RAPPORT - PROJET INFORMATIQUE

VISUALISATION D'ALGORITHME

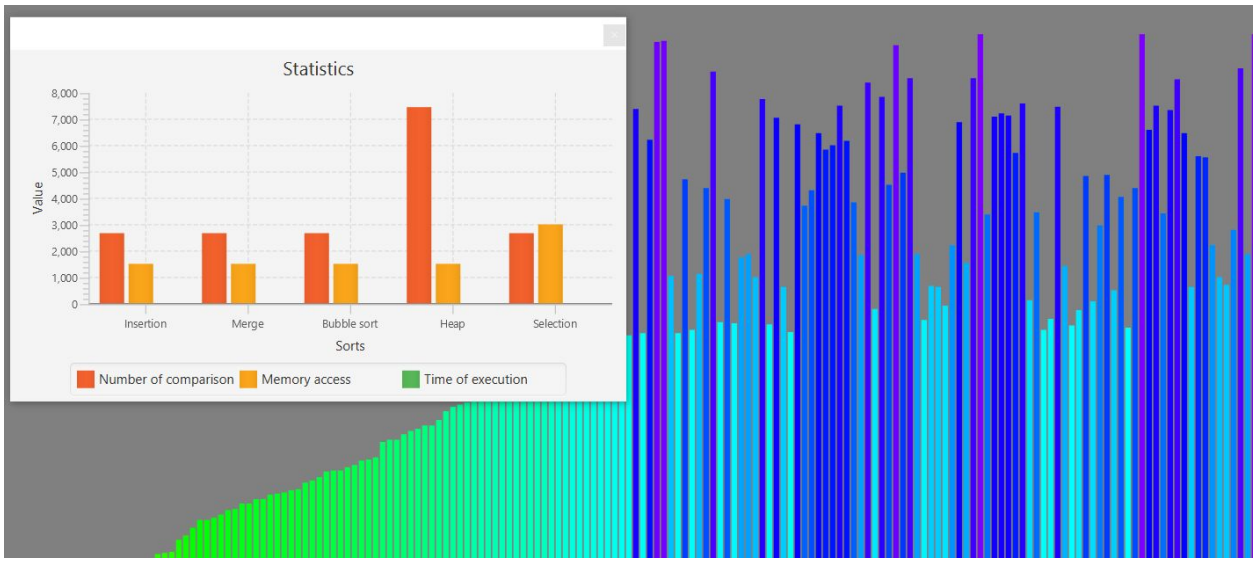


Fig.1: Premier aperçu de l'application.

