RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

UNIVERSITÉ CAEN NORMANDIE

LICENCE 2 INFORMATIQUE

Interpreteur de systemes de Lindenmeyer L-Système

Auteurs :
AISSATOU DIALLO
BOLUWATIFE ADEKOYA
PATRICE D. Z. COTCHO
JEREMIE LAYIZIA

Chargé du cours :
Bonnet GRÉGORY
Encadrant TP :
Boiteau ANTOINE

11 avril 2024

Table des matières

Introduction

Contexte

Le travail en groupe sur un projet offre une approche collaborative qui permet de tirer parti des compétences individuelles, de favoriser la communication et la résolution de problèmes, de répartir efficacement les tâches et de renforcer le sentiment d'appartenance à l'équipe. C'est dans cette logique qu'il nous a été proposé plusieurs sujets afin de travailler en groupe de quatre (4) étudiants. Le but étant de mettre en pratique nos connaissances acquises tout au long de l'année en programmation orientée objet (Java) pour réaliser une application fonctionnelle de bout en bout. Parmi les sujets proposés, nous avons choisi celui dont le thème est : "Interpréteur de systèmes de Lindenmeyer". Les systèmes de Lindenmeyer, ou L-systèmes, permettent de représenter des modèles de végétaux sous forme de système de réécriture. C'est un modèle formel utilisé pour décrire et simuler la croissance de structures biologiques telles que les plantes, les algues, ou même des structures artificielles comme les fractales. Il a été inventé par le biologiste théoricien Aristid Lindenmayer dans les années 1960.

Objectifs

Le but de notre projet étant de réaliser une application d'interprétation de L-systèmes qui prend en entrée des règles de réécriture et produit une image 2D ou une scène 3D, nous avons quatre (04) objectifs à atteindre. Il s'agit de :

1. Définir un langage pour construire un système de Lindenmeyer classique ;
2. Implémenter un interpréteur pour une visualisation 2D ;
3. Implémenter un interpréteur pour une visualisation 3D ;
4. Étendre le langage et les interpréteurs pour des systèmes stochastiques et/ou contextuels.

Organisation

Ce rapport est subdivisé en quatre chapitres. Le chapitre 1 traite de la revue de littérature. Dans ce dernier, nous avons exposé la notion de L-system. Le chapitre 2 est consacré à l'organisation du projet. Les éléments techniques de notre application ont été présentés dans le chapitre 3. Dans le dernier chapitre (chapitre 4), nous avons montré comment utiliser l'application.

Chapitre 1

Revue de littérature

1.1 Définition de L-System

Un L-System est un ensemble de règles et de symboles qui modélise un processus de croissance d'êtres vivants tels que des plantes ou des cellules. Le concept central des L-Systems est la notion de réécriture. La réécriture est une technique pour construire des objets complexes en remplaçant des parties d'un objet initial simple en utilisant des règles de réécriture. Pour ce faire, les cellules sont modélisées à l'aide de symboles. À chaque génération, les cellules se divisent, c'est-à-dire qu'un symbole est remplacé par un ou plusieurs autres symboles formant un mot [2].

Il a été inventé en 1968 par le biologiste hongrois Aristid Lindenmayer qui l'avait pensé comme un langage formel permettant de décrire le développement d'organismes multicellulaires simples. À cette époque, il travaillait sur les levures, les champignons et les algues. Mais l'informatique a permis d'exploiter ce système pour générer graphiquement des calculs de plantes très complexes.

1.2 Fonctionnement des L-Systèmes

Un L-System comprend :

- **Un alphabet :** l'ensemble des variables et constantes du L-Système ;
- **Un axiome :** qui représente le premier symbole à partir duquel on commence la synthèse des symboles suivants ;
- **Un ensemble de règles :** qui permettent de transformer un symbole en une chaîne de symboles.

En général, le système est utilisé pour simuler la croissance des arbres à partir d'un axiome (symbole de départ). Des règles définies préalablement seront appliquées à une suite de symboles pour donner naissance à une nouvelle chaîne. Le résultat final dépendra du nombre d'itérations qui seront interprétées et affichées sous forme de graphique 2D ou 3D. La figure ci-dessous montre un exemple d'utilisation de toutes ces notions abordées ci-dessus.

Axiome :	X
Règle 1 :	X=F[+X][-X]FX
Règle 2 :	F=FF
iteration 1	F[+X][-X]FX
Iteration 2	FF[+F[+X][-X]FX][-F[+X][-X]FX]FFF[+X][-X]FX
Iteration 3	FFFF[+FF[+F[+X][-X]FX][-F[+X][-X]FX]FFF[+X][-X]
	FX][-FF[+F[+X][-X]FX][-F[+X][-X]FX]FFF[+X][-X]FX]
	FFFFFF[+F[+X][-X]FX][-F[+X][-X]FX]FFF[+X][-X]FX

FIGURE 1.1 – Exemple de génération

1.3 Interprétation des symboles pour les rendus 2D et 3D

Un l-système étant essentiellement composé de symboles, notre application utilise les lettres de l'alphabet et des symboles ci-dessous :

