**Lancement du programme :**

Le programme a été développer en Java 13 avec Maven afin de faciliter l’écriture de celui-ci que ça soit pour la partie implémentation ou bien pour la partie graphique. De plus, nous avons choisi Maven afin d’importer si nécessaire des dépendances (ce qui était prévu à la base pour la partie graphique avec JGraphix), et d’avoir un lancement similaire indépendant de l’IDE.

Pour lancer le programme, il est donc nécessaire d’effectuer la procédure suivante :

* Installer maven
* Ouvrir un terminal et se placer dans le dossier
* Taper **mvn package** ou **mvn package exec:java -D"exec.mainClass"="mga.Launcher"**

Il est possible que la première commande ne fonctionne pour une raison inconnue.

Cela va lancer le programme avec les paramètres par défaut qui sont les suivant :

* Lecture du fichier de graphe « resources/JoliGraphe100.graphe »
* Lecture du fichier de coordonnée « resources/JoliGraphe100.coords »
* Écriture du résultat dans le fichier « resources/coloredGraph.colors »

Pour changer les fichier lus, il suffit de modifier les variables constantes de la class « Launcher » :

* graphFile correspond au fichier de graphe
* graphPointPosition correspond au fichier donnant la position des sommets du graphe.

Pour finir, le programme produira deux résultats :

* Le premier correspond aux couleurs des sommets (écrit dans le terminal)
* Le second correspond au graphe coloré affiché dans une fenêtre

On notera également que la taille du graphe sur la fenêtre n’est pas forcément adaptée. Pour changer cela, il est nécessaire de modifier la constante « graphSizeMult » dans la classe Launcher.

Plus cette variable est grande, plus le graphe sera grand.

**Les fonctions utilitaires**

Nous avons choisi d’implémenter quelques fonctions utilitaires en plus afin de faciliter l’écriture du code et en essayant de diminuer au maximum leurs cout

Fonction 1 : Algorithme -> echangerCouleur

Cette fonction s’effectue en temps constant ou au pire des cas en temps O(log(n)). Elle permet d’échanger deux éléments dans la map du graphe coloré.

Fonction 2 : Algorithme -> getUnusedColorInNeighbors

Cette fonction renvoie une liste des couleurs inutilisé par les voisins d’un sommet donné en paramètre. Elle s’effectue en temps O(n) ou au pire des cas, en temps O(n log(n))

Fonction 3 : Algorithme -> getRandomSommet

Renvoi un sommet aléatoire dans le graph donné avec une complexité de O(1)

Fonction 4 : Utils -> getColorName

Renvoi le nom d’une couleur sous forme de chaine de caractère plutôt que le toString classique de la classe Color de AWT.

**La partie graphique**

Nous avons choisi d’utiliser la bibliothèque awt + swing de java pour dessiner le graphe du fait de la simplicité de la fonction « paintComponent »

Tout d’abord on récupère les données du fichier .coords et du fichier .graph et on les placent dans deux listes contenant les tuples à 4 éléments afin de stocker toutes les informations nécessaires.

Ensuite, on transforme ces données afin d’adapter la taille du graphe.

Et enfin, on appel la fonction « paintComponent » qui permet de dessiner des formes sur la fenêtre (des ronds colorés pour les points et des traits noir pour les arêtes).