Art Abstrait et Arbres
Projet d'ASD3

A réaliser en binôme

Mathieu Vavrille Evgeny Gurevsky Irena Rusu

Novembre 2022

1 Introduction

Piet Mondrian est un peintre du début du 20ème siècle, pionnier de l'art abstrait. Son oeuvre est caractérisée par des oeuvres très géométriques, comme celle présentée en Figure 1[1].

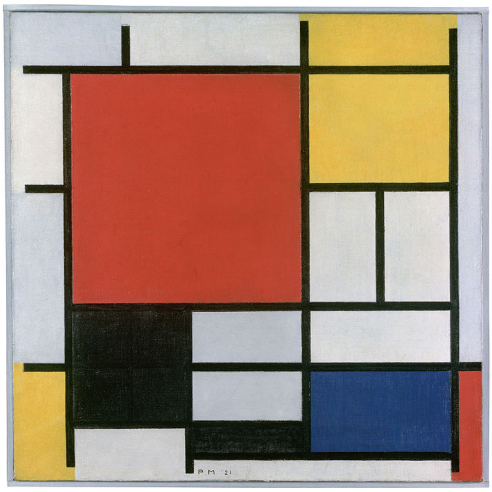
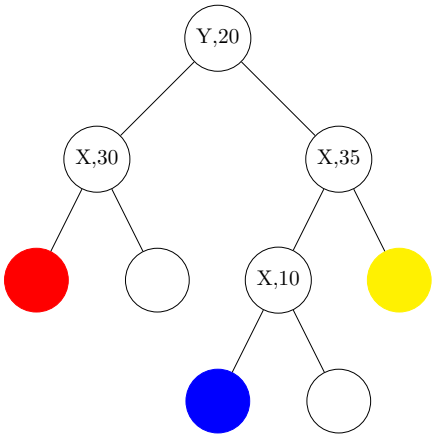


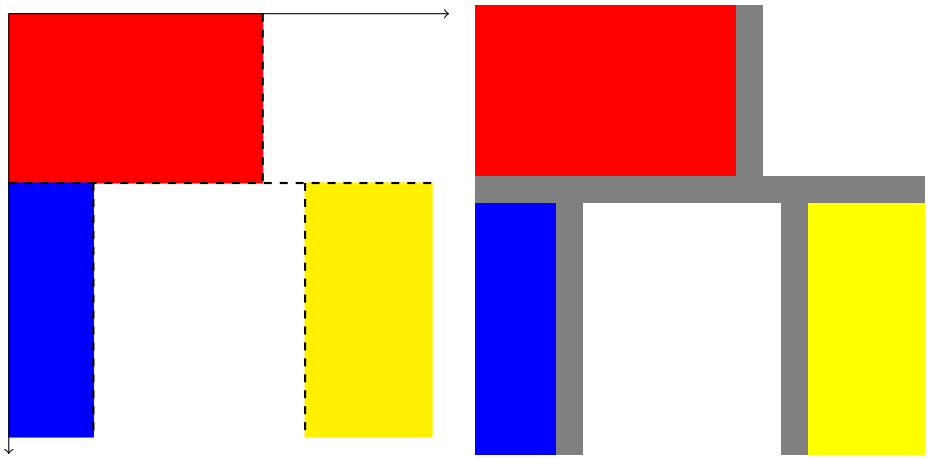
Figure 1: Composition avec grand plan rouge, jaune, noir, gris et bleu

Le but de ce projet est d'utiliser les structures vues en cours pour générer des images à la manière des tableaux de Mondrian. À la fin du projet vous proposerez un outil pour générer des peintures de manière aléatoire, avec plusieurs

[1] `https://fr.wikipedia.org/wiki/Composition_avec_grand_plan_rouge,_jaune,_noir,_gris_et_bleu`



(a) Arbre



(b) Représentation (c) Tableau généré

méthodes de génération, une méthode pour convertir un tableau en représentation textuelle et vice-versa, et un rapport qui montre ce que vous arrivez à générer avec votre travail.

2 Projet

2.1 Conversion arbre→peinture

Les arbres que vous allez implémenter et utiliser dans ce projet sont une variante des 2d-arbres vus en cours. Dans votre cours les 2d-arbres alternaient une division sur les X, et une division sur les Y, et contenaient des points pour connaître les coordonnées de la division. Ici, l'arbre doit permettre de :

- faire une découpe selon l'axe des Y au noeud racine;
- faire plusieurs découpes selon le même axe d'affilée;
- représenter des couleurs dans les noeuds.