Symboles	**Interprétations**
{A..Z}-{H,B}	Avancer d'une unité (dessine une branche)
+	Tourner à gauche d'un angle
-	Tourner à droite d'un angle
[	Enregistrer la position et l'orientation
]	Rétablir la positon et l'orientation
>	Rouler à gauche d'un angle α
<	Rouler à droite d'un angle α
B	S'incliner vers le bas en 3D
H	S'incliner vers le haut en 3D

TABLE 1.1 – Interprétations des symboles

Chapitre 2

Organisation du Projet

2.1 Répartition des tâches

Pour amorcer le projet, nous avons d'abord cherché à comprendre ensemble les L-systèmes dans leur globalité, en nous concentrant particulièrement sur les L-systèmes végétaux. Pour cela, nous nous sommes basés sur les ressources fournies avec le sujet, notamment un livre en ligne [1] détaillant les L-systèmes, afin d'adopter un modèle nous permettant de concevoir notre analyseur syntaxique et notre système de réécriture.

Par la suite, nous nous sommes informés sur les différentes options de moteurs de rendu graphique 2D et 3D. Après examen, notre choix s'est arrêté sur Java pour le rendu 2D et JOGL [1] pour le rendu 3D.

Après avoir saisi l'essence du sujet et choisi les technologies à utiliser, nous avons opté pour un démarrage du projet selon une hiérarchisation des priorités, nous permettant de progresser de manière méthodique et efficace. Nous avons ainsi commencé par la mise en place du Generator et de son modèle de réécriture, puis nous avons abordé le développement de l'interface graphique et du moteur de rendu 2D. Ensuite, nous avons prévu de traiter la fonctionnalité des systèmes stochastiques et enfin le moteur de rendu 3D. Notons que le rapport et la présentation ont été rédigés au fur et à mesure que nous avancions dans la réalisation du projet.

Le projet étant un travail de groupe, nous nous sommes réparti les tâches de la manière suivante : Aissatou **DIALLO** et Esther **ADEKOYA** étaient chargées de développer le Generator et d'écrire ses tests. Elles étaient également responsables de la rédaction du rapport et de la présentation. Patrice D. Z. **COTCHO** a travaillé sur la mise en place de l'interface graphique de l'application, le rendu 2D et les systèmes stochastiques. Enfin, Jérémie **LAYIZIA** s'est chargé de l'implémentation du rendu 3D.

2.2 Technologies utilisées

Les technologies utilisées lors du développement de cette application sont :

1. Java (POO, SWING)
2. Pattern Observer
3. Pattern MVC
4. JOGL

1. (Java Open Graphics Library)

2.3 Architecture du code

Pour des raisons de clarté, de maintenance et de facilité d'ajout de nouvelles fonctionnalités dans l'application, nous avons subdivisé le code en quatre packages :

- **model :** Ce package contient les classes Generator et Rule représentant nos modèles (en faisant référence au modèle MVC).
- **view :** Il contient toutes les classes associées à l'interface utilisateur. On note la présence des sous-packages rendu (contenant les deux rendus) et dialogue (contenant les vues d'information).
- **controler :** On note la présence de la classe AppController qui implémente toute la logique derrière la vue.
- **utils :** Il contient des classes susceptibles d'être utilisées par toutes les autres classes.

La figure 2.1 nous montre les différents packages développés dans l'application et les interactions entre eux. On peut dèjà lire que la classe AppContoller utilise des classes se trouvant dans tous les autres packages.

