

Projet 1. Comment dessiner un arbre (10+2 points)

1.1 Introduction

Ce devoir vous demande de développer une application pour dessin, basée sur un système formel utilisé dans la modélisation de structures biologiques (ou auto-similaires en général). L'implémentation par défaut est en Java (travail estimé : autour de 600 lignes). Si vous ne connaissez pas Java (car vous avez équivalence pour IFT1025 ou vous êtes dans la Maîtrise), vous avez le droit de soumettre une implémentation principalement en PostScript (expliquez vos raisons dans la soumission).

L'implémentation Java vise la programmation avancée avec


- ★ développement de code (en multiples langages) selon une spécification complexe, usage de bibliothèques d'opportunité
- ★ manipulation de types agrégés : entrée/sortie, dictionnaires, chaînes, objets géométriques
- ★ types paramétriques, itérateurs, récursion, encapsulation, classes internes.

Modalités

Le travail est conçu pour des équipes de deux : une des compétences importantes est la coordination de travail de programmation. Postez sur le Forum dans StudiUM pour former des équipes, et si vous acceptez un.e binôme au hasard. Ceux et celles qui sont d'accord, seront appariés au hasard. Ceci dit, il est possible de compléter le travail seul.e pour une pénalité modeste.

Évaluation

Soumettez un seul fichier JAR exécutable qui contient aussi les sources, et un rapport (en texte ou PDF) court (1–2 pages) qui nomme les membres de l'équipe, décrit l'organisation de travail, et explique votre implémentation.

- (i) (1 point) organisation de travail d'équipe : documentez (a) l'organisation de la collaboration (p.e. rencontres, *pair programming*) ; (b) vos méthodes de communication et de gestion de code entre membres ; (c) la partition des tâches (l'équité sera prise en compte dans l'évaluation) ; et (d) les conflits (s'il y en a) et leurs résolution.
-  vous perdez cet 1 point si vous travaillez seul.e (car aucune coordination de travail) ; sauf si (a) vous avez une accommodation pour travail en équipe, ou (b) l'autre membre d'équipe a abandonné le travail, ou (c) la formation d'équipe était impossible (à la discrétion du prof, p.e. nombre impair d'étudiants à appairer). Déclarez votre raison dans la soumission.
- (ii) (9 points) correction, qualité et efficacité du code développé (entre autres, il faut code bien structuré et documenté dans la source, avec la séparation claire des responsabilités des classes, et l'utilisation des types abstraits et structures adéquates)
- (iii) (0 points mais un grand merci) SVP, rapportez combien de temps vous avez passé à travailler sur ce devoir : les heures de codage (écrire, tester, déboguer) et les heures de travail au total.

1.2 Botanique algorithmique

L-systèmes

On veut implémenter un programme qui dessine des graphiques aléatoires à l'aide d'un **système de Lindenmayer**¹ ou **L-système**². Entre autres, on veut produire des dessins qui ressemblent à des plantes (application possible : rendement graphique de paysage). Le L-système a été inventé pour ce but : il permet de modéliser le développement de structures végétales. Il s'agit d'une grammaire formelle qui définit la génération de chaînes de symboles sur un alphabet. Les symboles correspondent à des unités structurales de la plante (branches), et la chaîne résultante définit la structure à dessiner.

1. P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 2004. <http://algorithmicbotany.org/papers/abop/abop.pdf>
2. W₍₆₎:L-système

Les règles et l'inférence de chaînes. On spécifie le système pour un alphabet fixé par la chaîne de départ α (**axiome**), et un ensemble de règles de réécriture dans la forme $\langle \text{symbole} \rangle \rightarrow \langle \text{chaîne de symboles} \rangle$, spécifiant qu'une application de cette règle remplace le symbole à la gauche de la flèche par la chaîne à la droite. Exemple sur l'alphabet $\{F, -\}$ avec une seule règle :

