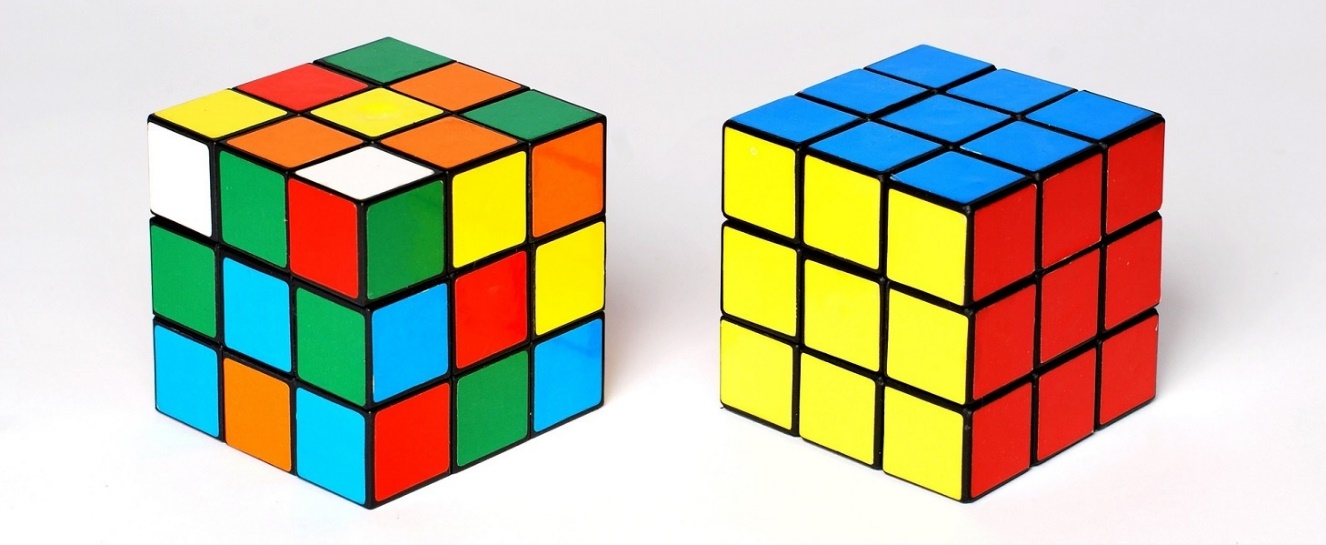
**Mini-projet 2A : Rubik’s cube**

Complément au code

**Présentation**

Le Rubik’s Cube est un jeu proposé sous la forme d’un cube possédant des faces pouvant se déplacer indépendamment les unes des autres (voir figure 1). L’objectif du Rubik’s Cube est de reconstituer le cube tel que ces faces soient chacune de couleur unie. 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 intéressante l’idée de proposer un programme permettant une résolution guidée, au rythme voulu par l’utilisateur de ce casse-tête à la fois populaire, complexe et fascinant.

*Figure 1 : Exemple de comparaison entre un Rubik’s Cube 3x3x3 résolu (droite) et non résolu (gauche)*

***Cahier des charges et limites***

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.
* Pouvoir mélanger aléatoirement ou manuellement le cube.
* Résoudre le Rubik’s Cube généré depuis n’importe quelle position en affichant l
* 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.
* Proposer un lecteur complet de la solution pour pouvoir visualiser la solution pas à pas, en avançant ou reculant d’un mouvement ou d’une étape à la fois.

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 du suivi de l’avancement du projet, de travailler sur Github, basé sur git. Cela nous permettra à la fois de pouvoir visualier l’avancement du projet en temps réel grâce au système d’historique de commit, mais aussi de gérer la progression du projet grâce à l’onglet « issues », qui permet le reporting de bugs et propositions d’amélioration, couplé aux « merge requests » qui permettent la validation du code pour coordonner l’équipe.

