Rapport

Interpréteur de systèmes de Lindenmeyer

Conception logicielle 1

Salah Eddine ELOUARDI

14 juillet 2024

Université de Caen Normandie UFR
des Sciences Département
Informatique
2ème année de licence d'informatique

Table des matières

1 Introduction

1.1 But du Projet

L'objectif principal de ce projet est de développer un interpréteur de L-systèmes en utilisant le langage de programmation Java. Cet interpréteur sera capable de transformer une séquence de symboles, partant d'un symbole initial et suivant des règles de réécriture spécifiées, en représentations visuelles des modèles végétaux qu'elles décrivent. Ces modèles, inspirés de la nature, sont fréquemment utilisés dans les univers virtuels tels que les jeux vidéo et les films d'animation, où ils contribuent à enrichir les paysages et les environnements avec des arbres, buissons et autres formes de végétation.

Pour atteindre cet objectif, le projet sera structuré en plusieurs étapes clés, comme suit :

- **Définition du Langage :** Définir un langage pour exprimer les règles de L-systèmes classiques.
- **Visualisation 2D :** Développement d'un interpréteur pour convertir les règles en images 2D des structures végétales.
- **Visualisation 3D :** Extension de l'interpréteur pour générer des visualisations tridimensionnelles des modèles.
- **Support des Systèmes Stochastiques et/ou Contextuels :** Amélioration de l'interpréteur pour gérer la variabilité et le contexte dans les règles de L-systèmes.

1.2 Développement du Projet

Nous avons commencé notre projet par une étape de recherche sur les L-systèmes, en utilisant des sources fiables comme Wikipedia [1] et le livre réalisé par Astrid Lindenmayer, *The Algorithmic Beauty of Plants* [2]. Cette étape nous a permis d'obtenir une compréhension approfondie des systèmes de réécriture, ce qui a été crucial pour développer notre propre moteur de réécriture. Ce moteur génère des séquences de symboles à partir d'un symbole initial, en suivant des règles de réécriture spécifiques sur un nombre déterminé d'itérations. Pour plus de détails sur le moteur de réécriture, voir la section 4.1.

Voici un résumé des composants principaux que nous avons développés :

1. **Moteur de Réécriture** : Permet de générer des séquences de symboles, utilisant un symbole initial et des règles spécifiées. Ce moteur est crucial pour la création des modèles que nous visualisons ensuite en 2D et 3D.
2. **Transformation Visuelle 2D** : Nous avons utilisé *Graphics2D*, un outil de Java pour le dessin graphique, pour convertir les séquences en visualisations 2D. Les dessins sont réalisés sur un *JPanel* via la méthode *PaintComponent* , offrant une zone de dessin dans l'interface utilisateur.
3. **Transformation Visuelle 3D** : Pour les visualisations en trois dimensions, nous avons utilisé la bibliothèque JOGL [3]) (Java Open Graphics Library qui permet d'accéder aux fonctionnalités d'OpenGL. Cela est réalisé à travers *GL2* et un *GLPanel*, qui fonctionne comme une zone de dessin 3D où nous créons des représentations graphiques avancées.

1. L-system
2. The Algorithmic Beauty of Plants
3. JOGL

4. **Interface Utilisateur et Navigation** : Nous avons développé une interface utilisateur qui utilise un *CardLayout* pour naviguer entre différentes vues. Avec des clics sur des boutons, l'utilisateur peut alterner entre un panel d'accueil et un panel de configuration. Ce dernier affiche les zones de dessin 2D et 3D activées par les actions de l'utilisateur.

Chaque composant a été conçu pour fonctionner de manière intégrée, fournissant une expérience utilisateur fluide et interactive tout en exploitant la puissance de la visualisation graphique avancée.

2 Système de Lindenmayer

2.1 Principe et fonctionnement

2.1.1 Qu'est-ce qu'un L-Système ?

Le L-Système [1], inventé en 1968 par un biologiste hongrois du nom de Aristid Lindenmayer, est une forme de grammaire formelle [2] principalement utilisé pour la modélisation de processus de développement et de prolifération de bactéries ou de plantes.

2.2 Fonctionnement des L-systèmes

Les L-systèmes utilisent des ensembles de règles de réécriture pour simuler le développement des organismes. Ils sont particulièrement efficaces pour représenter récursivement des structures fractales naturelles.

