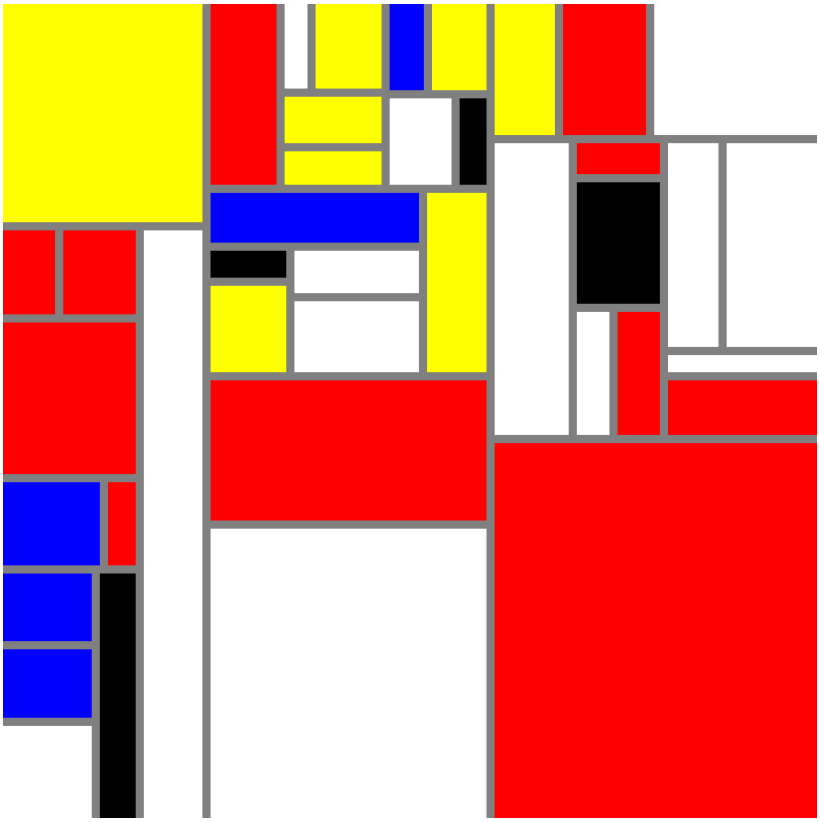
JOUSSELIN Athanaël
STALDER Emmanuel
584

ASD3 – Compte-rendu de projet



Responsable de module : RUSU-ROBINI Irena
Chargé de TD : VAVRILLE Mathieu
Encadrant de TP : RUSU-ROBINI Irena

Licence Informatique
Année universitaire 2022-2023

Sommaire

Rappel du sujet :

L'objectif de ce projet était de parvenir à produire des toiles semblables aux œuvres du peintre abstrait Piet Mondrian *via* l'utilisation d'arbres 2D et d'AVL en JAVA.

Partie I – Exécution du programme

1. Commandes d'exécution

Pour exécuter le code fourni, les commandes suivantes doivent êtres entrées dans le terminal de commande :

```
javac -g Main.java
java Main l h nbFeuilles minDimensionCoupe proportionCoupe memeCouleurProb largeurLigne
```

Afin d'avoir accès à plus de paramètres, la commande suivante peut être utilisée :

```
java Main l h nbFeuilles minDimensionCoupe proportionCoupe meemCouleurProb largeurLigne
seed strat2 stratGrosRect frequence name
```

Les spécifications des paramètres sont les suivantes :
- l ∈ $\mathbb{N}$*, la largeur de la toile souhaitée
- h ∈ $\mathbb{N}$*, la hauteur de la toile souhaitée
- nbFeuilles ∈ $\mathbb{N}$*, le nombre de feuilles maximum sur la toile
- minDimensionCoupe ∈ $\mathbb{N}$*, la taille minimum (en x ou en y) d'une feuille
- proportionCoupe ∈ [0, 0.5[, intervalle sur lequel découper lors du choix de découpe
- memeCouleurProb ∈ [0, 1], la probabilité qu'une feuille ait la même couleur que son parent
- largeurLigne ∈ [0, min(⌊l/2⌋, ⌊h/2⌋)[, l'épaisseur des lignes grises
- (optionel) seed ∈ $\mathbb{R}$ | 'n', la graine aléatoire ou son absence
- (optionel) Strat2 (y/n), si on utilise la 2^{e} stratégie ou non
- (optionel) StratGrosrect (y/n), si on souhaite avoir des rectangles plus gros à intervalles réguliers
- (optionel) frequence ∈ $\mathbb{N}$* | 'n', la fréquence d'apparition des gros rectangles, 10 par défaut
- (optionel) name, le nom de l'image générée

Si le choix est fait d'utiliser la commande étendue, tous les champs de paramètres doivent être renseignés.