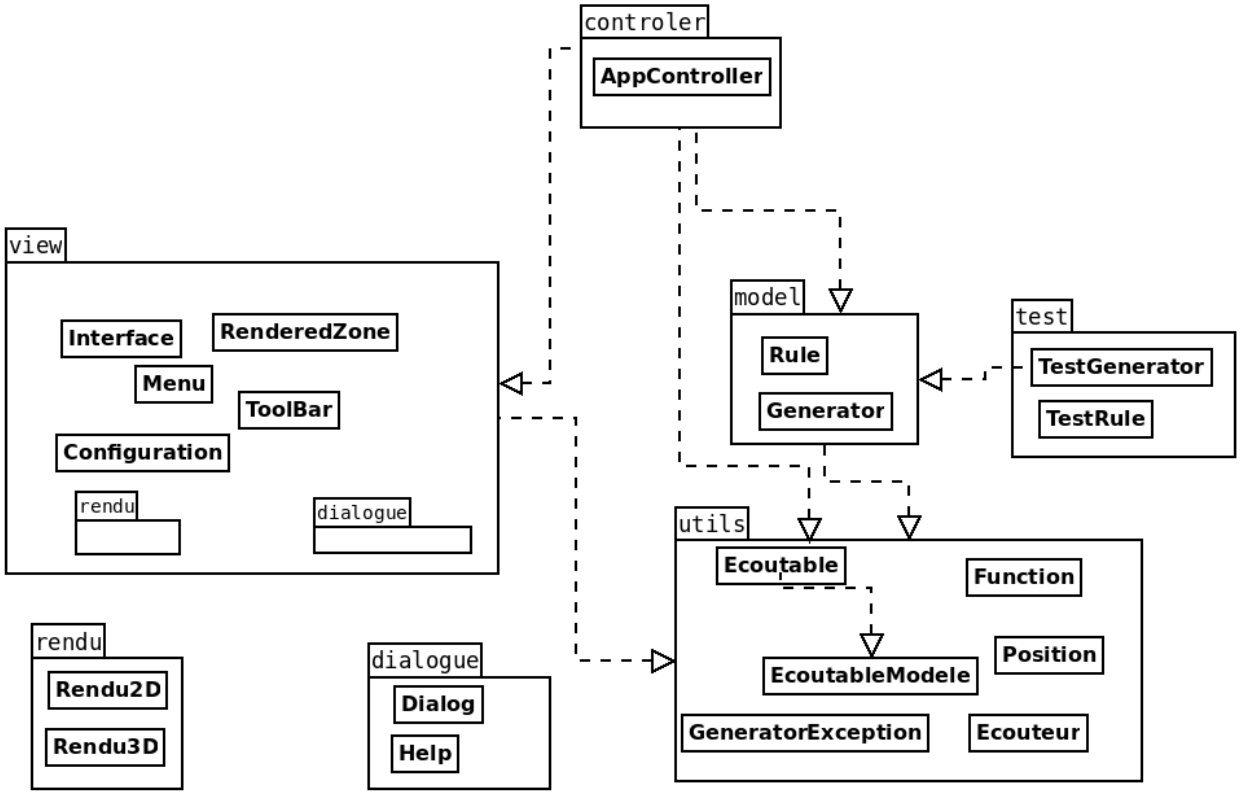

FIGURE 2.1 – Interaction entre les différents packages

La figure 2.2, nous montre une vue plus détaillée des classes développées dans le package model et la relation entre elles. On peut lire sur cette figure que la classe Generator utilise dans sa construction une ou plusieurs instances de la classe Rule.
Plus loin, sur la figure 2.3 nous voyons également les détails sur les classes du package view. Sur cette image, on remarque que la classe interface utilise toutes les autres classes du package dans sa construction. Chaque pièce de l'interface (le menu, la toolbar, la zone de rendu, la zone de configuration) est développée dans une classe séparée. Celà nous a permi de nous retrouver plus vite dans notre code et d'eviter les classes avec des milier de lignes. La classe Interface implémente Ecouteur et donc écoute le Rendu2D et le Rendu3D qui elles implémentent Ecoutable. Chaque classe de ce package étant des objets graphique, elles héritent d'une classe de java.swing.