$$\alpha = F \quad \text{avec règle} \quad F \rightarrow FF-F \quad (1.1)$$

Le L-système engendre la suite des chaînes $s_0 = \alpha, s_1, s_2, \dots$ en appliquant les règles de remplacement simultanément à tous les symboles de S_i . S'il n'y a pas de règle à appliquer, alors on garde le symbole tel quel (ce sont des symboles *terminaux*). On commence par l'axiome $s_0 = \alpha$. On dénote l'**inférence** (ou **dérivation**) par le taquet \vdash . Avec le système (1.1) par exemple, on a l'inférence

$$\begin{array}{ll} \alpha = F & \{ = s_0 \} \\ \vdash FF-F & \{ = s_1 \} \\ \vdash \underbrace{FF-F}_{1^{\text{er}} F} \underbrace{FF-F}_{2^{\text{e}} F} - \underbrace{FF-F}_{3^{\text{e}} F} & \{ = s_2 \} \end{array}$$

Multiplés règles : inférence aléatoire. S'il existe plusieurs règles avec le même côté gauche, on choisit une des règles applicables au hasard (à l'uniforme). Cela permet de générer une multitude de structures différentes avec un petit ensemble de règles. Avec l'axiome $\alpha = F$ et règles $\{F \rightarrow F-F, F \rightarrow +F\}$, on a l'inférence au hasard

$$\begin{array}{ll} \alpha = F & \\ \vdash F-F & \text{avec probabilité } 1/2 \\ \vdash +F-F-F & \text{avec probabilité } 1/4 = 1/2 \times 1/2 \end{array}$$

1.3 Graphisme tortue

On interprète la chaîne dérivée par l'application des règles comme une suite d'instructions pour graphisme tortue. La tortue, travaillant dans un système de coordonnées cartésiennes (avec axis Y orienté vers le haut et X vers la droite), possède un crayon pour dessiner en suivant une des instructions simples. L'**état de la tortue** est le triple (x, y, θ) avec sa position (x, y) et l'angle θ de son nez par rapport à la ligne horizontale (0° vers la droite, 90° vers le haut). La tortue possède aussi un mémoire organisé en pile ce qui permet à retourner à un état visité.

draw se déplacer en avant par la distance unitaire (d) en traçant une ligne.

L'état de la tortue change de (x, y, θ) à $(x + d \cos \theta, y + d \sin \theta, \theta)$.

move se déplacer en avant par la distance unitaire (d) sans tracer. L'état de la tortue change de (x, y, θ) à $(x + d \cos \theta, y + d \sin \theta, \theta)$.

stay ne rien faire. L'état ne change pas.

turnL se tourner contre le sens d'aiguille par l'angle unitaire (δ). L'état de la tortue change de (x, y, θ) à $(x, y, \theta + \delta)$.

turnR se tourner vers le sens d'aiguille par l'angle unitaire (δ). L'état de la tortue change de (x, y, θ) à $(x, y, \theta - \delta)$.

push sauvegarder (empiler) l'état courant (position et angle) de la tortue.

L'état ne change pas.

pop dépiler l'état le plus récemment sauvegardé. L'état change à (x', y', θ') ce qui était l'état au plus récent push.

TABLE 1: Jeu d'instruction pour la tortue

Table 1 montre le jeu d'instructions pour contrôler la tortue. Afin de dessiner une chaîne inférée par le L-système, les symboles sont interprétés comme des actions de la tortue : voir Table 2 pour l'encodage standard. Le dessin actuel se réalise en spécifiant l'état au début (par défaut, la tortue départ à $(0, 0, 90)$ avec le nez vers le haut), et les unités (distance d et angle δ).

D'où vient la connexion entre les grammaires, la tortue et les plantes ? Le principe est que les règles décrivent le développement de la plante : la règle $F \rightarrow F[+F]F$, par exemple, exprime la poussée avec une branche à la gauche ($+F$ enfermé en crochets). Des alphabets et de règles plus sophistiquées peuvent modéliser la formation d'organes différentes en 3D.