2.2.1 Principes de base

Un L-système est constitué des composants suivants :

— Un **alphabet** : un ensemble de symboles utilisés pour construire des chaînes.

— Un **axiome** : une chaîne initiale à partir de laquelle la construction commence.

— Des **règles de production** : règles qui remplacent certains symboles par des séquences de symboles pour former une nouvelle génération.

2.2.2 Exemple simple

Pour illustrer, considérons un L-système simple avec les spécifications suivantes :

1. Le système de Lindenmayer
2. Une grammaire formelle est un système composé de symboles et de règles utilisé pour générer des séquences de symboles ou des structures, souvent utilisé dans un contexte linguistique, informatique et biologique.

Composant	Détail
Alphabet	$\{A, B\}$
Axiome (Initiation)	A
Règles de production	$A \to AB$ $B \to A$
Génération 0	A
Génération 1	AB
Génération 2	ABA
Génération 3	$ABAAB$

3 Structure du Projet

Ce projet a été développé en suivant l'architecture du modèle MVC (Modèle-Vue-Contrôleur) et en utilisant le pattern Observer pour gérer les mises à jour des données suite aux interactions de l'utilisateur. Ces choix d'architecture permettent de maintenir une séparation claire des préoccupations et de faciliter la gestion des états de l'application.

3.1 Architecture MVC et Pattern Observer

Le modèle MVC divise l'application en trois composants principaux :

Modèle Le cœur fonctionnel de l'application qui gère les données, la logique et les règles du système.

Vue 5.2 L'interface utilisateur qui affiche les données à l'utilisateur et envoie les interactions utilisateur au contrôleur.

Contrôleur Agit comme un intermédiaire entre le modèle et la vue, en gérant le flux d'informations entre eux et mettant à jour la vue en conséquence.

Le pattern Observer est utilisé pour implémenter les notifications automatiques et les mises à jour entre le modèle et la vue lorsque l'état du modèle change. Cela aide à réduire le couplage entre les composants de l'application et améliore la réactivité de l'interface utilisateur.

3.2 Structure du Répertoire src

La structure principale du code source se trouve dans le dossier `src`, organisé comme suit :

```
src
 controle
    Controleur.java
    Mode.java
 mainPackage
    MainConsole.java
    MainGUI.java
 model
    Interpretation.java
    Lsysteme.java
    Regle.java
```

```
    Tortue2D.java
    Tortue3D.java
    Tortue.java
 util
    AbstractModeleEcoutable.java
    EcouteurModele.java
    ModeleEcoutable.java
 view
     AccueilPanel.java
     ConfigurationPanel.java
     MainFrame.java
     Panels.java
     TypeAffichage.java
```

Chaque dossier représente un composant de l'architecture MVC, et chaque composant joue un rôle spécifique dans l'application :

- **Le dossier controle** contient la classe `Controleur` qui réagit aux actions de l'utilisateur, comme les clics sur les boutons. Ces fonctions orchestrent les interactions entre la vue et le modèle, en déclenchant des réponses appropriées aux entrées utilisateur.
- **Le dossier mainPackage** contient les points d'entrée de l'application, avec deux classes principales :
  - `MainConsole` pour exécuter l'application en mode console,
  - `MainGUI` qui lance l'interface graphique de l'application. Ces classes initialisent l'environnement nécessaire et démarrer les interactions utilisateur selon le mode choisi.
- **Le dossier model** rassemble toutes les classes du Modèle, y compris la logique des L-systèmes et les objets `Tortue` pour le rendu graphique. Ce dossier constitue le cœur fonctionnel de l'application, gérant les données et la logique de transformation.
  - **Interface Tortue** : Cette interface définit les méthodes essentielles pour dessiner en utilisant une séquence de caractères. Elle est implémentée par deux classes principales :
    - `Tortue2D` : Gère le dessin des motifs L-systèmes dans un environnement bidimensionnel.
    - `Tortue3D` : Étend les capacités de dessin aux environnements tridimensionnels. Pour réaliser cela, la classe utilise la bibliothèque JOML [1]) (Java OpenGL Math Library), qui fournit des outils mathématiques comme `Vecteur3f` et `Quaternionf` pour manipuler les coordonnées et les orientations dans un espace 3D, facilitant ainsi le rendu de scènes complexes.
- **Le dossier util** inclut des utilitaires pour implémenter le pattern Observer. Ces outils facilitent la communication entre les différentes parties de l'application en permettant aux composants de rester informés des changements d'état.
- **Le dossier view**5.2 comprend les classes de la Vue, qui sont responsables de toute l'interface utilisateur. Cette partie de l'architecture gère la présentation et les interactions directes avec l'utilisateur. Les composants clés de ce dossier incluent :
  - **AccueilPanel**5.2.1 : Ce conteneur sert de panneau d'accueil pour l'application. Il propose une interface initiale où les utilisateurs peuvent choisir les options ou démarrer des actions spécifiques à l'application.