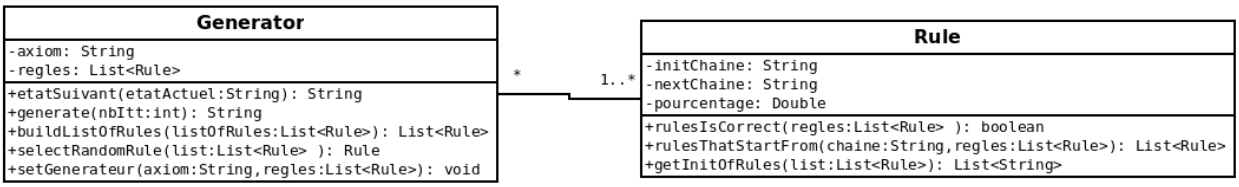

FIGURE 2.2 – Interaction entre les différentes classes du package model

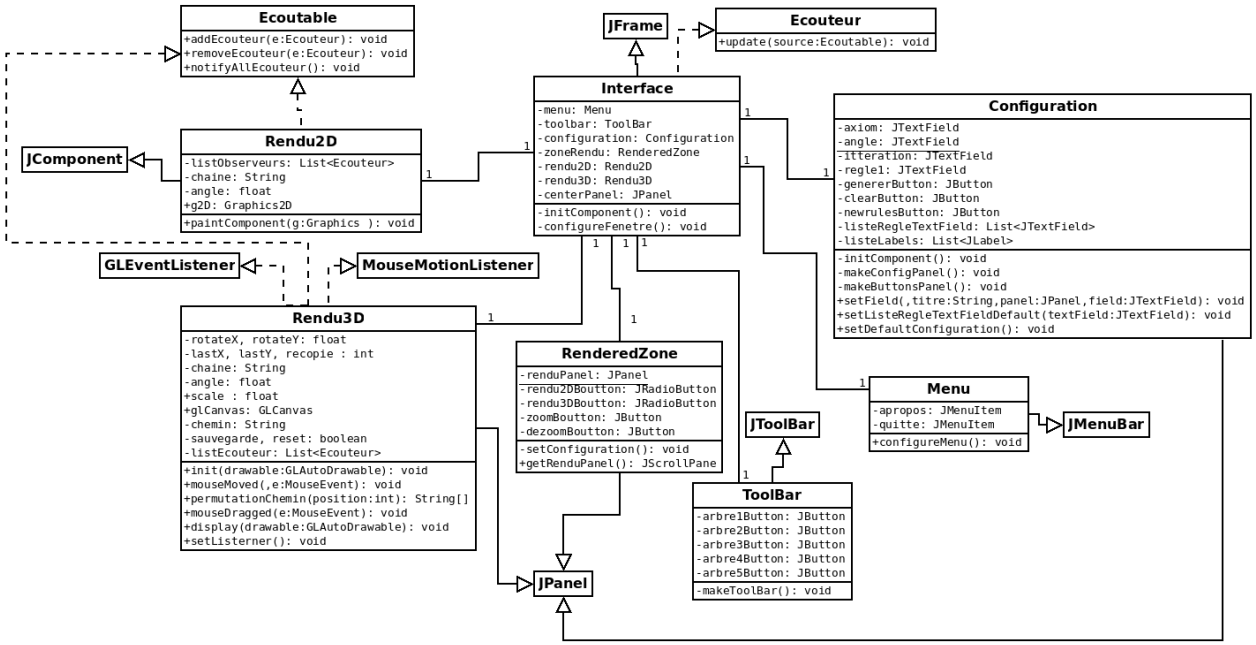

FIGURE 2.3 – Interaction entre les différentes classes du package vue

Enfin la figure 2.4 nous montre les détails sur la classe AppControler du package controler et ses relations avec les classes des autres packages. Il s'agit de l'organisation générale du code. Pour simplifier, nous avons gardé que la classe Interface et décider d'ommetre les autres classes du package view (le détails étant présenté à la figure 2.3). La classe AppController étant chargé d'assurer la logique derrière toute l'application, elle dispose d'une référence sur la vue et les modèles.

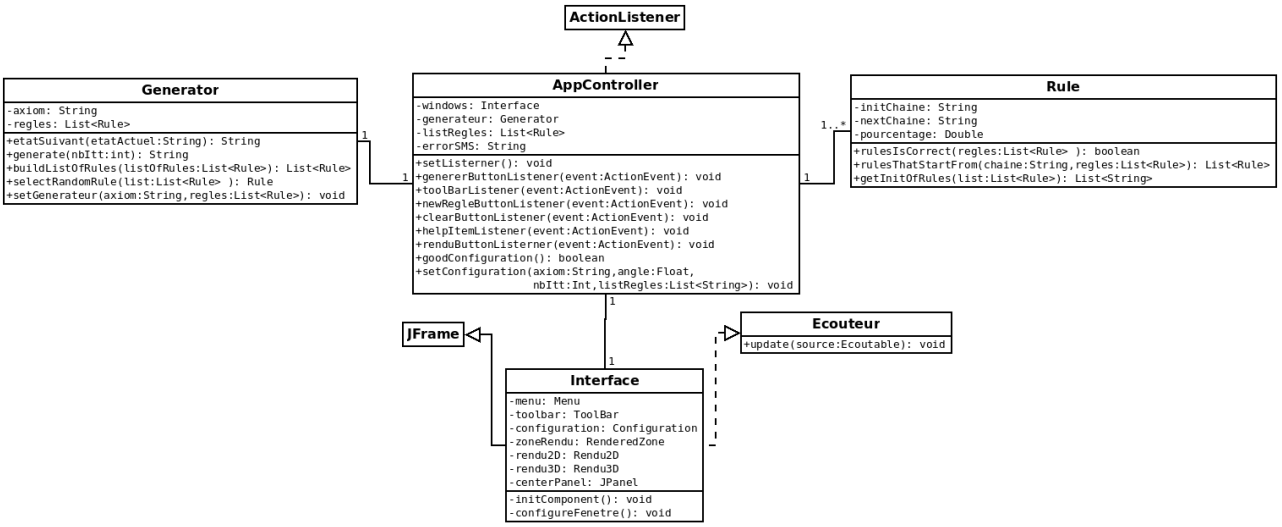

FIGURE 2.4 – Relation entre model vue controleur

Chapitre 3

Élément Technique

3.1 Generator

La class Generator représente le moteur de l'application. Elle est chargée de fournir la chaîne représentant un L-système en se basant sur un axiom de départ, un nombre d'itération , des règles créées à partir de la classe ***Rule***. C'est cette chaîne qui sera à la suite interprétée par les classes Rendu2D et Rendu3D. Ci-dessous, dans l'algorithme 1 nous avons la méthode permettant de générer une chaîne après un nombre d'ittération (nbItt) donné.

Algorithme 1 : Méthode **generate** : calcule la chaine representant un L-System après un nombre (*nbItt*) ittérations.

Entrées : Le nombre d'ittération (*nbItt*) à faire
Sortie : La chaine issue de la production après *nbItt* ittération

```
1 etat ← axiom
2 pour i ← 0 à nbItt faire
3 |   etat = etatSuivant(etat)
4 fin
5 retourner etat ;
```

On remarque que cette méthode utilise une autre méthode nommée **etatSuivant**(algorithme 2). Cette méthode est chargée de calculer l'état suivant d'une chaine donnée en argument en se basant sur les règles définies.