**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 général de fonctionnement de la résolution**

Expliquons maintenant le fonctionnement global de l’application, en passant par sa structure de classes. Notons déjà que sa compilation est gérée par Gradle, qui se lance grâce au wrapper (voir Readme).

Le premier problème auquel nous avons été confrontés a été la représentation informatique du Cube, sa structure atypique ne rendant pas la tâche aisée. Nous avons opté pour une structure par pièces physiques immobile, qui contiennent des facelettes changeant de couleur au gré des mouvements. 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. La figure ci-dessous présente la structure de classe de représentation du cube.

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

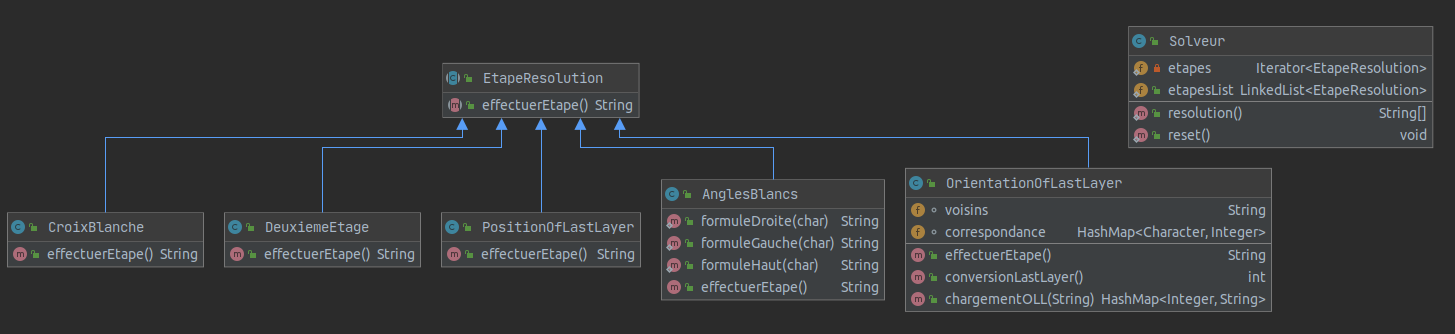
Description générée automatiquement

*Figure 2 : Exemple de différentes classes utiles pour repérer les configurations*

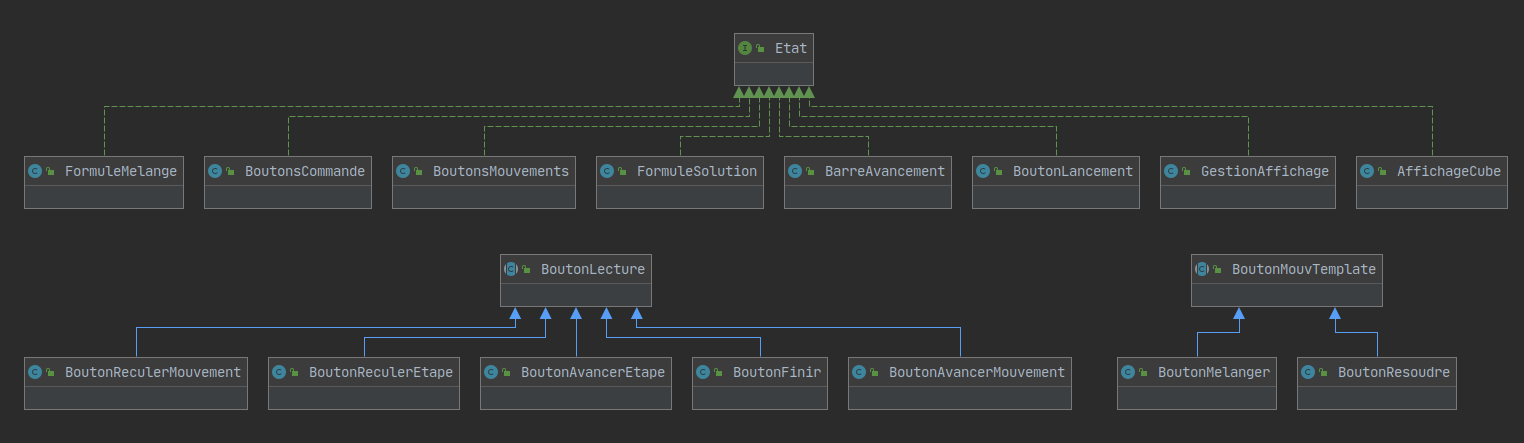
Puis, au niveau de la résolution pure du Rubik’s Cube, nous avons divisé la résolution en plusieurs étapes, comme pour une résolution classique à la main : 1ère face-1ère couronne, puis 2ème couronne, puis OLL (Orientation of Last Layer) et enfin PLL (Permutation of Last Layer). Les deux premières couronnes sont assez systématiques, nous nous sommes inspirés de la résolution à la main pour automatiser leur fonctionnement. Pour les deux dernières étapes, notre idée est de d’abord repérer la configuration dans laquelle on se trouve, puis d’appliquer la formule correspondante. Pour cela, nous avons créé des fonctions convertissant la configuration du cube en chaine binaire, pour être comparée aux configurations standards classiques trouvées dans la littérature.