Un exemple d'arbre à représenter est donné en Figure 2a. Cet arbre a une représentation, à la manière de celles que vous avez vues en cours, donnée en Figure 2b. À partir de cet arbre un tableau peut être généré. Des lignes doivent être ajoutées, avec une largeur `largeurLigne` (en nombre de pixels), pour finalement obtenir une image, comme celle en Figure 2c.

2.2 Génération aléatoire d'arbres

Pour générer aléatoirement un arbre (donc un tableau), on se donne une limite `nbFeuilles`, à ne pas dépasser, sur le nombre de feuilles de l'arbre final. On part d'un arbre initial qui contient uniquement une racine/feuille blanche (donc un tableau blanc), et on le transforme en plusieurs étapes. À chaque étape, une feuille de l'arbre courant est sélectionnée pour être divisée en deux, jusqu'à ce que le nombre maximum `nbFeuilles` de feuilles soit atteint, ou qu'on ne puisse plus diviser (voir la Section 2.2.1). Lors de la création d'une feuille, on lui assigne la couleur de son père avec une certaine probabilité, sinon une couleur choisie aléatoirement parmi l'ensemble de couleurs possible. Le tableau représenté par l'arbre est ensuite dessiné. Les détails sont fournis ci-dessous.

2.2.1 Choix de la feuille à diviser

On associe à chaque feuille un poids défini comme suit. Soit F une feuille de l'arbre, notons w et h la largeur et la hauteur de la région représentée par la feuille. Le poids de la feuille est défini comme

$$\omega(F) = \frac{wh}{(w+h)^{1.5}}$$

La feuille choisie pour être divisée est la feuille de plus grand poids parmi les feuilles dont les dimensions sont encore assez grandes pour être divisées. L'idée de ce poids est d'essayer de diviser prioritairement les feuilles carrées. En revanche il ne faut pas diviser les régions trop petites, au risque de se retrouver avec uniquement des traits. Pour s'en empêcher, on se donne une valeur `minDimensionCoupe`, et – si la taille dans au moins une des deux dimensions de la région est plus petite que `minDimensionCoupe` – on ne peut pas diviser la feuille. Si, à la fin, toutes les régions sont trop petites et que le nombre de feuilles voulu `nbFeuilles` n'est pas atteint, ce n'est pas grave, la procédure s'arrête sans avoir atteint le nombre de feuilles.

2.2.2 Choix de l'axe/coordonnée de la division

Quand une feuille est choisie pour être divisée, il faut choisir l'axe de la division, ainsi que la coordonnée. L'axe est choisi en fonction de la forme de la région à diviser. On va privilégier de diviser selon le grand axe, plutôt que selon le petit. Soient w et h la largeur et la hauteur de la feuille choisie. L'axe choisi est l'axe des X avec probabilité $\frac{w}{w+h}$, et l'axe des Y sinon (donc avec probabilité $\frac{h}{w+h}$).

Pour la coordonnée, il ne faut pas que la découpe se fasse trop près d'un des deux bords. On se donne donc un paramètre `proportionCoupe` entre 0 et 1 qui détermine la région où l'on s'autorise à découper. Soit s la taille de la dimension découpée, la découpe se fera selon un axe choisi aléatoirement entre $\lfloor s \cdot \texttt{proportionCoupe} \rfloor$ et $\lceil s \cdot (1 - \texttt{proportionCoupe}) \rceil$. Par exemple, si `proportionCoupe` vaut 0.1, sur une découpe verticale, on s'interdit de découper dans la partie 10% à gauche ou 10% à droite.

2.2.3 Choix de la couleur

Pour le choix de la couleur, un paramètre `memeCouleurProb` est fixé. Chaque nouvelle feuille va prendre la couleur de son parent avec une probabilité de `memeCouleurProb`, ou alors une couleur choisie aléatoirement parmi un ensemble fixé de couleurs (rouge, bleu, jaune, noir, blanc) avec probabilité $(1 -$ `memeCouleurProb`$)/5$ pour chacune des cinq couleurs.

2.2.4 Un mot sur l'aléatoire

Les algorithmes aléatoires doivent être reproductibles[2]. Pour cela, les générateurs aléatoires (tels que `java.util.Random`) proposent de fixer une graine aléatoire, `seed`, en fonction de laquelle sont construits de manière déterministe les nombres aléatoires produits par le générateur. Il faut que votre programme, si appelé deux fois avec les mêmes paramètres et la même graine aléatoire, génère le même arbre.