Algorithme 2 : Méthode **etatSuivant**

Entrées : etatActuel : chaine représentant l'état actuel
Sortie : Une chaine representant l'etat suivant

```
1  listedeRegleAleatoire ← construireListeDeRegle(regles)
2  pour chaque caractere dans etatActuel faire
3  |  trouver = Non
4  |  pour i ← 0 à taille de listedeRegleAleatoire faire
5  |  |  si debut de listedeRegleAleatoire(i) == caractere alors
6  |  |  |  trouver ← Oui
7  |  |  |  resultat+ = transformer listedeRegleAleatoire(i)
8  |  |  |  break
9  |  |  fin
10 |  fin
11 |  si trouver == Non alors
12 |  |  resultat+ = caractere
13 |  fin
14 fin
15 retourner resultat ;
```

L'algorithme 3 permet de construit une liste de règle qui sera utilisé par la méthode **etatSuivant** 2 pour calculer l'état suivant d'une chaine. On remarque que cet algorithme utilise trois autres méthodes dans son exécution à savoir : **obtenirDebutDesRegles**, **obtenirDebutDesRegles**,**obtenirDebutDesRegles**. L'algorithme 4 montre comment se fait le choix d'une règle en se basant sur la probabilité d'apparition de chacune des règles d'une liste.

Algorithme 3 : Méthode **construireListeDeRegle** : construit une liste de règle qui sera utilisée pour la production d'un état suivant.

Entrées : listeDeRegle : Une liste de règles
Sortie : Une autre liste de règles

```
1  listeDesDebutDesRegles ← obtenirDebutDesRegles(listeDeRegle)
2  pour caractere ← listeDesDebutDesRegles faire
3  |  regles ← regleCommencantPar(caractere, listeDeRegle)
4  |  resultat.ajouter(selectionnerUneRegleAléatoirement(regles))
5  fin
6  retourner resultat ;
```

Algorithme 4 : Méthode **selectionnerUneRegleAléatoirement** : construit une liste de règle qui sera utilisée pour la production d'un état suivant.

Entrées : Une liste de règles (*liste*) ayant une même chaîne comme chaineInit
Sortie : Une règle selectionné en fonction de la probabilité d'apparition de chacune des règles de *liste*

```
1  listeDePoid ← listeVide
2  listeDePoid.ajouter(liste.get(0).pourcentage()
3  pour i ← 1 à taille(liste) faire
4  |  listeDePoid.ajouter(liste.get(i).pourcentage() + listeDePoid.get(i − 1))
5  fin
6  nombreAlatoire = On_tire_un_nombre_alatoire_entre_1_et_100
7  indexDeRegleAleatoire ← 0
8  pour i ← 1 à taille(listeDePoid) faire
9  |  si (nombreAlatoire − listeDePoid.get(i)) ≤ 0 alors
10 |  |  indexDeRegleAleatoire = i
11 |  |  break
12 |  fin
13 fin
14 retourner liste.get(indexDeRegleAleatoire) ;
```

3.2 Rule

C'est la classe qui permet de représenter une règle. Une règle permet de spécifier la transformation à effectuer sur un symbole ou une chaîne pour passer de l'état actuel à l'état suivant. Dans l'objectif de mettre en place la fonctionnalité de systèmes stochastiques, nous avons doté nos règles de trois attributs : *initChaine*, *nextChaine* et *pourcentage*. *Pourcentage* représente la chance d'apparition ou de sélection d'une règle dans une liste de règles ayant les mêmes chaînes de départ (*initChaine*). La somme des pourcentages des règles ayant une même chaîne de départ doit obligatoirement faire 100. Sur l'interface graphique la synthaxe pour représenter une règle est celle ci : initChaine :nextChaine :pourcentage. Sans cette synthaxe correcte, l'application nous renvoie un message d'erreur. **F :F[FX[X]F] :100** est un exemple de règle valide.

3.3 Interface

Vu que nous avons étudié Java Swing en cours d'interface graphique et que nous sommes familliarisé avec ce dernier nous avons décidé de l'utiliser. Nous avons donc créer plusieurs classes dont Interface (sous classe de JFrame) , Menu (sous classe de JMenuBar), ToolBar (sous class de JToolBar), RenderedZone et Configuration (qui sont des sous classes JPanel). La classe Interface constitue la fenêtre principale de notre application.

3.4 Moteur graphique

En ce qui concerne nos moteurs de rendu, nous utilisons du java swing pour le rendu 2D et JOGL [1] pour le rendu 3D.

1. Java Open Graphics Library

3.4.1 Rendu 2D

La classe Rendu2D est une sous-classe de JComponent (javax.swing.JComponent). Grâce à la méthode ***paintComponent(Graphics2D)*** de cette dernière, nous avons pu dessiner ce composant nous-mêmes. La méthode **drawLine** de la classe **Graphics2D** nous a permis de tracer les lignes (branches). Nous avons utilisé une classe Position pour sauvegarder une position. La figure 3.1 montre une vue d'un arbre obtenu grâce au rendu 2D que nous avons implémenté.