**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 fichiers annexe : certains outils comme les formules de résolution pour les dernières étapes. 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 présentés précédemment. Le Cube contient des pièces qui sont composées de facelettes, qui appartiennent à des faces. La résolution est gérée par la classe Solveur, qui appelle tour à tour les différentes étapes dérivant d’une classe abstraite EtapeResolution.

 *Figure 3 : Organisation des classes de la résolution*

Dans le dossier IHM, on retrouve tout d’abord le dossier AffichageCube qui se charge de la représentation du patron du Cube à l’utilisateur. L’affichage est géré par la classe GestionAffichage, qui met à jour tous les éléments en appelant leur fonction d’actualisation. Une interface est utilisée, Etat, implémentée par tous les composants de l’IHM (dans ComposantsUI), qui doivent avoir une fonction d’actualisation. L’application possède deux états, « mélange », activé au lancement de l’app, qui laisse libre cours aux mouvements de mélange libre (les lettres) et désactive le lecteur, ainsi que « resolution », activé après avoir appuyé sur « Résoudre » qui active les boutons de lecture.

*Figure 4 : Organisation des classes de l’IHM*

Pour uniformiser le style, une classe abstraite définissant le style des boutons a été créée, BoutonMouvTemplate, dont dérivent tous les boutons de l’interface.

**Pistes d’amélioration**

Une première piste d’amélioration de notre programme réside dans l’amélioration de nos méthodes de résolution du Rubik’s Cube 3x3x3. En effet, la méthode utlisée, calquée sur la résolution manuelle n’est pas la plus optimisée en terme de mouvements. N’importe quelle configuration du cube peut être résolue en moins de 20 mouvements, or notre solveur se utilise en moyenne 150 mouvements. Et pour cela, nous nous étions intéressés aux « Méthodes de Kociemba », qui proposent une résolution par force brute en deux étapes, garantie en moins de 20 mouvements. Nous avons fait beaucoup de recherches dessus comme l’indique notre bibliographie, mais cette solution s’est avérée trop compliquée pour l’ampleur de notre projet, et inintéressante du point de vue technique.

De plus, une de nos ambitions initiales était 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.

