# Projet Java

Simon Mauras

9 Janvier 2015

## Table des matières



## 1 Introduction

Ce projet a été réalisé au cours du premier semestre de l'année de L3 au département d'informatique de l'ENS de Lyon. L'objectif étant de se familiariser avec le langage Java et différents algorithmes de Clustering.

Le sujet nous invitait dans un premier temps à réaliser une interface graphique de manière à valider le bon fonctionnement de notre implémentation en considérant un ensemble de points du plan. La seconde partie nous proposait un choix de différentes pistes à approfondir : utiliser un algorithme plus efficace pour le calcul des diagrammes de Voronoï, se placer dans un espace de dimension supérieure à 2 afin de pouvoir analyser des données de différents types, implémenter un algorithme de Clustering spectral, trouver une heuristique permettant de deviner le nombre optimal de Clusters, ...

J'ai choisi d'implémenter un algorithme suivant le paradigme "diviser pour régner" permettant de calculer une triangulation de Delaunay en temps $\mathcal{O}(n \log n)$. Avec cet algorithme, publié en 1980 par Lee et Schachter, il est possible de construire des diagrammes de Voronoï de manière efficace.

## 2 Organisation générale du projet

Dès le début du projet, j'ai essayé de bien séparer les classes responsables de l'interface graphique et les classes implémentant les différents algorithmes.

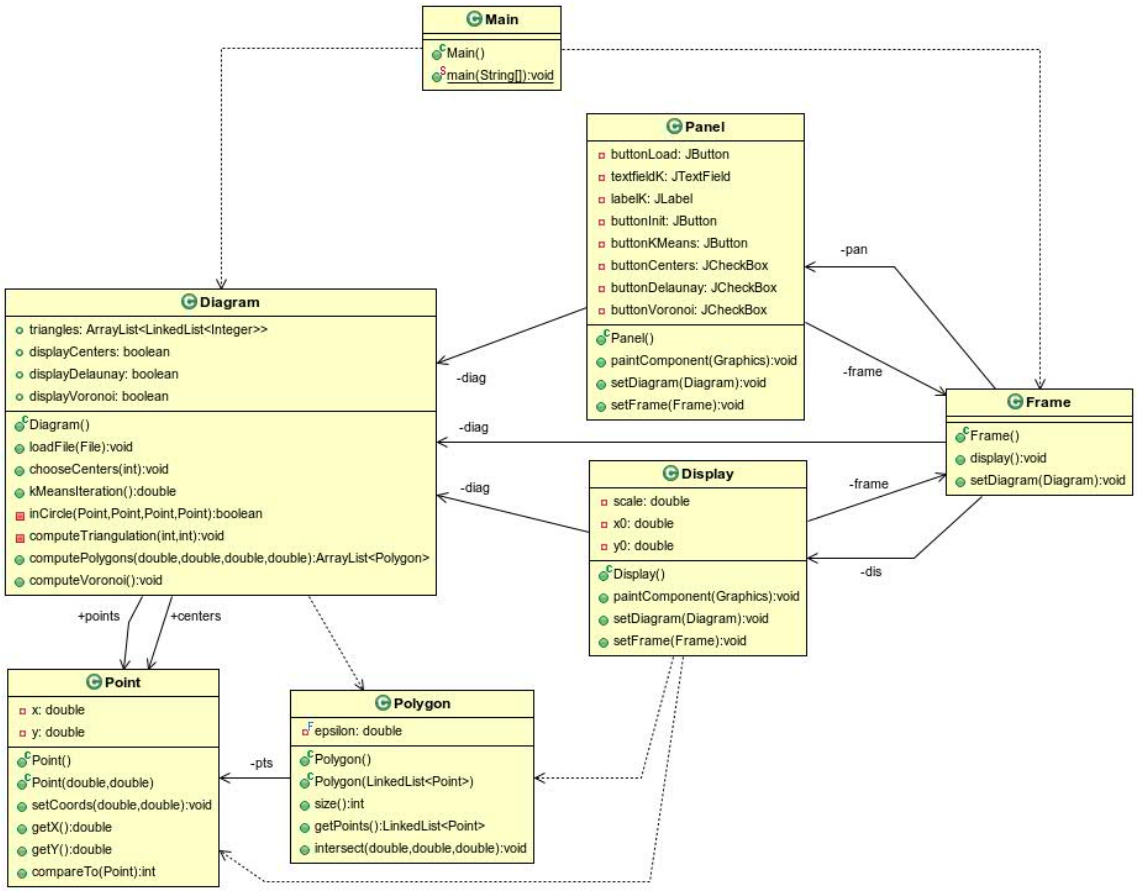


FIGURE 1 – Diagramme UML du projet.

La classe `Main` est appellée au début de l'exécution. Elle crée une instance de la classe `Frame` (responsable de l'affichage graphique) et une instance de la classe `Diagram` (effectuant les différents calculs).