Hind **HALIMA**

Joris **IDJELLIDAINE**

Lucian **PETIC**

L2 - Groupe 3

SOMMAIRE

1. DESCRIPTION DU PRODUIT

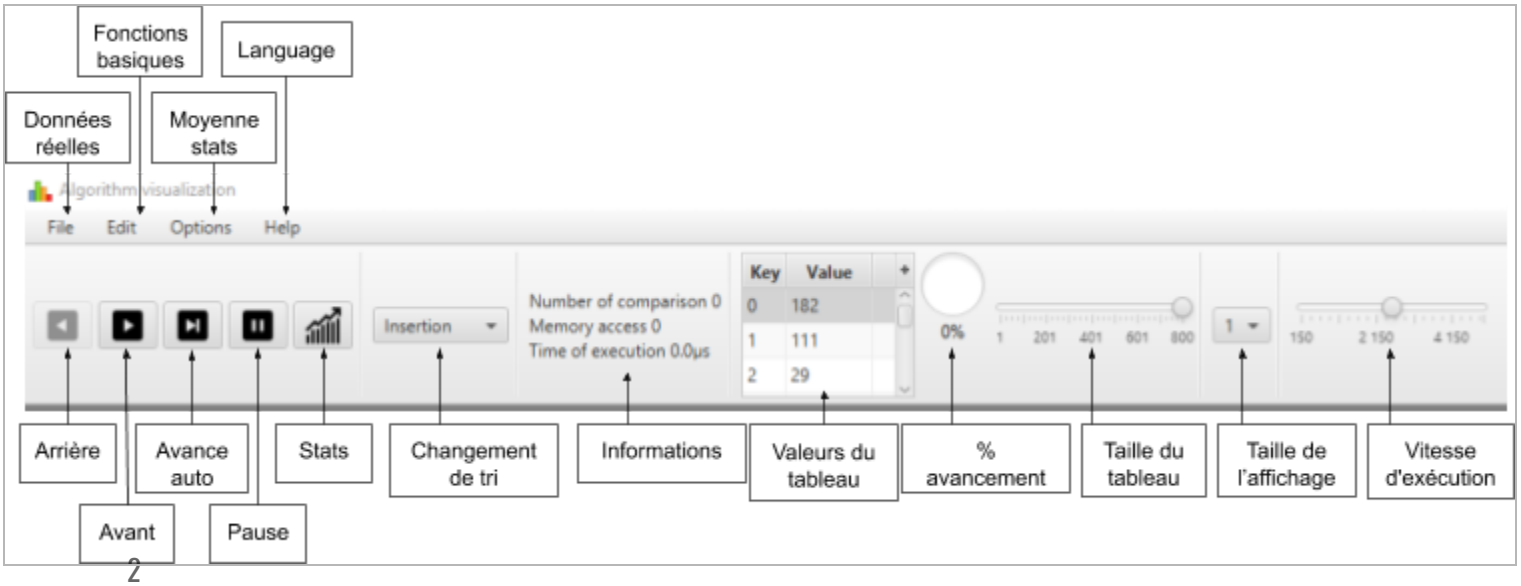
Le modèle MVC[2] (modèle, vue, contrôleur) a été utilisé pour ce projet. De plus, les différents fichiers du projet sont séparés en fonction de leurs utilitée et tous les fichiers annexes (données réelles, langage), sont aussi séparés pour un maximum de clarté.

a. Fonctionnalités

Le programme possède toutes les fonctionnalitées nécessaire afin d'analyser le déroulement d'un tri en temps réel. Il est capable de créer aléatoirement un tableau d'une taille variant de 1 à 960 éléments (limite imposée pour des raisons de performances), de trier ce tableau avec un tri donné et de parcourir le tri effectué en étape par étape de plusieurs façon différentes.

Les étapes peuvent être parcourues automatiquement avec un temps entre chaque étape réglable, ou bien de façon manuelle avec une étape par clic. Il est aussi possible de mettre en pause l'avance automatique ainsi que de revenir en arrière dans le tri, et chaque étape du tri contient des informations sur le temps d'exécution, le nombre de comparaisons et le nombre d'accès en mémoire. Il existe aussi un indicateur du pourcentage de complétion du tri et un petit tableau avec une barre de défilement affichant toutes les valeurs contenues dans le tableau affiché à l'écran.

Il est aussi possible de changer la langue, la taille d'affichage du tableau et le tri choisis simplement via l'interface graphique, ainsi que de charger des statistiques sur le tableau affichants différentes informations sur l'efficacité de chaque tri. D'autre part, le programme supporte l'utilisation de données réelles qui peuvent être importées à partir de fichiers Excel et triées en sélectionnant une colonne spécifique. L'importation de données réelles supporte tous types de fichiers et trie aussi bien le texte par ordre alphabétique que les nombres.



b. Modèle

Le modèle a pour but d'exécuter les tris et de renvoyer les informations nécessaire afin d’afficher entièrement le tri en étape par étape. Afin de réaliser cette tâche, les informations sont réunies en “Bloc”, qui est une classe contenant les informations pour une étape (tableau et stats). Ainsi, l’affichage d’une étape du tri est géré par un seul bloc et le reste des étapes sont stockés dans une `LinkedList` de `Bloc`. Ce fonctionnement permet de facilement parcourir un tri avec peu de calculs ainsi que de faciliter le retour en arrière au milieu du tri étant donné qu’il s’agit de juste reprendre le bloc précédent.