1. JOML

— **ConfigurationPanel**5.2.2 : Ce conteneur permet aux utilisateurs de configurer les paramètres nécessaires pour le rendu des L-systèmes. Ils peuvent personnaliser un L-système en spécifiant les paramètres tels que l'axiome, les règles, le nombre d'itérations, et l'angle. Le panel offre également des options pour visualiser des modèles prédéfinis, facilitant ainsi l'accès à des configurations typiques ou exemples.

⇒ Les deux panels sont intégrés dans un `CardLayout` qui est ajouté à la `MainFrame` de l'application. Le `CardLayout` permet de switcher facilement entre l'`AccueilPanel` et le `ConfigurationPanel`, offrant ainsi une expérience utilisateur fluide et intuitive. Ce mécanisme de commutation est essentiel pour maintenir l'interface utilisateur organisée et réactive, adaptant l'affichage aux actions et aux besoins des utilisateurs.

Cette structure organisée aide à maintenir une séparation claire des préoccupations et favorise une évolution et une maintenance plus aisées du code.

3.3 Diagramme de Classe

Pour une meilleure compréhension visuelle de la structure et des relations entre les composants, un diagramme de classe est présenté à la fin de cette section. Ce diagramme illustre les dépendances et les interactions entre les classes du modèle, du contrôleur et de la vue.

