UMONS
Université de Mons

Faculté des Sciences


# Rapport de projet

## Rapport Final

S-INFO-820 Projet de structures de données II

Made by Roméo IBRAIMOVSKI & Maxime NABLI

Faculté
des Sciences

Supervised by Gauvain DEVILLEZ

3e Bachelier en Sciences Informatiques
Année 2021-2022

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

**Résumé**

Ce rapport d'implémentation est rendu dans le cadre de l'AA-S-INFO-820 "Projet de structure de données II", supervisé par l'Assistant Gauvain Devillez en année académique 2021-2022. Ce rapport a pour but d'expliquer et de justifier nos différents choix de conception.

# Table des matières

# Introduction

Dans le cadre des cours de "Structure de Données I" et "Structure de Données II" , nous avons vu les arbres binaires et d'autres structure de données étant des variantes de ces Arbres. Pour ce projet du cours de "Structure de Données II", nous avons du implémenter les "Binary Search Partition Tree", notés Arbres BSP pour le reste de ce rapport, structure de données modelée après les arbres binaires. Ces Arbres BSP sont utilisés en géométrie pour représenter les objets contenus dans un plan. Avec une droite, on coupe ce plan en 2 sous-plans, aussi appelés demi-espace ouvert positif ou négatif. Le demi-espace ouvert positif est le sous-plan ou lorsque les coordonnées d'un point sont remplacée dans l'équation de la droite, on a un résultat positif. Pareil pour le demi-espace ouvert négatif. et, nous répétons cette action dans chacun des sous-plan crée jusqu'à ce qu'il ne reste qu'un fragment d'objet par sous-plan. Les noeuds internes de l'arbre représentent les droites coupant le plan et les sous-plans, les feuilles sont les fragments d'objets. Pour notre projet, les objets sont des segments de droites et donc, en une dimension. Dans ce cas la, certains segments peuvent se retrouver contenu dans les droites de coupes. Si cela arrive, nous pouvons retrouver ces segments dans une liste associée au noeud de la droite.

# 1. Géométrie

Nous avons tout d'abord commencé par implémenter l'aspect géométrique de l'application.

## 1.1. Les Points (Classe Point)

Base de tous les objets géométrique dont on a besoin, un point est composé de 2 doubles x et y représentant ses coordonnées.

## 1.2. Les Segments (Classe Segment)

Les Segments sont composés de 2 points, les extrémités, et de la couleur du segment. On peut en calculer la longueur, on peut le diviser en deux, et on peut le transformer en droite.

## 1.3. Les Vecteurs (Classe Vector)

Les Vecteurs sont composé de deux doubles x et y. On peut le construire soit avec un couple de double, soit avec un couple de points.
On peut calculer leur norme, leur opposer, en additionner, en soustraire et en multiplier par des scalaires.

## 1.4. Les Droites (Classe Line)

Une droite dans le plan d'équation $\alpha\ x^2 + \beta\ x + \gamma = 0$. $\alpha$, $\beta$ et $\gamma$ étant des doubles. On peut calculer son intersection avec une autre droite, savoir si un segment se trouve à droite ou à gauche de cette droite, et avoir la liste des segments confondus à cette droite.
Pour savoir si un segment se trouve dans le demi-espace ouvert positif ou négatif, on regarde si les deux extrémités du segment s'y trouvent en replaçant les coordonnées dans l'équation de la droite.
Si les 2 extrémités ont un résultat positif, le segment est dans le demi-espace positif.
Si les deux extrémités ont un résultat négatif, le segment est dans le demi-espace négatif.
Si les deux extrémités ont un résultat nul, le segment est confondu à la droite.
Si une extrémité à un résultat positif et l'autre à un résultat négatif, alors le segment est intersecté par la droite.

# 2. Arbres BSP (Classe BSPTree)

Un arbre BSP est, dans notre cas, une structure de donnée représentant un plan contenant des segments. Chaque noeud interne est une droite coupant le plan en deux, séparant dans l'arbre droite et gauche les segments étant à droite et à gauche de la droite de coupe. Chaque feuille contient un segment.
Pour construire cet arbre, il y a plusieurs étapes.

## 2.1. Choix de la droite de coupe

On commence par choisir une droite de coupe. On la choisit via une heuristique qui sera décrite dans la section 3 ci-dessous.

## 2.2. La Distribution des Segments (Classe SegmentDistribution)

On commence par créer 2 ArrayList de segments, toutes les deux vides. Une pour les segments dans le demi-espace ouvert positif, une pour les segments dans le demi-espace ouvert négatif. Pour chaque segment de la liste, on utilise la fonction de Line permettant de savoir dans quel demi-espace ouvert il est situé par rapport à la droite de coupe et on l'ajoute à l'ArrayList correspondante. Si le segment [A,B] est intersecté par la droite, on le divise en 2 segments [A,C] et [C,B], C étant le point d'intersection entre le segment et la droite, et on ajoute les deux nouveaux segments ainsi créés à la bonne ArrayList. Si le segment est confondu à la droite, on ne l'ajoute à aucune des deux ArrayLists.