2. Fonctionnement du programme

Variables :
  Tableau de chaines de caractères args
Début
  entier larg ← 0
  entier haut ← 0
  entier nbFeuilles ← 0
  entier minDimCoupe ← 0

```
    reel propCoupe ← 0
    reel memeCouleurProb ← 0
    entier largeurLigne ← 0
    long entier seed ← 0
    boolean seedUsed ← false
    booleen strat2 ← false
    boolean stratGR ← false
    entier freq ← -1
    chaîne nom ← «sans titre»

    Si (args.length ≥ 7) Alors
        larg ← parseInt(args[0])
        haut ← parseInt(args[1])
        nbFeuilles ← parseInt(args[2])
        minDimCoupe ← parseInt(args[3])
        propCoupe ← parseDouble(args[4])
        memeCouleurProb ← parseDouble(args[5])
        largLigne ← parseInt(args[6])
        Random seedGenerator ← new Random()
        seed ← seedGenerator.nextInt()
        Si ( args.length > 7) Alors
            seed ← parseLong(args[7])
            seedUsed ← true
            Si (args[8] = «y») Alors
                strat2 ← true
            Fin Si
            Si (args[9] = «y») Alors
                stratGR ← true
            Fin Si
            freq ← parseInt(args[10])
            nom ← args[11]
        Fin Si
    Si (seedUsed) Alors
        Param p ← new Param(larg, haut, nbFeuilles, minDimCoupe, propCoupe, memeCouleurProb,
largLigne, seed, stratGR)
    Sinon
        Param p ← new Param(larg, haut, nbFeuilles, minDimCoupe, propCoupe, memeCouleurProb,
largLigne)

    Fin Si
    strategy s ← new strategy(p)
    s.genererArbreAleatoire()
    Si (strat2 = true) Alors
        s.creerImageS2(nom)
    Sinon
        s.creerImage(nom)
    Fin Si
Fin
```

Partie II - Complexités

1. *choisirFeuille()*

```
choisirFeuille(AVL tab) : arbre2d
Début
   arbre2d stock ← tab.max();
   tab ← tab.supp(stock);
   retourner stock;
Fin
```

Complexité : O(log n) où n est le nombre de nœuds de l'AVL.

L'AVL est rangé par rapport au poids des feuilles, qui est calculé au moment de leur créations. Ainsi, on récupère la feuille la plus lourde et on la retire de l'AVL.

2. *choisirDivision()*

```
choisirDivision (listeDesParam p, entier minX, entier maxX, entier minY, entier maxY, arbre2d a) :
arbre2d
Début
   Si (maxX-minX < p.MinDimensionCoupe) ou (maxY-minY < p.MinDimensionCoupe) Alors
      // on arrète de découper notre arbre et on a notre toile prête
   Sinon
      réel random ← nouveau random()
      Si (random ≤ ((maxX-minX) / (maxX-minX+maxY-minY))) Alors
         decoupe en X
         random ← nouveau random()
         réel floor ← ⌊(maxX-minX)*p.ProportionCoupe⌋
         réel ceil ← ⌈(maxX-minX)*(1-p.ProportionCoupe)⌉
         entier val ← random*(ceil-floor)+floor+minX
      Sinon
         decoupe en Y
         random ← nouveau random()
         réel floor ← ⌊(maxY-minY)*p.ProportionCoupe⌋
         réel ceil ← ⌈(maxY-minY)*(1-p.ProportionCoupe)⌉
         entier val ← random*(ceil-floor)+floor+minY
      Fin Si
   Fin Si
   a.decoupe ← decoupe
   a.val ← val
   retourner a
Fin
```

Complexité : Θ(1)

On choisit un nombre aléatoire entre 0 et 1 et on détermine si on coupe en X ou en Y. On choisit ensuite un second nombre aléatoire pour trouver où découper, puis on change la feuille en conséquence.