Ce fonctionnement permet d’afficher n’importe quel tri tant que celui-ci nous permet d’obtenir une `LinkedList` de `Bloc`. Pour obtenir cette `LinkedList`, la plupart des tris se contentent juste de créer et ajouter des objets `Bloc` à cette `LinkedList` durant leurs exécution ce qui permet d’obtenir un résultat correct. D’autres tris demandent un peu plus d’efforts, c’est notamment le cas du tri par fusion, qui demandent bien plus d’étape afin de reconstituer le tableau fragmenté pour créer son `Bloc` et l’ajouter à la LinkedList.

Chaque tri hérite de la classe abstraite `Tri`, qui contient plusieurs informations utiles à l’exécution du tri, ainsi que la LinkedList de Bloc. C’est de cette classe dont hérite chaque tri, et la fonction `execution()` est remplacée dans chaque tri par le code permettant de trier un tableau d’entier.

Il existe beaucoup d’autres fonctions dans le modèle de notre projet, notamment pour l’utilisation des données réelles qui utilise des HashMaps afin de recomposer un tableau Excel en tableau d’entier triable, mais le coeur du programme est basé autour de l’utilisation de Bloc pour l’affichage du tableau et de sa création pour chaque tri. En somme, il est relativement facile d’ajouter de nouveaux tris au modèle car il suffit de simplement écrire la fonction `execution()` du tri pour que celui-ci fonctionne correctement.

c. Vue

La vue est basée entièrement sur le framework et la bibliothèque `JavaFx`. Cette dernière a été choisie pour sa bibliothèque extrêmement complète et sa facilitée d’utilisation. Ce projet aura été d’autre part l’occasion de passer de Swing (enseigné au premier semestre) à `JavaFx`. Cette bibliothèque était bien plus simple et intuitive que celle de `Swing` et elle nous a permis de réaliser notre interface graphique de manière organisée. En séparant le modèle de la vue, nous avons eu plus de facilitées au niveau de l’organisation. Nous avons d’autre part pu profiter des fonctions déjà présentes dans le framework de `JavaFx` afin d’illustrer notre application.

La vue est composées de 3 éléments. Chacuns de ces éléments fait partie de la vue et utilise le contrôleur principal afin de gérer les interactions avec l’utilisateur. Le code utilisé pour la création des éléments à l’écran est assez bien organisé et il serait même possible de le réutiliser pour de futurs projets.

Toutes les interactions utilisateur/interface graphique passent par le contrôleur. En fonction des entrées de l’utilisateur, une fonction donné est lancée grâce au contrôleur. Ainsi, le programme est constamment à l’écoute de nouveaux événements, et c’est cela qui permet à l'utilisateur d’interagir avec la vue. Nous avons par ailleur introduit différents raccourcis afin faciliter l’UX (user experience) et d’éviter que l'utilisateur ne doive toujours utiliser l’interface graphique pour ses actions .

Une autre fonctionnalité disponible dans le projet a été l’introduction du de changement de langue. Un système de clé/valeur nous a paru la solution la plus adéquate pour ce système. Tout se passe dans le fichier `Resources` qui s’occupe du bon affichage de la langue de l’utilisateur et l’implémentation de cette fonctionnalité nous a permis d’en apprendre plus sur le fonctionnement `HashMap`, chose dont nous ignorions encore le fonctionnement jusqu'à présent.

2. RÉPARTITION DES TÂCHES

a. Organisation

Notre répartition durant ce projet a été la même que celle communiqué dans le cahier de charge initial, la répartition des tâches est donc :

Hind **HAMILA**	Création et implémentation des classes de tri.
Joris **IDJELLIDAINE**	Logique de la visualisation, fonctionnalitées et gestion du modèle.
Lucian **PETIC**	Interface graphique JavaFX, liaison entre le modèle et la vue.

b. Utilisation de git

Lors du commencement du projet, un document a été réalisé: "Organisation, conventions et factorisation du code"[1] afin faciliter l'organisation de l'équipe. Ce document évoque principalement le besoin de commenter son code mais aussi suivre les conventions de nommage et l'organisation générale du projet.