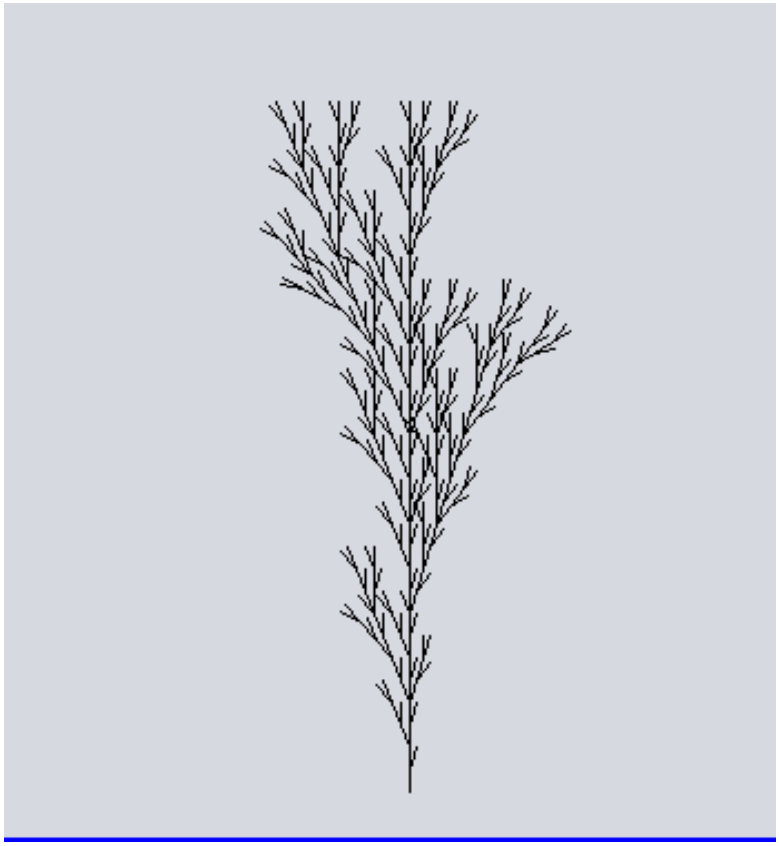

FIGURE 3.1 – Arbre obtenu via le rendu 2D

3.4.2 Rendu 3D

Comme mentionné ci-dessus, nous avons utilisé la bibliothèque Java OpenGL (JOGL) pour réaliser le rendu 3D. Notamment nous nous sommes servis de : GL2 pour afficher les lignes, GLU pour placer la caméra, et GLUT pour afficher les cylindres. GLCanvas crée la fenêtre, GLEventListener initialise et affiche l'environnement 3D.

FIGURE 3.2 – Exemple de rendu 3D

Chapitre 4

Expérimentation

Notre avons développé notre application avec du java standard Édition 11 pour des raisons de compatibilité avec les ordinateurs de l'université.

4.1 Lancement de l'application

Nous avons proposé deux moyens pour lancer l'application. Le premier étant l'exécution du .jar situé dans le dossier dist. Il sufit donc, une fois à la racine du projet d'exécuter les commandes suivantes :

1. cd dist/
2. java -jar Lsystem.jar

Nous avons également écrit un script bash (App.sh) pour simplifier le lancement de l'application. Ce fichier se trouve dans le dossier src.Il faut donc une fois à la racine du projet, lancer les commandes linux suivantes :

1. cd src/
2. chmod +x App.sh
3. ./App.sh

4.2 Fonctionnement de l'interface graphique

Au lancement du programme, l'utilisateur voit l'interface de la figure 4.1. En haut, dans la barre d'outils, se trouvent des boutons représentant quelques exemples d'arbres. En cliquant sur l'un d'eux, l'utilisateur verra s'afficher dans la zone de rendu (la zone bordée en couleur bleue) l'arbre correspondant en 2D. Juste au-dessus de la zone de rendu se trouvent les boutons permettant de passer de la vue 2D à la vue 3D. À droite, nous avons la zone de configuration, où l'utilisateur peut renseigner les paramètres de l'arbre qu'il souhaite afficher. Il convient de noter que la configuration de l'arbre obtenu via les exemples de la barre d'outils est automatiquement renseignée dans cette zone, permettant ainsi à l'utilisateur de s'en inspirer. En bas de la zone de configuration se trouvent les boutons de contrôle. En bas de la zone de rendu se trouvent les boutons de zoom et de dézoom. Ces boutons ne sont malheureusement fonctionnels qu'en 3D.