3. *choisirCouleur()*

choisirCouleur (couleur précédenteCouleur, listeDesParam p) : couleur
Début
  random ← nouveau random()
  réel probMeme ← p.MemeCouleurProb
  Si (random ≤ probMeme) **Alors**
    retourner précédenteCouleur
  Sinon Si (random ≤ probMeme+((1-probMeme)/5)) **Alors**
    retourner rouge
  Sinon Si (random ≤ probMeme+((1-probMeme)/5)*2) **Alors**
    retourner bleu
  Sinon Si (random ≤ probMeme+((1-probMeme)/5)*3) **Alors**
    retourner jaune
  Sinon Si (random ≤ probMeme+((1-probMeme)/5)*4) **Alors**
    retourner noir
  Sinon
    retourner blanc
  Fin Si
Fin

Complexité : Θ(1)

On choisit un nombre aléatoire entre 0 et 1 et on retourne la couleur obtenue.

4. *genererArbreAleatoire()*

genereArbreAleatoire et **genereMeilleurArbreAleatoire** (listeDesParam p) : Arbre2d, AVL
Début
  AVL tab ← nouveau AVL
  Arbre2d a ← nouveau Arbe2d(blanc)
  tab ← tab.add(a)
  Pour i **Allant** de 1 **À** p.NbFeuilles **Faire**
    Si (i mod p.frequence = 0 et p.StratGrosRect = vrai) **Alors**
      stock ← choisirFeuille(tab)
      stock.poids ← 0
      tab ← tab.add(stock)
    Sinon
      stock ← choisirFeuille(tab)
      a ← choisirDivision(p, a)
      a.gauche ← nouveau Arbre2d(choisirCouleur(a.couleur))
      a.droite ← nouveau Arbre2d(choisirCouleur(a.couleur))
      tab.add(a.gauche)
      tab.add(a.droite)
    Fin Si
  Fin Pour
  retrouner a, tab
Fin

Complexité : O (n log(m)), avec n = nbFeuilles et m le nombre de nœuds de l'AVL

On prend la plus lourde feuille et on la découpe jusqu'à ce que l'on ait découpé le nombre de feuilles demandé. Lors de l'application de la stratégie dite des « gros rectangles », on saute des découpes en fonction de la fréquence indiquée en paramètre.

5. *creerImage()*

a. Sous-fonction colorierBranche()

```
colorierBranche (arbre2d a, image img) : void
Début
   Si (a!=feuille et a.val>0) Alors
      img.setRectangle(gris)
      colorierBranche(a.gauche, img)
      colorierBranche(a.droite, img)
   Fin Si
Fin
```

Complexité : Θ(n)

b. creerImage()

```
creerImage (arbre2d a, AVL tab) : void
Début
   image img=nouvelle image
   img.setRectangle(magenta)
   Faire
      stock=tab.max()
      img.setRectangle(a.couleur)
      tab=tab.supp(stock)
   Tant Que tab ≠ vide
   colorierBranche(a, img)
   img.save()
Fin
```

Complexité : O(n log(m)), où n est le nombre de nœuds de l'Arbre2d et m celui de l'AVL.

On commence par colorier la toile en magenta (pour voir les éventuelles erreurs), puis on peint tous les rectangles. Enfin, on trace les marges en gris.

Partie 3 – Galerie

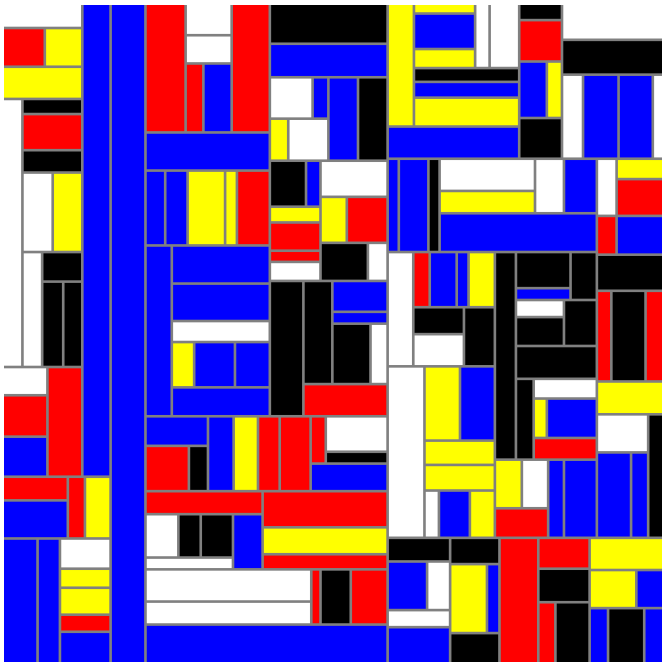


Figure 1 (1000, 1000, 200, 10, 0.2, 0.0, 2)