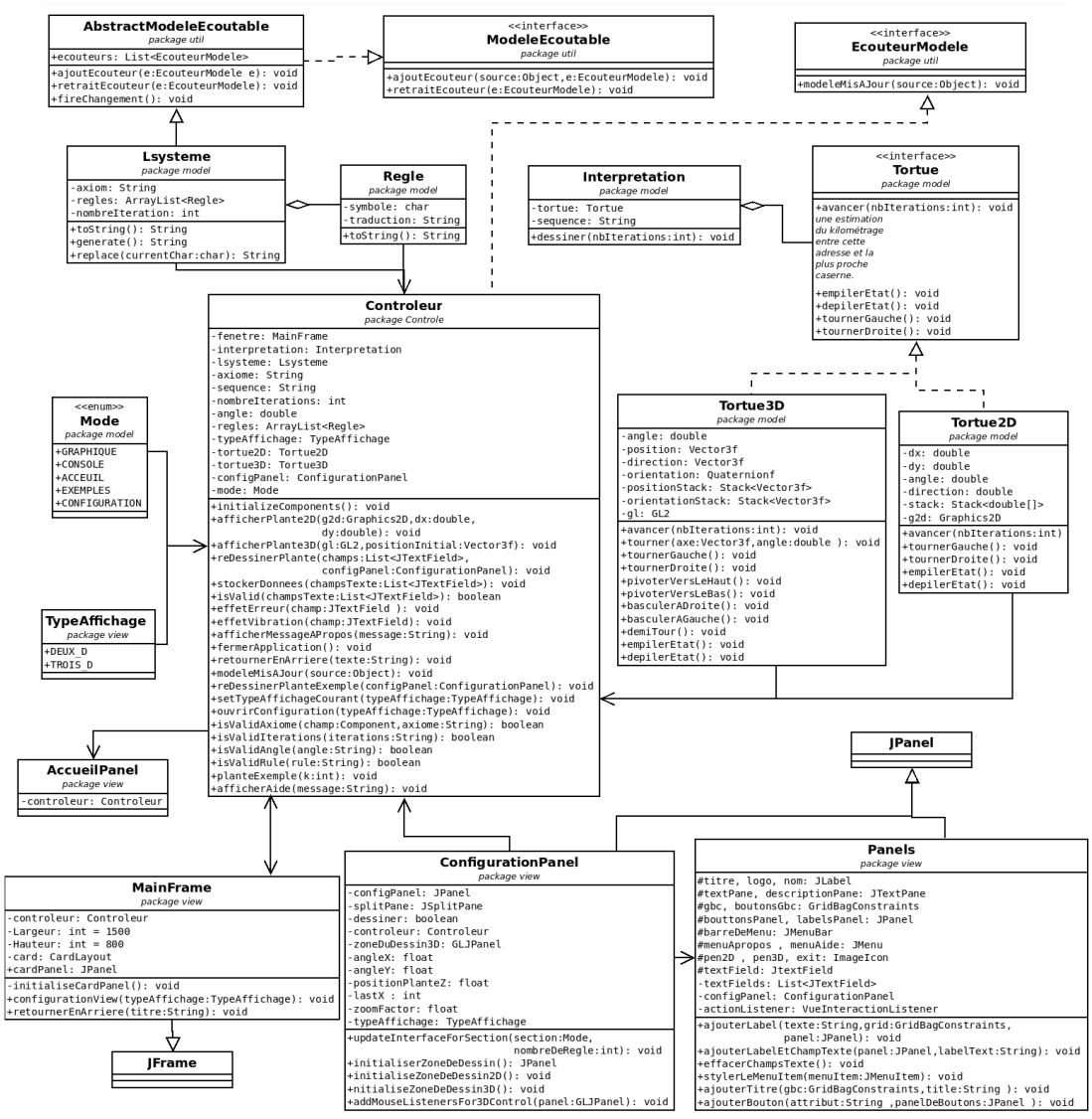

FIGURE 3.1 – Diagramme de classe illustrant l'architecture MVC et les interactions entre composants

4 Éléments Techniques

Ce chapitre détaille le moteur de réécriture utilisé pour générer des séquences basées sur les L-systèmes ainsi que l'intégration du pattern Observer qui facilite la communication entre le modèle et la vue. Nous explorerons spécifiquement les méthodes clés qui composent le cœur logique de l'application et comment les modifications du modèle sont propagées au contrôleur et à la vue.

4.1 Moteur de Réécriture

Le moteur de réécriture est essentiel pour transformer les axiomes selon des règles prédéfinies. Il est implémenté principalement à travers les méthodes dans la classe `Lsysteme`.

4.1.1 La méthode replace

La méthode `replace` remplace chaque caractère par sa traduction appropriée en suivant les règles définies. Si aucune règle n'est applicable, le caractère d'origine est retourné.

Algorithme 1 : Méthode Replace

Entrées : Un caractère *currentChar*
Sortie : La chaîne résultante après remplacement

```
1  Function replace(currentChar) :
2      remplacement ← new StringBuilder()
3      pour regle ∈ this.regles faire
4          si currentChar == regle.getVariable() and
             !regle.getTraduction().isEmpty() alors
5              remplacement.append(regle.getTraduction())
6              retourner remplacement.toString()
7          fin
8      fin
9      remplacement.append(currentChar)
10     retourner remplacement.toString()
```

4.1.2 La méthode generate

La méthode `generate` utilise `replace` pour générer une séquence finale à partir d'un axiome et des règles de production à travers un nombre spécifié d'itérations. Elle notifie également les observateurs après chaque modification.

Algorithme 2 : Méthode Generate

Entrées : Nombre d'itérations *nombreIteration*
Sortie : L'axiome final après application des itérations

```
Function generate() :
    pour i ← 0 à nombreIteration faire
        nouveauAxiome ← new StringBuilder()
        pour chaque caractere ∈ this.axiom.toCharArray() faire
            nouveauAxiome.append(replace(caractere))
        fin
        this.axiom ← nouveauAxiome.toString()
    fin
    fireChangement()
    retourner this.axiom
```

4.2 Intégration du Pattern Observer

La méthode **fireChangement** est cruciale pour la mise en œuvre du pattern Observer. Lorsqu'un changement est détecté, cette méthode est appelée pour notifier tous les observateurs enregistrés, y compris le contrôleur qui réagit en mettant à jour la vue.