## 2.3. La Construction des sous-arbres gauche et droite

Grace aux ArrayList obtenues grâce à la Distribution des Segments, on peut dès à présent effectuer récursivement la construction des sous-arbres droite et gauche.

## 2.4. Complexité

# 3. Les Heuristiques

Pour choisir la droite de coupe utilisée pour construire notre arbre, nous devons utliser une heuristique. Dans le cadre du projet, nous avons du implémenter 3 heuristiques. Pour ce faire, nous avons utilisé les Interfaces de Java.

## 3.1. Heuristique Aléatoire

En utilisant le Random de Java, on choisit un segment aléatoire et on l'utilise comme droite de coupe.

## 3.2. Heuristique Standard

Cette heuristique prend le premier segment de la liste et l'utilise.

## 3.3. Heuristique TWNB

Cette heuristique choisit le segment qui maximise la fonction suivante :

$$F = f_{d+} \cdot f_{d-} - w \cdot f_d \tag{1}$$

$f_{d+}$ correspond au nombre de segments dans le demi-espace ouvert positif.
$f_{d-}$ au nombre de segments dans le demi-espace ouvert négatif.
$f_d$ au nombre de segments intersectés par la droite.
w a un poids à fixer.
Après uné série de tests, nous avons choisi un poids de 7.

On commence par prendre le premier segment de la liste que l'on ajoute dans une liste de segments déjà utilisés. On ajoute aussi à cette liste tous les segments se trouvant sur la droite. Ensuite, on calcule le nombre de segments dans ses demis-espaces ouverts positif et négatif via des méthodes de la Classe Line, et ensuite on calcule la liste des segments intersectés par cette droite et on effectue le calcul de la fonction notée ci-dessus et, on garde en mémoire le ratio des segments. On prend le nombre de segments à gauche que l'on divise par le nombre de segments à droite pour savoir combien de fois on a à gauche ce que l'on a a droite.

Ensuite, on prend le reste des segments dans l'ordre.
Premièrement : On va regarder si ils ne sont pas déjà dans la liste des segments utilisés. Si il l'est, on passe au segment suivant.
Deuxièmement : On calcule le ratio des segments. Si on a un ratio plus petit que le segment gardé en mémoire, on passe à l'étape suivante sinon, on ajoute le segments et tous les segments confondus à sa droite dans la liste des segments déjà utilisés et on passe au segment suivant. On veut équilibrer le nombre de segments de chaque côté.
Troisièmement : On calcule le nombre de segments intersectés par la droité crée par le

segment actuel. Si on a moins de segments intersectés, on passe à l'étape suivante sinon, on ajoute le segments et tous les segments confondus à sa droite dans la liste des segments déjà utilisés et on passe au segment suivant. On veut minimiser le nombre d'intersection.
Enfin, on calcule le résultat de la foncion. Si ce résultat est plus petit, on garde le segment, son ratio et le résultat en mémoire, on ajoute le segments et les segments confondus a la liste des segmens utilisés et on passe au segment suivant.
Sinon, on ajoutes les segments et on passe au suivant.

Dans le pire des cas, aucun segment ne se trouve sur la même droite qu'un autre, chaque segment a un ratio plus petit que le précédent et est intersecté par moins de segments que le précédent. Pour chaque segment, on parcout 2 fois la liste de segments. Une fois pour faire le ratio et une fois pour les intesection. Par segment, la complexité est en O($n^2$). Et, comme on doit parcourir la liste entièrement, on doit faire n fois l'opération, donc la complexité de cet algorithhme est en O(n) $\cdot$ O($n^2$) , c'est à dire en O($n^3$).

## 3.4. Comparaison des différentes heuristiques

Excepté pour les fichiers Random, l'heuristique aléatoire à toujours constamment un plus grand nombre de noeuds que les autres heuristiques.

Par exemple, pour rectangesSmall, un de nos essai nous a donné 500 noeuds pour l'heuristique aléatoire mais 75 et 73 pour les autres heuristiques.

On remarque aussi que, en terme de taille d'arbre, l'heuristique standard et l'heuristique TWNB sont toujours très proche.

### 3.4.1. Taille de l'Arbre

Excepté pour les fichiers Random, l'heuristique aléatoire à toujours constamment un plus grand nombre de noeuds que les autres heuristiques.

Par exemple, pour rectangesSmall, un de nos essai nous a donné 500 noeuds pour l'heuristique aléatoire mais 75 et 73 pour les autres heuristiques.

On remarque aussi que, en terme de taille d'arbre, l'heuristique standard et l'heuristique TWNB sont toujours très proche.

### 3.4.2. Hauteur de l'Arbre

Ici, l'heuristique aléatoire est celle donnant les arbres avec la plus petite hauteur. Pour le fichier randomHuge, on a eu lors d'un de nos tests une hauteur de 49 pour l'aléatoire et 323 pour la standard et la TWNB. Et, comme précédemment, l'heuristique Standard et l'heuristique TWNB ont des hauteurs similaires.