2.2.5 Amélioration de la stratégie

Pour le projet, vous devrez implémenter dans un premier temps la méthode définie ci-dessus. Cette méthode a plusieurs défauts artistiques[3]. Vous devez proposer une nouvelle méthode, et l'implémenter pour corriger un des défauts. Dans votre rapport vous montrerez avec images à l'appui que votre stratégie corrige un défaut, et vous mettrez en avant les défauts de votre nouvelle approche. Votre nouvelle stratégie ne peut pas se contenter d'une petite modification de l'approche présentée ci-dessus (par exemple changer la fonction de poids ou les couleurs n'est pas suffisant), mais doit être une nouvelle idée.

[2] Vérifiez votre programme pour que cela soit bien le cas.

[3] Par exemple, vous pourrez constater que certains types de tableaux ne peuvent pas être construits avec la méthode fournie.

2.2.6 Résumé

Votre programme doit permettre à l'utilisateur·ice de sélectionner les paramètres suivants :

- `strategy`, la stratégie de génération d'arbre, celle de l'énoncé ou celle que vous avez créé. Les paramètres suivants concernent la stratégie de l'énoncé. Si votre stratégie nécessite des paramètres, il doit être possible pour l'utilisateur·ice de les spécifier
- `largeur`, `hauteur`, la hauteur et largeur (en nombre de pixels) de l'image générée
- `nbFeuilles`, le nombre de feuilles maximum de l'arbre
- `minDimensionCoupe`, la taille minimum des dimensions d'une région pour pouvoir la diviser
- `proportionCoupe`, la valeur qui permet de ne pas découper trop proche des bords de la région
- `memeCouleurProb`, la probabilité de garder la couleur du parent lors d'une division.
- `largeurLigne`, la largeur (en pixels) de la ligne qui sépare les régions
- `seed`, la graine aléatoire utilisée pour générer l'arbre.

Votre code doit absolument contenir (avec ce nom) les fonctions :

- `chooseLeaf` (ou `choisirFeuille`) qui choisit une feuille à diviser
- `chooseDivision` (ou `choisirDivision`) qui choisit l'axe et la coordonnée de la division d'une feuille donnée
- `chooseColor` (ou `choisirCouleur`) qui choisit la couleur d'une feuille nouvellement créée
- `generateRandomTree` (ou `genereArbreAleatoire`) qui génère un arbre aléatoire selon la procédure vue plus haut
- `generateBetterRandomTree` (ou `genereMeilleurArbreAleatoire`) qui génère un arbre aléatoire selon la procédure que vous avez créée
- `toImage` qui à partir d'un arbre, génère l'image au format PNG du tableau.

Vous choisirez également des noms clairs pour vost structures de données, par exemple `Quadtree`, et/ou `ABR`, et/ou `AVL`, etc.

Important. Votre programme doit implémenter efficacement la procédure indiquée, en réalisant une *complexité au pire* minimum pour l'ensemble du traitement. Pour cela, à chaque étape de la procédure (et surtout dans le choix de la feuille) vous devez choisir et implémenter avec soin vos structures de données.

3 Rendu de Projet

Pour vous aider avec la gestion des images en Java, nous vous fournissons une classe `Image.java` avec laquelle vous pouvez créer une image, colorier des pixels ou des régions, et sauvegarder au format png. Les programmes doivent être implémentés par vous mêmes en Java 1.7[4] et doivent fonctionner parfaitement sur les machines du CIE, sous Linux. Le programme sera lancé en ligne de commande et pourra être utilisé indépendemment d'Eclipse ou autre IDE. L'appel à des structures de données optimisées existantes (du genre `TreeSet` ou `HashMap`, `HashSet` etc.) est interdit : vous devez implémenter vous-mêmes vos structures de données[5]. Seuls les tableaux et les listes chaînées peuvent être utilisés tels que fournis par Java. Aucun appel à une méthode mise à disposition dans les bibliothèques Java (par exemple un tri d'une liste) n'est autorisé si vous ne connaissez pas le fonctionnement de la méthode et sa complexité (ce que vous montrerez dans les calculs de complexité demandés ci-dessous).

Un rapport d'au maximum 12 pages sera fourni, incluant :

- les commandes de compilation et exécution en mode console
- une vue globale de votre programme, sous la forme d'un algorithme (c'est-à-dire en pseudo-code), dont les instructions sont des appels à des procédures/fonctions ; la spécification et les paramètres de chaque procédure/fonction seront précisément décrits
- pour chaque méthode parmi celles de la liste demandée :
  - le détail de la procédure/fonction, sous forme algorithmique (c'est-à-dire en pseudo-code) ;
  - l'explication de son fonctionnement ;
  - sa complexité.
- Des exemples de résultats obtenus avec votre programme.

Important

- Les fichiers du programme, ainsi que les jeux de données, seront mis dans un répertoire appelé `Nom1Nom2` (où `Nom1` et `Nom2` sont les noms des deux étudiant·e·s du binôme). Ce répertoire sera ensuite archivé sous le nom `Nom1Nom2.zip`. L'archive sera déposée sur Madoc, au plus tard le **vendredi 2 décembre à 23h55** (Madoc n'acceptera pas de retard).