La classe `Display` dessine les points et polygones, elle interagit avec la classe `Diagram` (pour obtenir les donnée à afficher) et la classe `Frame` (pour gérer le zoom et forcer le ré-affichage de la fenêtre).

La classe `Panel` contient les différents boutons et gère les évenements associés, elle interagit avec la classe `Diagram` (pour lancer les différents calculs) et la classe `Frame` (pour forcer le ré-affichage à la fin d'un calcul).

La classe `Diagram` implémente différents algorithmes (K-Moyennes, triangulation de Delaunay et diagramme de Voronoï). Elle est responsable de la lecture des données dans un fichier (`loadFile`). Les structures `Point` et `Polygon` sont utilisées pour représenter les données. Les fonctions `kMeansIteration` et `computeDelaunay` sont appellées toutes les 100ms par la classe `Panel` lorsqu'un calcul est en cours. La fonction `computeVoronoi` est appellée par la classe `Display` à chaque ré-affichage.

## 3 Lecture dans un fichier et algorithme des K-Moyennes

Nous avons choisi de pouvoir lire les points dans un fichier. Les générateurs (écrits en OCaml) ainsi que des données utilisées sont disponibles dans le dossier `test`. Les différents générateurs couvrent des cas particuliers.

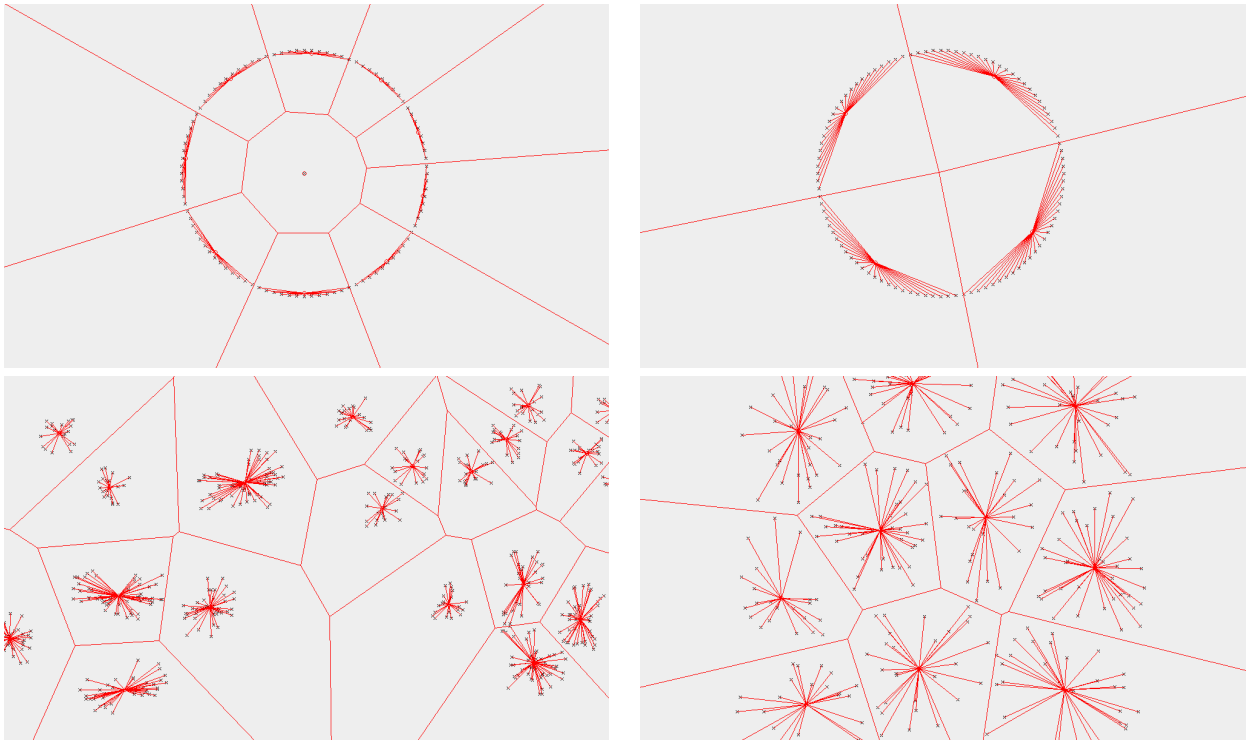

FIGURE 2 – Différents générateurs

Les centres sont actuellement choisis de manière aléatoire parmis les points. La fonction `kMeansIteration` effectue une seule modification des centres. L'animation consiste à itérer tant que la variation n'est pas devenue infime.

## 4 Triangulation de Delaunay

Une triangulation de Delaunay est le graphe dual d'un diagramme de Voronoï. Le passage de l'un à l'autre peut se faire de en temps linéaire. Nous avons décidé de commencer par calculer la triangulation de Delaunay de notre ensemble de centres avant de générer le diagramme de Voronoï, de manière à éviter les problèmes qui peuvent se poser avec les arrondis des nombres en virgule flottante.