Par exemple, pour le fichier rectangleHuge, Standard nous a donné lors d'un essai une hauteur de 50 et TWNB donne une hauteur de 49.

#### 3.4.3. Temps CPU pour construire un arbre BSP

Dans tous les cas, l'heuristique aléatoire est celle qui construit les arbres le plus rapidement.

Pour le fichier randomHuge de 47000 segments (fichier choisi pour présenter l'exemple de cette sous-section ainsi que la suivante car fichier le plus volumineux) , on a pour l'heuristique aléatoire constamment une construction en une seconde ou moins. Parfois même en dessous d'une demi-seconde.

Pour l'heuristique Standard, on a en moyenne une construction de l'arbre en 8 secondes et pour l'heuristique TWNB on a une moyenne de 17 secondes.

#### 3.4.4. Temps CPU pour effectuer l'Algorithme du Peintre

Comme pour la construction de l'arbre, c'est l'heuristique aléatoire qui produit l'arbre sur lequel l'algorithme du peintre est le plus rapide.

Pour le fichier randomHuge , on a constemment une centaine de millisecondes pour l'heuristique aléatoire.

Pour l'heuristique standard, l'algorithme est effectué en moyenne de 1.5 secondes et, pour l'heuristique TWNB on a une moyenne de 1 seconde.

#### 3.4.5. Conclusion de la comparaison

- L'heuristique aléatoire crée des arbres contenant beaucoup de noeuds mais une petite hauteur.

- L'heuristique Standard et l'heuristique TWNB créent des arbres contenant un nombre similaire de noeuds et ayant une hauteur semblable.

- L'echelle de grandeur pour le temps de création de l'arbre est Aléatoire => Standard => TWNB

- Pour l'algorithme du peintre est Aléatoire => TWNB => Standard.

Avec, dans les deux cas, une plus grosse différence entre les heuristiques plus le fichier est grand.

# 4. L'Algorithme du Peintre

# 5. Applications

## 5.1. Lecture des fichiers texte représentant une scène (Classe SceneReader

Notre SceneReader est un objet prenant en paramètre un fichier et le parcours deux fois. D'abord pour vérifier qu'il soit au bon format et ensuite pour le lire.

On commence par regarder si la première ligne contient bien 3 entiers, le premier représentant la limite de l'axe des abscisses, le deuxième représentant la limite de l'axe des ordonnées et le troisième représentant le nombre de segments contenus dans le fichier.

Ensuite, on parcourtle reste du fichier. A chaque ligne, on vérifier que les 4 premiers éléments dont bien des doubles, ensuite que l'on a bien une couleur valide de l'application. Et pour terminer, on compte chaque ligne pour vérifier que l'on a bien le bon nombre de segments annoncé.
Si le fichier n'existe pas ou n'a pas le bon format, la taille de sa liste de segments, la limite en X et la limite en Y sont à 0.
C'est cela que nous utiliseront par la suite dans le mode console et interactif pour "fermer" l'application.

Ensuite, pour la lecture du fichier, on stocke les 3 entiers de la première ligne et, à chaque ligne, on crée le segment correspondant avant de l'ajouter a la liste des segments du fichier.

## 5.2. Mode Console (Classe TestConsole)

### 5.2.1. Implémentation

On a commencé par imprimer un message de bienvenue dans la console et proposer un choix du type de scene à écrire dans l'invite de commande.

On récupère le choix de l'utilisateur avec un Scanner. Si l'utilisateur entre autre chose que les choix proposés, un fichier par défaut sera utilisé : le fichier randomHuge.

En utilisant un switch sur le choix fait par l'utilisateur, on arrive à l'étape suivante : le choix de la taille de la scene. Comme précedemment, on récupère le choix de l'utilisateur via un scanner et utilisons un switch pour récupérer le path vers le fichier choisi. Pour les fichiers fournit par les enseignants, le path est déjà prêt dans le code. Pour un fichier ajouté manuellement, il y a d'abord la vérification de la conformité et ensuite, si il ne l'est pas, l'application est quittée.

Pour terminer, on imprime quelques informations sur la scene choisie : Son nom, son nombre de segments et la limite de ses axes avant de construire les arbres à partir des différentes heuristiques.

Pour chacune d'entre elle, on construit l'arbre et récupérons le temps CPU necessaire pour sa construction grace à la classe Instant de Java, ensuite nous faisons la même chose en effectuant l'algorithme du peintre.

Lorsque toutes les heuristiques ont été faites et que toutes les informations sont regroupées dans un tableau dans la console, on peut soit relancer le même fichier, soit relancer avec un autre fichier, soit quitter l'application. Si l'utilisateur entre autre chose, l'application se fermera.

### 5.2.2. Guide d'utilisation

- Lancer TestConsole.

- Choisir un type de fichier parmi ceux proposés , ou entrer le path vers un fichier externe ajouté via l'invite de commande.

- Si un type de fichier à été choisi, choisir sa taille via l'invite de commande.

- Recommencer ou non.

### 5.3. Mode Graphique (Classe TestInteractive)

#### 5.3.1. Implémentation

#### 5.3.2. Guide d'utilisation

## Conclusion