Le gestionnaire de code git nous aura vraiment été utile étant donné le fait que notre équipe soit composée de 3 collaborateurs. Le système de tickets[4] nous a également permis de se désigner des tâches, rapporter les bugs ou bien faire des notices sur les prochaines fonctionnalitées à introduire.

D'autre part l'utilisation du git nous permettait de aisément observer l'avancement des parties de chacun grâce aux commits. Ces derniers donnent un rapide aperçu des nouveautés du code apporté par les membres du groupe.

3. DIFFICULTÉS RENCONTRÉES ET PARTIES PERSONNELLES

Hind **HAMILA :**

Les premières difficultés que nous avions rencontrés étaient en étroit rapport avec la mémoire occupée par les données conservées dans des tableaux d'entiers, ces dernières servant à la fluidité d'utilisation du programme (pouvoir voir les tris par étape, revenir sur une étape passée) on a, à contrario de ce que l'on voulait, ralenti l'exécution du programme, ce qui nous a alors incité à utiliser d'autres stratégies d'optimisation de la mémoire qui se trouvaient plus efficaces.

Toutefois nous avions dû redéfinir à plusieurs reprises le modèle à cause de plusieurs incompatibilités entre les tris que l'on voulait ajouter à notre projet, ainsi que le modèle en lui-même : la classe `Fusion.java` par exemple, prenant en charge les fonctions du tri fusion est plus complexe que la classe `Selection.java` n'utilisant qu'une fonction pour le tri par sélection.

On peut donc remarquer un certain écart de difficulté entre l'implémentation de tris "naïfs" comme les tri par sélection et par insertion ainsi que les tris plus complexes requérant l'utilisation d'autres tris comme `TimSort` qui joint tri fusion et tri par insertion à la fois, nous obligeant à changer la classe abstraite Tris.java pour qu'elle puisse être plus flexible à utiliser dans le cas d'implémentation de tris complexes.

L'implémentation de `TimSort` a été un élément primordial au projet, en effet, étant un hybride du tri par insertion et du tri fusion, il est largement plus efficace sur des données réelles sachant que sa complexité est linéaire (dans les meilleurs cas) nous facilitant donc leur tri.

Par ailleurs, l'utilisation de statistiques après exécution des tris donne beaucoup d'informations quant à la complexité de ces derniers ainsi que leur efficacité sur le tableau donné. L'utilisation de HashMap a également été primordial dans la réalisation de la classe `StatMaker.java`.

Joris **IDJELLIDAINE :**

Au tout début du projet, lors de la mise en place du modèle, le coté Bloc de données était déjà abordé mais le fonctionnement du projet était vastement différent. Initialement, le modèle prenait en argument un tableau et appliquait le code d’un tri à celui-ci afin de renvoyer le tableau qui serait affiché dans l’interface graphique. Pour revenir en arrière, le code de chaque tri était divisé en “étape”, en quelque sorte, et une fonction gardait en mémoire l’étape à laquelle se trouvait le tri.

Cette méthode fonctionnait relativement correctement pour les tris les plus simples comme le tri par insertion ou le tri par sélection, mais quand il nous a fallu faire le tri par fusion, c’est là que je me suis rendu compte des limitations imposées par ce modèle. Après une très grosse restructuration du modèle et du code, je suis arrivé à ce modèle actuel qui fonctionne très bien que ce soit du côté performances ou ajouts de fonctionnalités. En effet, tous les calculs sont fait directement lors de la création du tableau et de chaque changements, ce qui diminue la charge, et simplifie le code étant donné qu’il ne faut plus se soucier de pouvoir revenir en arrière.

Nous avons aussi eu beaucoup de difficulté sur l’implémentation de certains tri. La plupart des tris en place ne posent pas de problème et il suffit de rajouter une fonction au bon endroit pour que ceux-ci fonctionnent, mais d’autres tris comme le tri fusion ou le tri `TimSort` qui ont une complexité spatiale plus grande ont tendances à poser beaucoup plus de problème pour reformer un tableau qui représente correctement les changements effectués par le tri.