L'algorithme que j'ai adopté est celui proposé par Lee et Schachter en 1980. Il s'agit d'un algorithme qui trie les sommets par abscisse croissante puis procède en suivant le paradigme "diviser pour régner" : il coupe l'ensemble en deux, calcule la triangulation dans les parties droite et gauche puis fusionne les deux en $\mathcal{O}(n)$. Une description plus complète est disponible en [1].

La principale difficulté que j'ai pu rencontrer est de bien comprendre comment gérer les cas ou plusieurs points ont la même abscisse, que ce soit pour le choix d'un ordre valide pour le tri et aussi lors de la fusion entre les parties gauche et droite qui s'effectue de bas en haut.

## 5 Génération du diagramme de Voronoï

Par la suite, générer le diagramme de Voronoï à partir d'une triangulation de Delaunay est en tout point identique à l'algorithme basique décrit dans le sujet à l'exception du fait qu'il nous suffit de calculer les intersection des demi-plans pour les voisins immédiats dans le graphe représentant la triangulation de Delaunay. Si on sait que le nombre d'arêtes dans un graphe planaire est linéaire en le nombre de sommet, le calcul de l'ensemble des cellules peut se faire de manière efficace.

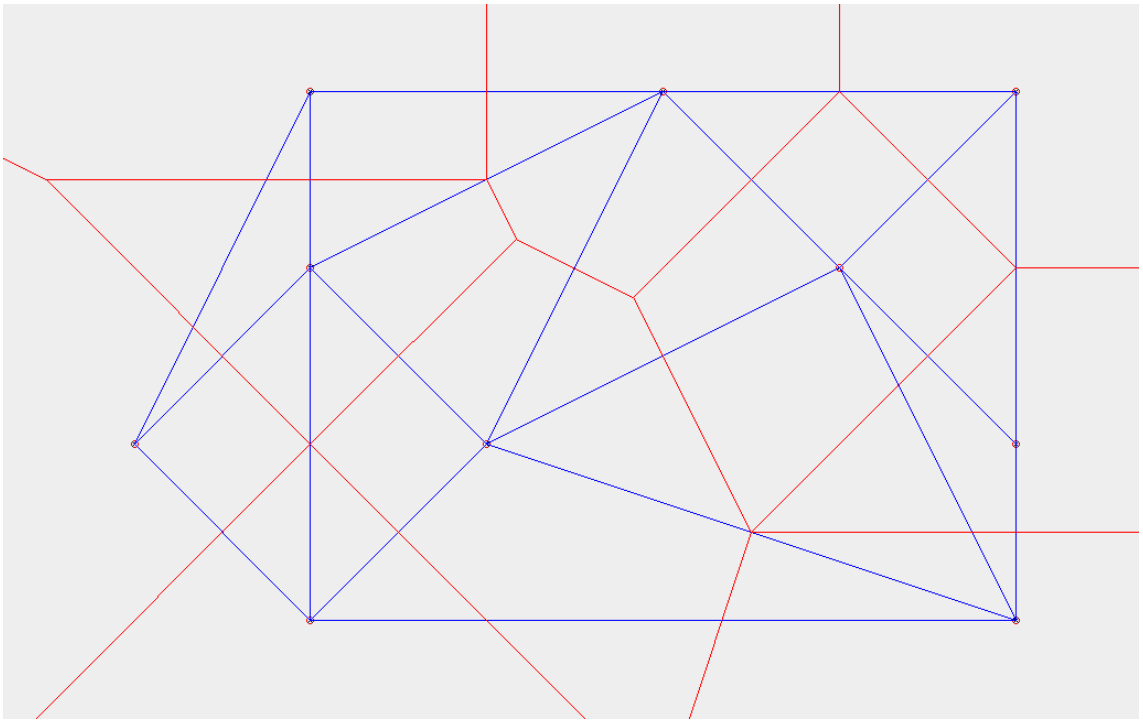

FIGURE 3 – La triangulation de Delaunay est le graphe dual du diagramme de Voronoi

# 6 Conclusion

J'ai tiré de ce projet plusieurs points enrichissants. Parmis eux la satisfaction de s'être plongé dans un article de recherche pour comprendre en profondeur un algorithme assez complexe. J'ai également appris à organiser mon code de manière cohérente en séparant l'affichage des calculs (pattern MVC malgré une séparation peu marquée entre Vue et Contrôle).

# 7 Références


– D.T. Lee, B.J. Schachter, "Two Algorithms for Constructing a Delaunay Triangulation", *International Journal of Computer and Information Sciences*, Jun 1980
– S. Peterson, "Computing constrained Delaunay triangulations", [en ligne, consulté en janvier 2015]