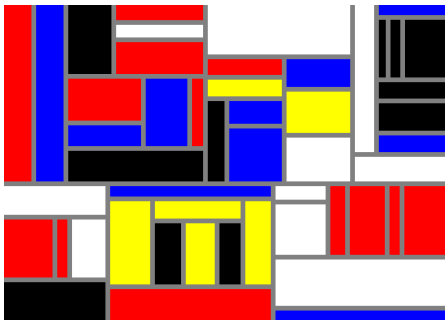
[4] Afin d'éviter l'utilisation de fonctionnalités plus récentes qui permettent d'écrire du code dont la complexité serait insuffisamment maîtrisée.

[5] Toute ressemblance - même mineure - entre deux programmes, ou entre un programme et du code sur Internet, sera considérée comme non-fortuite et entraînera automatiquement la note de 0 sur la partie concernée ainsi que sur toute autre partie utilisant, même très peu, la partie concernée. Une pénalité supplémentaire de 5 points sera également appliquée. Vous devez prendre des mesures pour vous assurer que votre code ne ressemble à aucun autre et n'est pas partagé, avec ou sans votre accord.

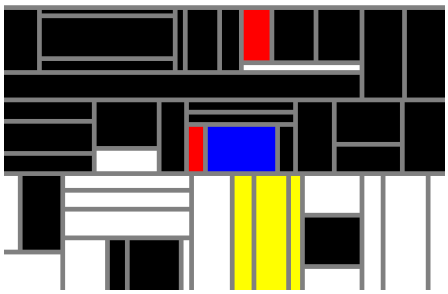
- La dernière séance est consacrée à une démo de 10 minutes par projet, pour laquelle des paramètres vous seront fournis en entrée. Vous devez donc impérativement suivre les consignes concernant le format du fichier en entrée. Cette dernière séance **n'est donc pas une séance de travail sur le projet**.
- Tout est important pour la notation. En particulier, il sera accordé beaucoup d'attention :
  - au respect des consignes
  - à la recherche d'une complexité minimum via la mise en place des structures de données les plus adaptées
  - à la clarté du code et du rapport.

A Exemples de résultats

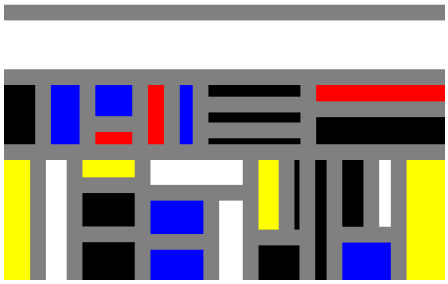
La figure 3 montre des résultats obtenus par la méthode expliquée dans le sujet. Pour chaque image, les paramètres sont donnés dans l'ordre `(nbFeuilles, proportionCoupe, minDimensionCoupe, memeCouleurProb, largeurLigne)`. Ces exemples sont là pour vous montrer l'impact des paramètres, ainsi que pour vous donner des idées d'améliorations à faire pour votre approche (pour commencer à y réfléchir rapidement).



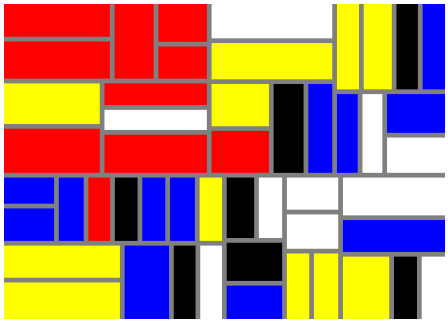
(a) $(50, 0.1, 50, 0.5, 15)$



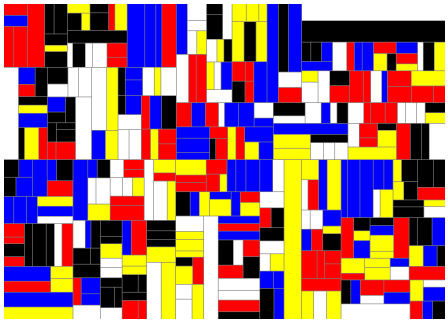
(b) $(50, 0.1, 50, 0.9, 15)$



(c) $(30, 0.3, 50, 0.5, 50)$



(d) $(50, 0.45, 50, 0.5, 15)$



(e) $(400, 0.3, 5, 0.5, 2)$

(f) $(400, 0.01, 200, 0.5, 2)$

Figure 3: `largeur`, `hauteur` $= (1400, 1000)$