TABLE 2: Encodage standard des actions

action	symbole
push	[
pop]
turnL	+
turnR	-
draw	F et toute autre lettre majuscule
move	f et toute autre lettre minuscule

1.4 Spécifications

Tortue

Le type abstrait de la tortue comprend les opérations de dessin (`draw`, `move`, `turnL`, `turnR`, `push`, `pop`, `stay`), l'initialisation et les quêtes d'état (`init`, `getPosition`, `getAngle`), et l'affectation/quête de paramètres d'échelle (`setUnits`, `getUnitStep`, `getUnitAngle`). On peut imaginer des implémentations différentes (p.e., dessin en PostScript, ou sur l'écran), donc on déclare la tortue par une interface. On se sert des classes de `java.awt.geom` (dans module `java.desktop`) pour la géométrie. Table 3 montre le l'interface Java de l'implémentation. L'interface inclut une implémentation de la méthode pratique `Runnable action(String nom)` qui récupère l'action nommée comme un exécutable à l'aide des *réflexions*³. Ainsi,

```
turtle.action("pop").run()
```

est identique à `turtle.pop()`.

3. Via `getClass().getDeclaredMethod` et `java.lang.reflect.Method.invoke`

Système de Lindenmayer

L'API du L-système inclut la représentation de symboles, de chaînes, et de règles, ainsi que les opérations pour dériver les chaînes, et pour les interpréter comme instructions à la tortue :

- ★ représentation de symboles de l'alphabet : une classe simple `Symbol` encapsulant un caractère qui vous est fournie ;
- ★ représentation de chaînes de symboles : `Iterator<Symbol>` définit le parcours de la chaîne ;
- ★ représentation du L-système par une classe avec des méthodes publiques pour initialisation : `Symbol setAction(Character c, String action)` définit l'action de la tortue pour le caractère donné, et initialise le symbole qui représente le caractère donné et définit l'action de la tortue associée ;
`void setAxiom(String str)` définit l'axiome α ; `void addRule(char gauche, String droite)` ajoute une règle ;
- ★ accès aux règles et les actions : `Iterator<Symbol> getAxiom()` récupère l'axiome ; `Iterator<Symbol> rewrite(Symbol sym)` retourne la substitution selon une règle choisie au hasard parmi celles avec `sym` à la gauche ou `null` si aucune règle n'applique (symbole terminal),
`void tell(Turtle turtle, Symbol sym)` demande la tortue à exécuter l'instruction associée avec `sym` (celle qui avait été spécifiée par `setAction`) ;
- ★ inférence (description en bas) : `Rectangle2D tell(Turtle turtle, Iterator<Symbol> seq, int rounds)` pour exécuter les instructions après `rounds` itérations de réécriture.

Table 4 montre l'API Java pour le L-système.

```

package lindenmayer;
import java.awt.geom.Point2D;
public interface Turtle
{
    /* actions */
    public void draw();
    public void move();
    public void turnR();
    public void turnL();
    public void push();
    public void pop();
    public void stay();
    /**
     * Access to the interface actions by String name
     */
    public default Runnable action(String name) { ... }
    /**
     * Initializes the turtle state (and clears the state stack)
     * @param pos turtle position
     * @param angle angle in degrees (90=up, 0=right)
     */
    public void init(Point2D position, double angle);
    /**
     * position of the turtle
     * @return position as a 2D point
     */
    public Point2D getPosition();
    /**
     * angle of the turtle's nose
     * @return angle in degrees
     */
    public double getAngle();
    /**
     * sets the unit step and turn
     * @param step length of an advance (move or draw)
     * @param delta unit angle change in degrees (for turnR and turnL)
     */
    public void setUnits(double step, double delta);
    public double getUnitStep();
    public double getUnitAngle();
}

```

TABLE 3: Interface de programmation pour tortue

```

package lindenmayer;
import java.awt.geom.Rectangle2D;
import org.json.JSONObject;
public class LSystem ... {
    /* constructeur pour un système avec alphabet vide et sans règles */
    protected LSystem(){ ... }
    /* méthodes d'initialisation de système */
    public Symbol setAction(char sym, String action){ ... }
    public void setAxiom(String axiom){...}
    public void addRule(Symbol sym, String expansion) {...}

    /* accès aux règles et exécution */
    public Iterator<Symbol> getAxiom(){ ...}
    public Iterator<Symbol> rewrite(Symbol sym) {...}
    public void tell(Turtle turtle, Symbol sym) {...}

    /* inférence + dessin */
    /* retourne BoundingBox pour le dessin */
    public Rectangle2D tell(Turtle turtle, Symbol.Seq seq, int n){ ...}
    ...
    /* initialisation par fichier JSON */
    protected void initFromJson(JSONObject obj, Turtle turtle);
}