Ces difficultées sont par ailleurs visibles dans la complexité des fichiers de tri fusion et `TimSort`, ces deux derniers sont beaucoup plus volumineux que les tris plus simples, tel que le tri à bulle ou le tri par sélection. Malgré cette complexité, il est toujours possible de les représenter normalement en stockant les tableaux crées puis en les couplants ensemble afin de reformer le tableau initial, le tri fusion est de loin le tri qui nous aura été le plus difficile à implémenter dans le projet, étant donné qu’il n’existe que peu d’exemple sur Internet permettant de recréer le tableau comme on souhaitait le faire.

Bien d’autres problèmes mineurs auront été résolus au cours de la création de notre programme, notamment au niveau de la création de statistiques et des données réelles. Les fonctions permettant d’obtenir les valeurs demandés auront subis BEAUCOUP de modifications au fur et à mesure du temps, et il n’aura pas été rare de penser qu’une version était correcte avant de se rendre compte après coup après une erreur que sa fondation était mauvaise et qu’il fallait alors réécrire celle-ci correctement.

Lucian **PETIC** :

Dans un premier temps, la première difficultée rencontrée aura été de créer une architecture qui saurait répondre à nos besoins. Le modèle MVC était une évidence, mais il restait tout de même des parties à organiser. Le code source est assez bien organisé à présent et il est relativement facile de maintenir ou bien de rajouter de nouvelles fonctionnalités.

Les liaisons entre le modèle et la vue auront été un petit défi. Aujourd'hui, le contrôleur principal comporte de nombreuses méthodes, tandis qu'au début il était plutôt question de créer un contrôleur par composant, ce qui, après coup, n'était pas forcément une bonne idée, car les composants doivent communiquer entre eux et une telle séparation causerait plus de problème que de gain pour le code étant donné que cela impliquerait que le contrôleur A doive reconnaître le contrôleur B et vice-versa. Cette vision est assez maladroite et le contrôleur a rapidement été corrigé vers le début du projet.

Un autre problème rencontré aura été de savoir s'il fallait utiliser plutôt les interfaces ou bien directement les expressions lambda. Le fonctionnement global reste le même, mais l'organisation du code en est impacté. Le rendu final utilise, lui, des expressions lambda, étant donné qu'elles sont un peu plus intuitive et font appel à une autre fonction présente dans le contrôleur principal.

Durant le développement du projet, l'idée de l'animation pour l'interface graphique nous était relativement claire. Cependant, sa mise en place aura été particulièrement difficile, notamment à cause du code nécessaire pour démarrer ou arrêter l'animation [3]. L'intuition initiale était d'utiliser les fonctionnalités de la classe `Thread`, mais après beaucoup de problème lié à notre utilisation de cette classe, nous avons plutôt opté pour une classe spéciale faite pour les animation présente dans la bibliothèque JavaFx : `Timeline`.

La création des tableaux, statistiques ainsi que de la scrollbar pour l'interface graphique n'auront pas été évidents. Leurs fonctionnement a donné suite à beaucoup de confusion et nombreuses recherches sur le fonctionnement et l'implémentation de ces différentes fonctions.

Le passage d'un tri à l'autre pourrait paraître être une opération relativement simple, mais en réalité, celle-ci est bien plus délicate qu'il n'y, car il faut bien reprendre les données du tri précédent afin de pouvoir effectuer un changement sans accros.

4. LIENS

Liens	URL
Lien 0 GitHub & Gogs Project	https://github.com/lpetic/algo-view https://gogs.script.univ-paris-diderot.fr/hhamil29/PI4
Lien 1 Doc conventions	https://github.com/lpetic/algo-view/wiki/Organisation,-conventions-et-factorisation-du-code
Lien 2 MVC	https://www.screencast.com/t/GsE4WFI7N
Lien 3 Animation issue	https://www.screencast.com/t/o35VBBArpU0
Lien 4 Tickets	https://www.screencast.com/t/eIl5XUqtEqD