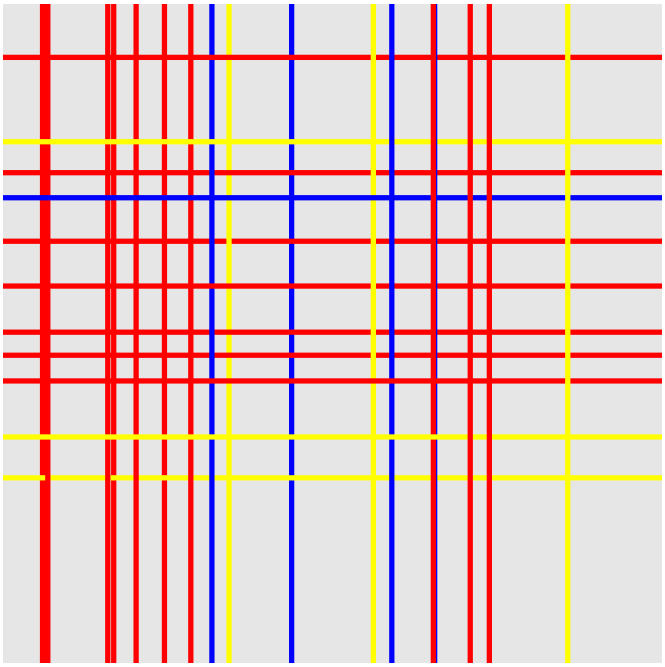


Figure 2 (1000, 1000, 50, 10, 0.2, 0.2, 4, 5892564, n, y, 5, untilted)

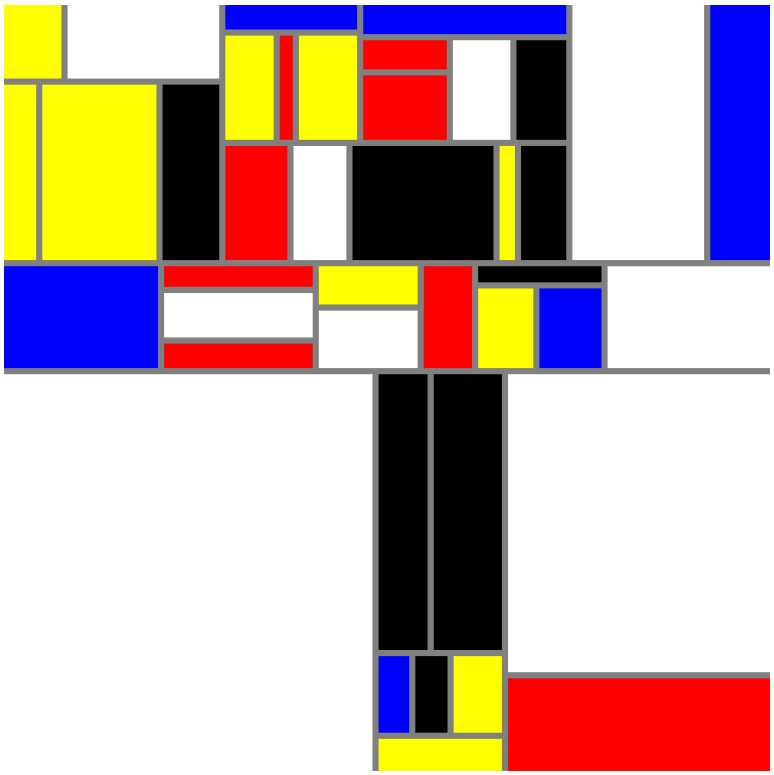


Figure 3 (1000, 1000, 50, 10, 0.2, 0.2, 4, 5892564, n, y, 5, untilted)