```

TABLE 4: Interface de programmation pour L-système

1.5 Entrée-sortie

On initialise le système et l'échelle de la graphique en un fichier de format JSON⁴ qui suit la syntaxe de Javascript avec les clés :

- ★ `actions` : associations entre symboles et les actions de la tortue
- ★ `axiom` : chaîne de départ (string);
- ★ `rules` : règles comme un objet d'associations de symbole à tableau de strings (toutes les règles pour un symbole)
- ★ `parameters` : initialisation de la tortue, un objet avec `step` (d), `angle` (δ), et `start` qui est un tableau avec 3 éléments numériques du l'état initial (x, y, θ)

Table 5 montre un exemple.

```
{
  "actions": {"F": "draw", "[": "push", "]": "pop", "+": "turnL", "-": "turnR"},
  "rules": {"F" : ["F[+F]F[-F]F", "F[+F]F", "F[+F]F[-F]F"]},
  "axiom": "F",
  "parameters" : {"step": 2, "angle": 22.5, "start": [0,0,90]}
}
```

4. W₍₆₎:format JSON: *JavaScript Object Notation*

TABLE 5: Fichier JSON pour spécification du L-système et les paramètres graphiques

L'API pour l'initialisation comprend la méthode

```
initFromJson(JSONObject obj, Turtle turtle)
```

qui initialise le système et la tortue à partir d'un fichier JSON. Notez qu'il est extrêmement simple de lire un tel fichier : installez le package **JSON-java**⁵, et consultez sa documentation⁶. Par exemple :

```
protected void initFromJson(JSONObject pars, Turtle turtle) {
    JSONObject actions = pars.getJSONObject("actions");
    for (String code: actions.keySet()) {
        String action = actions.getString(code);
        char c = code.charAt(0);
        this.setAction(c, action);
    }
    String axiom = pars.getString("axiom");
    this.setAxiom(axiom);
    ...
}
```

On construit l'argument en lisant le fichier à l'entrée :

```
pars = new JSONObject(new JSONTokener(new FileReader(file)));
```

Votre application principale, lancée à partir de la ligne de commande, lit le fichier JSON du L-système (spécifié comme premier argument), et affiche le dessin sur la sortie standard (`System.out`), en format EPS (PostScript encapsulé). D'autres applications utilisent des tortues différentes pour afficher dans un autre format, ou sur l'écran.

5. <https://github.com/stleary/JSON-java>

6. <http://stleary.github.io/JSON-java/index.html>

1.6 Travail à faire

a. *Tortue à crayon imaginaire.* ► Implémenter une classe bidon avec l'interface de tortue qui maintient les états correctement, mais ne dessine rien en vérité.

Indice: Je recommande une classe imbriquée `State` pour encapsuler l'état, et la dèque `java.util.ArrayDeque` (mieux que `java.util.Stack` qui utilise des méthodes synchronisées) pour empiler-dépiler. Faites attention à la conversion degré \leftrightarrow radian.

b. *L-système.* ► Implémenter les opérations principales du TA (`setAction`, `setAxiom`, `addRule`, `getAxiom`, `rewrite`, et `tell(Turtle, Symbol)`) avec la classe `Symbol`, et une classe `LSystem`.