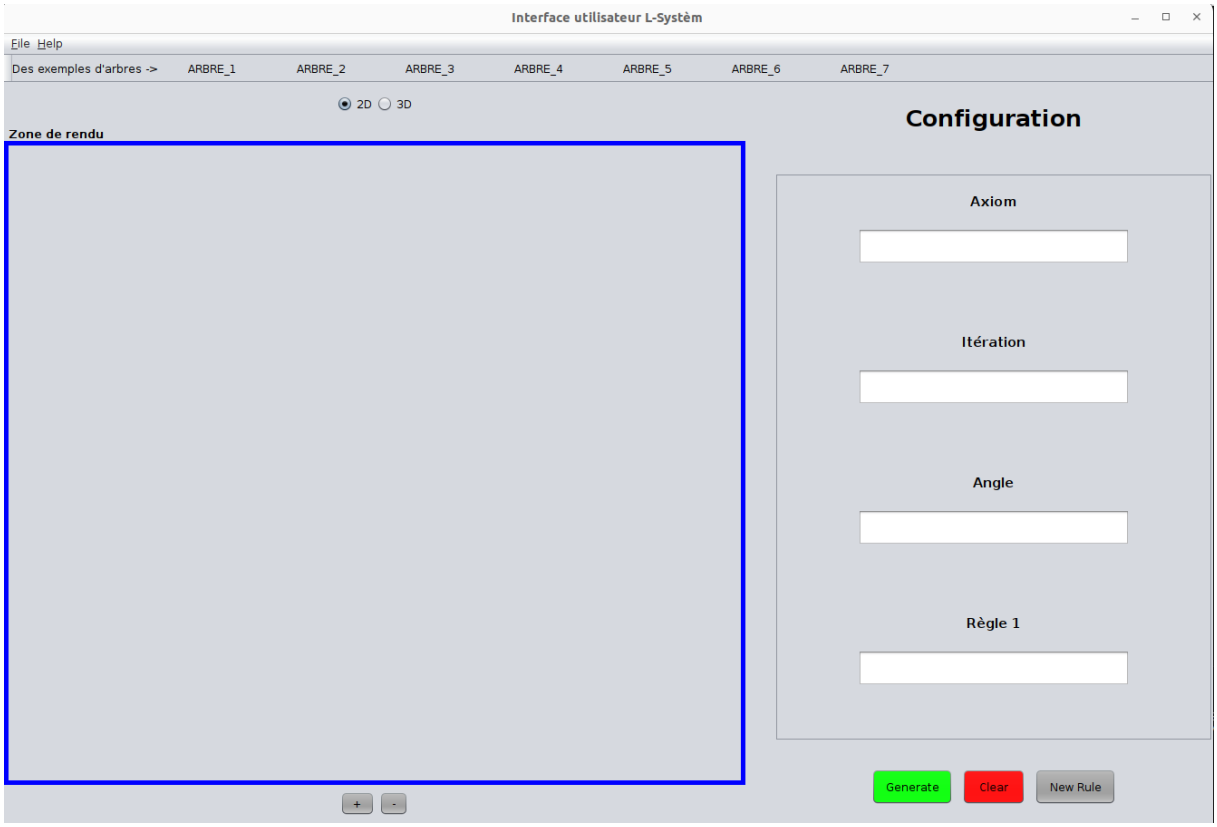

FIGURE 4.1 – Interface utilisateur de l'application

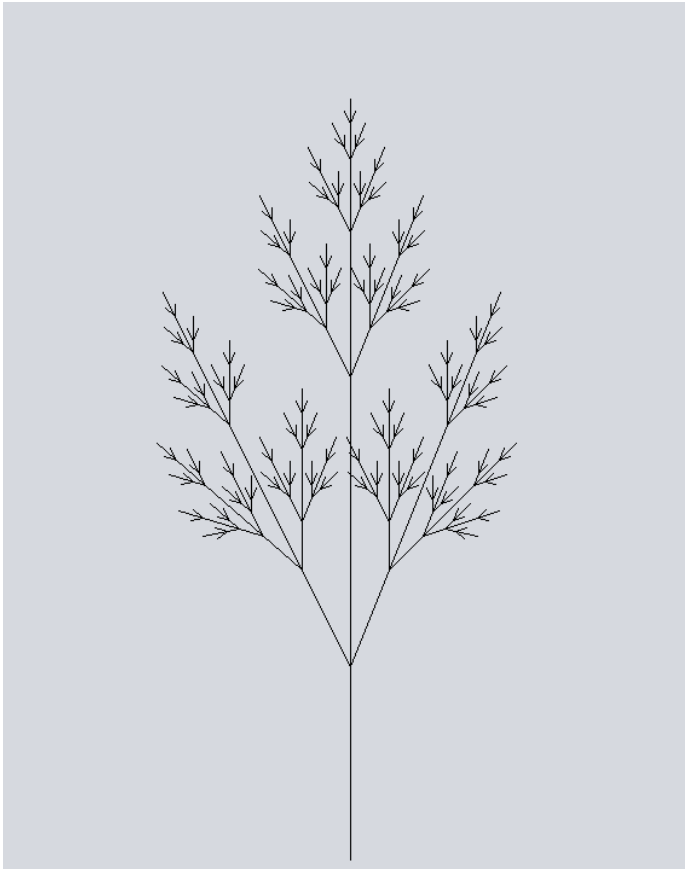

FIGURE 4.2 – Arbre de l'exemple 4 en 2D

Un clic glissé de la souris sur le rendu 3D :

1. **Vers la droite :** permet de faire tourner l'arbre autour de lui-même dans la même direction ;
2. **Vers la gauche :** permet de faire tourner l'arbre autour de lui-même dans la même direction ;
3. **Vers le bas :** incline l'arbre vers le bas.

