notre idée était de d’abord repérer la configuration dans laquelle on se trouvait (pour cela, nous avons créé des outils de caractérisation comme représenté sur la figure ?).

*Une image contenant capture d’écran, noir, assis, tenant

Description générée automatiquement*

*Figure ? : Exemple de différentes classes utiles pour repérer les configurations*

Une fois la configuration trouvée, il suffit de lancer les mouvements de résolution correspondants (que l’on a trouvé dans nos ressources bibliographiques).

Par ailleurs, concernant le principe de fonctionnement des mouvements, c'est le cube qui appelle la méthode mouvement. Et en appelant la méthode mouvement, il appelle la méthode mouvement des pièces. Ainsi, si la pièce appartient à la face qui est bougée, alors les autres parties de la pièce seront bougées également, ce qui nous permet d’avoir les mêmes mouvements que dans la réalité.

Puis, au niveau de la génération du cube en lui-même, nous avons crée les outils indiqués sur la figure ??. Ces outils nous ont permis de générer un Rubik’s Cube cohérent avec la réalité, en nous permettant de donner des conditions de génération (exemple simple : une pièce ne peut pas avoir deux facelettes de la même couleur).

Bibliographie :

* + Auteur : ' cyril ', « Le Rubik's Cube facile ? Vraiment ? », f*rancocube.com*, (depuis 1998), <https://www.francocube.com/cyril/rubik_index>, (page consultée le 9 mars 2020)
  + Auteur : ZEMDEGS, Feliks, « PLL Algorithms », *cubeskills.com*,(depuis 2006), <https://www.cubeskills.com/tutorials/pll-algorithms>, (page consultée le 9 mars 2020)
  + Auteur : ZEMDEGS, Feliks, « OLL Algorithms », *cubeskills.com*,(depuis 2006), <https://www.cubeskills.com/tutorials/oll-algorithms>, (page consultée le 9 mars 2020)
  + Auteur : ' hkociemba ', « [RubiksCube – Two Phase Solver](https://github.com/hkociemba/RubiksCube-TwophaseSolver) », g*ithub.com*,(depuis septembre 2019), <https://github.com/hkociemba/RubiksCube-TwophaseSolver>, (page consultée le 9 mars 2020)
  + Auteur : Wikipedia Contributors, « Algorithmes optimaux de résolution du Rubik's Cube », w*ikipedia.org*, (dernière modification le 23 novembre 2019), [https://fr.wikipedia.org/wiki/Algorithmes\_optimaux\_de\_r%C3%A9solution\_du\_Rubik%27s\_Cube](https://fr.wikipedia.org/wiki/Algorithmes_optimaux_de_résolution_du_Rubik%27s_Cube), (page consultée le 9 mars 2020)
  + Auteur : ' Herbert Kociemba ', « Kociemba's Homepage », *kociemba.org*,(mars 2016), <http://kociemba.org/index.html>, (page consultée le 9 mars 2020)
  + Auteur : ' Jaap's ', « Thistlethwaite's 52-move algorithm », *jaapsch.net*, (décembre 2014), <https://www.jaapsch.net/puzzles/thistle.htm>, (page consultée le 9 mars 2020)
  + Auteur : JOYNER, David, « Adventures in Group Theory : Rubik's Cube, Merlin's Machine, and Other Mathematical Toys », *Johns Hopkins University Press*, 2002, ([ISBN](https://fr.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-8018-6947-1](https://fr.wikipedia.org/wiki/Spécial:Ouvrages_de_référence/978-0-8018-6947-1)), (consulté le 9 mars 2020)
  + Auteur : HYPOLITE, Damien, « Le Rubik's Cube a 40 ans », *sciencesetavenir.fr*, 19 mai 2014, <https://www.sciencesetavenir.fr/decryptage/le-rubik-s-cube-a-40-ans_37858>, (consulté le 9 mars 2020)
  + Auteur : Anonymous Contributor, « Permutation of the Last Layer », *speedsolving.com*, janvier 2020, <https://www.speedsolving.com/wiki/index.php/PLL>, (consulté le 9 mars 2020)
  + Auteur : HERBY, Cyrille, « Interagissez avec des boutons », *openclassrooms.com*, février 2020, <https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/23727-interagissez-avec-des-boutons>, (consulté le 9 mars 2020)
  + Auteur : 'Oracle', « Java™ Platform, Standard Edition 7API Specification », *oracle.com*, (depuis 1993), <https://docs.oracle.com/javase/7/docs/api/>, (consulté le 9 mars 2020)
  + Auteur : 'Oracle', « Package javax.swing », *oracle.com*, (depuis 1993), <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>, (consulté le 9 mars 2020)

Structuration du programme et des données

Notre programme est structuré autour de deux dossiers principaux : le dossier Java et le dossier Ressources.

Dans le dossier Ressources, on retrouve des outils nécessaires à la compilation du code, tel que certaines images, certains outils comme les formules de résolution, …

Le dossier Java contient le code de notre programme. On y retrouve notamment le Cube, l’IHM et le Solveur.

Dans le dossier Cube, on retrouve Cube, Face, Facelette, et Pièce. Ce sont les outils que nous avons utilisés pour représenter notre Rubik’s Cube de la manière la plus simple possible.

Dans le dossier IHM, on retrouve tout d’abord évidemment le dossier AffichageCube qui se charge de la représentation du patron du Cube à l’utilisateur. Ensuite, on a un autre dossier qui se charge du reste de l’interface : positionnement des boutons, jauge d’avancée de la résolution, …

Enfin, dans le dossier Solveur on recense les différentes étapes de la résolution (ce qui permet ensuite une résolution plus ludique à l’utilisateur car il peut basculer d’une étape de résolution à une autre à son gré). Il y a le solveur pur, qui est évidemment chargé de résoudre le Rubik’s Cube depuis n’importe quelle position.

(n’hésitez pas à rajouter, j’ai dû en oublier ou me tromper)

Suggestions d’améliorations

Une première piste d’amélioration de notre programme réside dans l’amélioration de nos méthodes de résolution du Rubik’s Cube 3X3X3. Et pour cela, nous nous étions intéressés aux « Méthodes de Kociemba » (Kociemba est un mathématicien allemand qui a dédié une grande partie de sa vie à la recherche mathématique autour du Rubik’s Cube). Nous avons fait beaucoup de recherches dessus comme l’indique notre bibliographie, mais cela s’est finalement avéré être trop compliqué : nous n’arrivions pas à comprendre les méthodes au niveau mathématiques, ce qui rendait leur intégration dans le programme impossible.

De plus, nous ambitionnions initialement de proposer un affichage du Rubik’s Cube en 3 dimensions, pour que notre programme soit plus ludique, nous avons alors fait des recherches sur les solutions possibles, mais finalement cela aurait demandé énormément de temps travail en plus, donc nous avons préféré nous concentrer sur d’autres problèmes plus essentiels. Nous avons donc décidé de laisser pour l’affichage, le patron du Rubik’s Cube en 2 dimensions.

(voir plus)

Carnet de Route

Nous avons décidé, comme précisé dans le cahier des charges, de travailler avec l’aide de l’outil GitHub. GitHub offre la possibilité de suivre l’évolution du programme au fil du temps. Vous trouverez donc ci-dessous le lien URL du suivi de l’avancée du programme :

<https://github.com/Lgt2x/Rubik-s-Cube-Java/commits/master>