Indice: Utilisez toujours la même instance de `Symbol` — une table `Map<Character, Symbol>` est utile pour les associations des `Symbol` avec les `chars`. On peut stocker les règles en un `Map<Symbol, List<Iterable<Symbol>>` — comme ça, on peut récupérer toutes les règles d'un symbole dans une liste et en choisir une au hasard.

c. *Initialisation par fichier.* ► Implémenter la méthode `initFromJson` qui initialise le système à partir d'un objet JSON donné (avec clés `actions`, `rules`, `axiom`, et `parameters`).

d. *Déclencher la bête* ► Implémenter une méthode récursive

```
Rectangle2D tell(Turtle T, Iterator<Symbol> seq, int n)
```

qui performe n itérations de réécriture sur `seq`, et exécute la chaîne résultante avec la tortue spécifiée. La méthode retourne les dimensions de la rectangle minimale incluant toutes les positions visitées par la tortue, ou le *bounding box*.

Indice: On peut éviter le calcul explicite de la chaîne complète⁷, en exploitant la récursivité. Si $n = 0$, on exécute directement la chaîne avec `tell(turtle, sym)` symbole-par-symbole (et on consulte la position de la tortue); sinon on fait des appels récursifs (avec `tell(..., n-1)`). Élaborez l'algorithme sur papier avant de coder (p.e., vérifiez l'exécution avec $n = 2, 3$ et des règles simples). Si la tortue passe par les coordonnées $(x_1, y_1), \dots, (x_m, y_m)$, alors on a besoin des coordonnées extrêmes $x_{\min} = \min\{x_1, \dots, x_m\}$, x_{\max} , y_{\min} , et y_{\max} . Il est pratique de travailler avec des rectangles en calculant leur union comme dictée par la géométrie (étudiez la documentation de `Rectangle2D`).

e. *Application complète.* ► Implémenter une application complète (JAR exécutable appelé `lindenmayer.jar`) pour affichage en format EPS (Encapsulated PostScript), à la sortie standard (par défaut). On la lance à partir de la ligne de commande avec 2 arguments obligatoires : (1) le nom du fichier JSON avec la description du L-système, et (2) le nombre d'itérations de réécriture. Dans un shell Linux :

```
% java -jar lindenmayer.jar test/buisson.json 5 > buisson5.eps
% epstopdf buisson5.eps > buisson5.pdf
```

7. Par contre, une solution avec le calcul *paresseux* de la chaîne

```
Iterator<Symbol> chain = rewrite(seq, n);
```

est possible où la version récursive de `rewrite` retourne un itérateur qui applique la réécriture seulement au besoin, un symbole à la fois afin de minimiser l'usage de mémoire. L'idée est de combiner l'itérateur de `seq` avec un itérateur sur l'expansion et mettre ce dernier à jour par récursion avec `rewrite(..., n-1)`. Ensuite, on peut appeler `tell(turtle, sym)` en parcourant les symboles de `chain`.

Indice: Il suffit d'implémenter une tortue qui affiche du code PostScript lors de ses actions et l'utiliser dans `tell(...)`. Par exemple :

- ★ au début, écrire l'en-tête structurée `!PS-Adobe-3.0 EPSF-3.0`, suivi par des lignes de commentaires incluant `%BoundingBox: (atend)`, récupérer la position initiale de la tortue, et initialiser un chemin de dessin avec `newpath x y moveto` ;
- ★ déplacer ou tracer par `x y moveto` ou `lineto` dans `move()` et `draw()`, avec la position courante (x, y) retournée par `getPosition()`
- ★ lors de `push()`, dessiner le chemin existant, et recommencer un nouveau chemin pour dessin `currentpoint stroke newpath moveto` (`currentpoint` sauvegarde la position courante de PostScript sur la pile, parce que `stroke` efface la position, et ainsi on commence le nouveau chemin dans exactement la même position) ;
- ★ lors de `pop()`, dessiner le chemin existant, et recommencer un nouveau chemin à la position de la tortue récupérée `stroke x y newpath moveto` .
- ★ à la fin, dessiner le chemin par `stroke` , et écrire le `%Trailer` avec le `%BoundingBox` ;