**Carnet de Route**

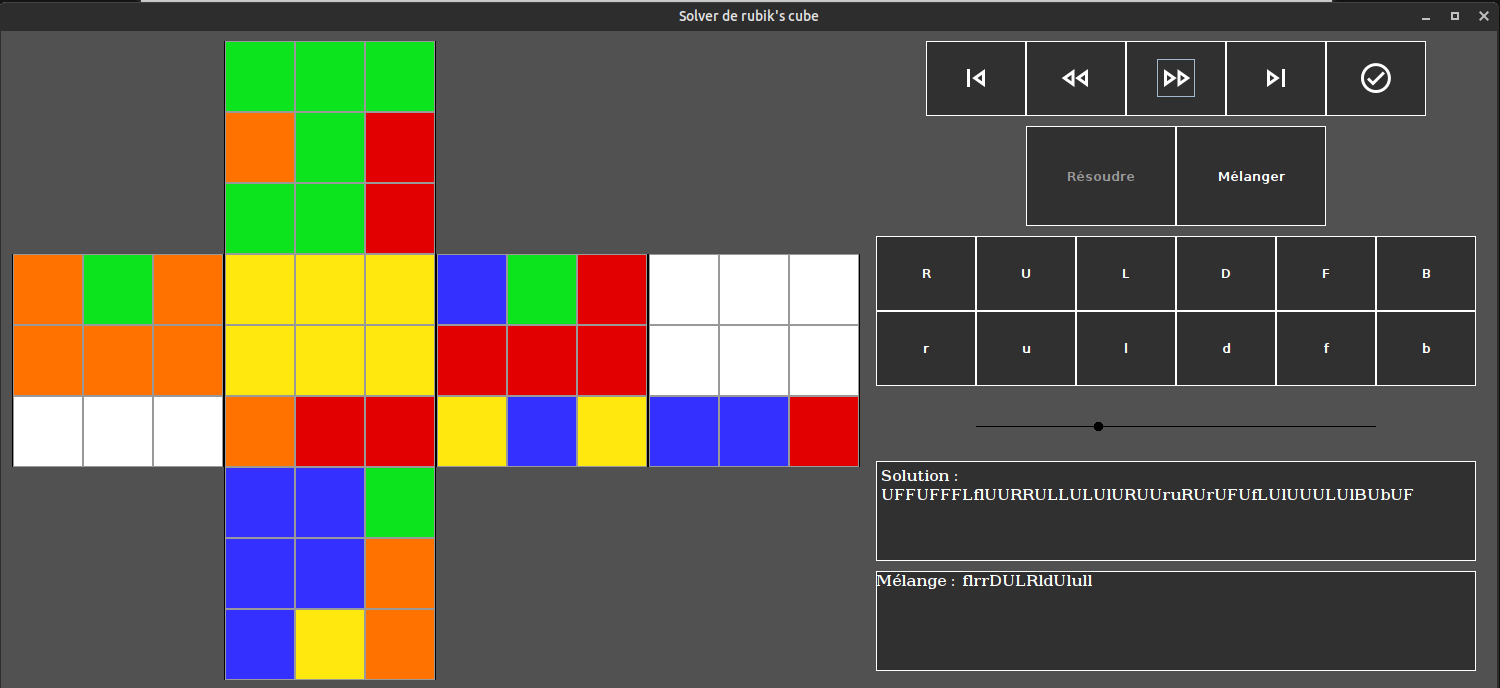
Nous avons décidé, comme précisé dans le cahier des charges, de travailler avec l’aide de Github, qui offre la possibilité de suivre l’évolution du projet au fil du temps. Vous trouverez donc ci-dessous le lien URL du suivi de l’avancée du programme :

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

Sous l’onglet releases est proposée une version compilée

**Conclusions**

Ce projet nous aura surtout beaucoup appris sur le travail en groupe et l’organisation sur git, ainsi que l’organisation en temps et en classes d’une application plus conséquente qu’un programme de TP. L’expérience fut convaincante pour l’ensemble du groupe.



*Figure 5 : Aperçu du rendu de l’application*

**Bibliographie**

Voici une partie des références utilisées pour la conception du logiciel, ayant apporté des pistes de réflexion ainsi que des solutions concrètes au problème posé.

* + 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's_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 : 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 : 'Oracle', « Java™ Platform, Standard Edition 7API Specification », *oracle.com*, (depuis 1993), <https://docs.oracle.com/javase/7/docs/api/>, (consulté le 9 mars 2020)