4.3 Test du logiciel

Nous avons utilisé le framework open source **Junit-4** pour le développement et l'exécution de tests unitaires. Deux classes ont été principalement testées. Il s'agit de **Generator** et **Rule** qui représentent nos modèles. Les résultats issus de ces tests sont présentés par les figures 4.3 et 4.4. On remarque que tous les tests effectués se sont bien déroulés.

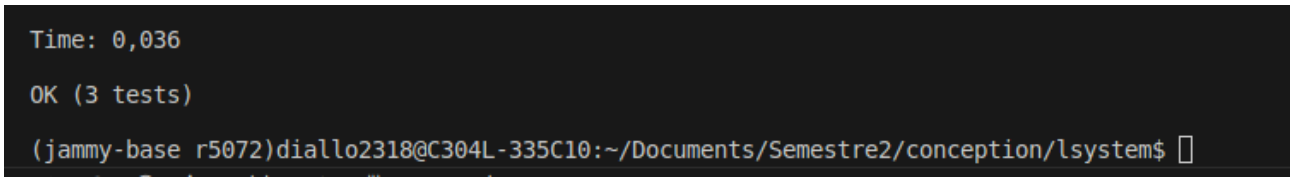

FIGURE 4.3 – Résultat des tests effectués sur la classe Generator

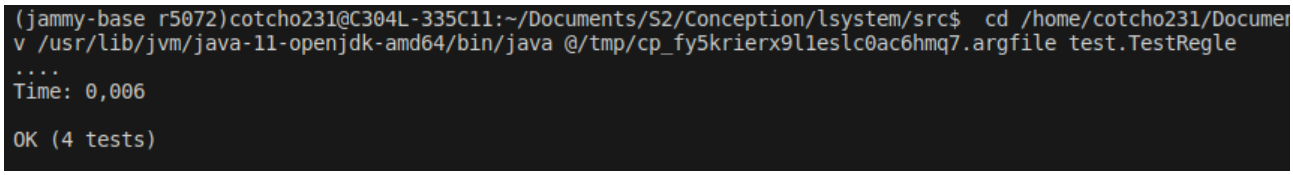

FIGURE 4.4 – Résultat des tests effectués sur la classe Rule

Conclusion

Bilan des travaux

La réalisation de ce projet sur l'interpréteur des systèmes de Lindenmeyer a été une expérience enrichissante et stimulante pour notre équipe. À travers ce travail, nous avons pu concrétiser nos connaissances théoriques en programmation orientée objet, en mettant en pratique des concepts fondamentaux dans le développement logiciel. En nous concentrant sur les objectifs définis, nous avons progressivement élaboré une application fonctionnelle et polyvalente, capable de générer des représentations visuelles à la fois en 2D et en 3D à partir de règles de réécriture de L-systèmes.

Au fil de notre travail, nous avons également eu l'occasion de consolider nos compétences en matière de communication et de gestion de projet. La répartition efficace des tâches, les échanges régulières et la résolution collective des problèmes ont contribué à renforcer notre cohésion d'équipe et à maintenir un rythme de progression soutenu tout au long du projet.

Obstacles rencontrés

La mise en œuvre de cet interpréteur nous a confrontés à divers défis techniques, nous obligeant à repousser nos limites et à rechercher des solutions innovantes. De la conception du langage à l'implémentation des interpréteurs, en passant par l'extension pour prendre en charge des systèmes stochastiques, chaque étape du processus a nécessité une réflexion approfondie et une collaboration étroite au sein de notre équipe et avec le professeur chargé du TP. Ce dernier nous a été d'une grande aide car il nous aidait à trouver des solutions à chaque fois qu'on avait des incompréhensions sur des notions ou qu'on était bloqué. Nos difficultés était surtout au niveau de :

— L'implémentation de la fonctionnalité des systèmes stochastiques
— Mise en place de la vue 3D

Discussion

Etant conscient que tout n'est pas parfait dans l'application, nous tenons à souligner les possibles problèmes que l'utilsateur peut rencontrer lors de son utilisation. En effet, lorsqu'on démarre l'application et qu'on essaie de générer un arbre avec un nombre d'itération au delà d'un certain seuil, la représentation 2D et ou 3D pourrait déborder. Dans ce cas une solution serait de diminuer le nombre d'itération. En 3D nous avons la possibilité de dézoomer. D'autre part nous avons essayé de limiter la taille de la chaine générée par la méthode **generate** de la classe **Generator** à 20000 caractères. Toujours dans le but de contôler la taille de l'arbre.

Améliorations possibles

Plusieurs fonctionnalités peuvent faire évoluer ce projet. Il s'agit entre autres de :
— La simulation d'une foret par ajout de plusieurs arbres dans la même fenêtre cote à cote ;
— La possibilité de zommer et de dezommer le rendu 2D ;
— Faire en sorte que l'arbre grandisse dans la direction du soleil ;
— Ajouter des contrôles clavier sur le rendu 3D.

Bibliographie

[1] Przemyslaw Prusinkiewicz & Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag `http://algorithmicbotany.org/papers/abop/abop.pdf`, New York, electronic version edition, 1990 - 1996.

[2] ManiacParisien. L-system - définition. `http://www-cs-faculty.stanford.edu/~uno/abcde.html`.