Table 6 montre l'écriture du fichier EPS avec une telle approche.

```
// en-tête de fichier EPS
out.println("!PS-Adobe-3.0 EPSF-3.0");
out.printf("%%%Title: (%s)\n", title);
out.println("%%Creator: (" + getClass().getName() + ")");
out.println("%%BoundingBox: (atend)"); // on saura le bounding box après avoir dessiné
out.println("%%EndComments");
...
Rectangle2D bbox = bbox = lsystem.tell(turtle, lsystem.getAxiom(), n); // dessin par tortue
// fin du fichier EPS
out.println("%%Trailer");
out.printf("%%%BoundingBox: %d %d %d %d\n", (int)bbox.getMinX(), (int)bbox.getMinY(),
    , (int)bbox.getMaxX(), (int)bbox.getMaxY());
out.println("%%EOF");
```

TABLE 6: Écriture du fichier EPS

Mon code

Vous trouverez mon code source dans le fichier JAR qui vous êtes fourni :

- ★ classe `Symbol` ;
- ★ interface `Turtle` ;
- ★ classe abstraite `AbstractLSystem` qui définit l'API et offre des méthodes pour produire des valeurs pseudo-aléatoires. En particulier, on peut l'instancier avec la graine (*seed*) du générateur de nombres aléatoires (`java.util.Random`) et puis le remettre dans le même état ce qui est utile pour arriver à des exécutions identiques (dans débogage). Utilisez, modifiez, ou ignorez cette classe mais respectez la spécification.

Implémentations alternatives (2 points boni)

Si vous ne faites pas l'implémentation Java (a.–e.), l'implémentation en PostScript (f.) vaut le travail complet (9 points). Autrement, vous pouvez soumettre aussi (f.) ou (g.) pour 2 points boni.

f. Calcul en PostScript. ► Implémenter une application où l'inférence des chaînes se fait dans le code PostScript. L'exécutable (en Java ou autre langage) contient le code minimal pour lire la spécification JSON et afficher le résultat à la sortie standard. La sortie (toujours EPS) inclut le code pour implémenter l'inférence de chaînes, et ainsi sa taille ne dépend pas de la taille du dessin résultant.

Indice: Le résultat PostScript exploite la récurrence, en traduisant les règles différentes en opérateurs qui prennent un seul argument. Ce seul argument spécifie la profondeur d'expansion : si on arrive à 0, on exécute l'opération appropriée avec la tortue, sinon on soustrait 1 de la profondeur et applique une règle. Le code de Table 7 montre cette idée pour la règle $F \rightarrow F-F$.

Si on utilise les transformations de l'espace graphique (`translate` pour `move` et `draw` et `rotate` pour tourner), alors `gsave` et `grestore` suffisent pour `push` et `pop`.

g. Affichage sur l'écran ou autre format graphique. Vous pouvez élaborer le support pour autres clients avec relativement peu de codage à l'aide de tortues différentes.

► Implémenter une application Java avec affichage sur l'écran par JavaFX (par une tortue qui connaît le `GraphicsContext`) ou par Swing, ou avec affichage dans le format Scalable Vector Graphics⁸ (par une tortue qui construit un `<chemin>`), ou un autre format graphique approuvé (discutez avec le prof).

TABLE 7: Expansion de règles dans PostScript

```
/T:turn % deg turn -
{ rotate } def
/T:draw { ... } def % à développer
/F % iter F -
{
  dup 0 eq % si iter==0
  { % cas de base...
    10 T:draw % dessin avec la tortue
  }{ % cas récursif: expansion de règle
    1 sub % decrements le compteur
    dup F % dupliquer compteur pour 2e F
    -22.5 T:turn % virage gauche
    F
  } ifelse
} def
```

8. W_(fr):SVG

Références

B. Casselman. *Mathematical Illustrations : A Manual for Geometry and PostScript*. Cambridge University Press, 2004. <http://www.math.ubc.ca/~cass/graphics/manual/>; (si vous voulez savoir davantage sur des astuces de produire des dessins).

Adobe Systems Inc. *PostScript Language Reference*. Third edition, 1999. <https://www.adobe.com/jp/print/postscript/pdfs/PLRM.pdf>.

P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 2004. <http://algorithmicbotany.org/papers/abop/abop.pdf>.

G. C. Reid. *Thinking in PostScript*. Addison-Wesley, 1990. (très bonne discussion de la pratique de programmation en PostScript).