4.2.1 Réaction du Contrôleur

La réaction du contrôleur est gérée par **modeleMisAJour**, qui met à jour la vue en fonction des changements dans le modèle. Cette méthode spécifique active la mise à jour du dessin dans le panneau de configuration. Voici le code de la méthode **modeleMisAJour** :

```java
@Override
public void modeleMisAJour(Object source) {
    // Verifie que la source est l'instance attendue
    if (source instanceof Lsysteme) {
        // Active le dessin dans le panneau de configuration
        this.configPanel.setDessiner(true, this.typeAffichage
            );
    }
}
```

5 Exécution et Utilisation du Projet

Ce chapitre décrit comment démarrer et interagir avec le système de L-système développé. Il guide l'utilisateur à travers les étapes nécessaires pour exécuter le projet à partir du répertoire `scripts` et détaille les différents modes d'utilisation offerts par l'interface graphique.

5.1 Démarrage du Projet

Pour exécuter le projet, il est nécessaire de naviguer vers le répertoire contenant le script de démarrage, nommé `script.sh`. Suivez les étapes suivantes pour démarrer correctement l'application :

1. Ouvrez un terminal et déplacez-vous dans le répertoire `scripts` où se trouve le fichier`script.sh`.
2. Avant d'exécuter le script, assurez-vous qu'il dispose des permissions nécessaires. Tapez la commande suivante dans le terminal :
   ```
   chmod +x script.sh
   ```
3. Une fois les permissions ajustées, lancez le script en entrant :
   ```
   ./script.sh
   ```

5.1.1 Menu Principal

À l'exécution, le script affiche un menu principal offrant plusieurs options :

— **1 Mode Console** : Permet d'exécuter l'application en mode console. Dans ce mode, l'utilisateur peut uniquement personnaliser son modèle en saisissant les paramètres via la console. Après la saisie des paramètres, l'affichage du modèle de plante se fait graphiquement, illustrant ainsi les résultats de la configuration spécifiée.

— **2 Mode Graphique** : Lance l'interface graphique de l'application.

— **3 Mode Graphique** : Lance l'interface graphique de l'application.

— **4 Voir la documentation** : Permet de consulter la documentation générée pour ce projet dans votre navigateur par défaut.

— **5 Quitter** : Ferme l'application.

5.2 Mode Graphique

Le mode graphique offre une interface utilisateur riche permettant une interaction visuelle avec le système de L-système.

5.2.1 Accueil Panel

Dans le `AccueilPanel`, l'utilisateur est accueilli par deux boutons :

— **Générer2D** : Permet de passer au `ConfigurationPanel` pour générer des figures en 2D.

— **Générer3D** : Similairement, ce bouton permet de passer à la configuration pour la génération de figures en 3D.

5.2.2 Configuration Panel

Le `ConfigurationPanel` contient un `NavigationPanel` qui offre des options supplémentaires :

— **Personnaliser Modèle** : Permet à l'utilisateur de saisir ou de modifier les paramètres du modèle L-système.

— **Voir Modèles Prêts** : Affiche des modèles prédéfinis sous forme de boutons que l'utilisateur peut visualiser et utiliser en cliquant dessus.

5.2.3 Illustrations des Interfaces

Ci-dessous, vous trouverez des illustrations visuelles de l'`AccueilPanel` et du `ConfigurationPanel`, qui montrent concrètement l'interface utilisateur décrite précédemment. Ces images permettent de visualiser les interactions possibles ainsi que l'aspect général des interfaces.

FIGURE 5.1 – Vue de l'Accueil Panel

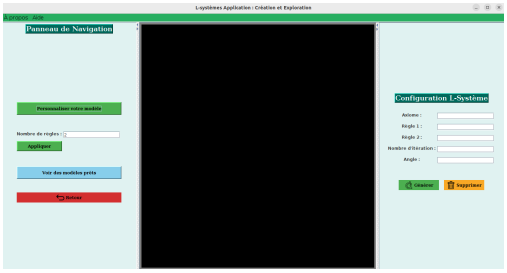

FIGURE 5.2 – Personnalisation du modèle

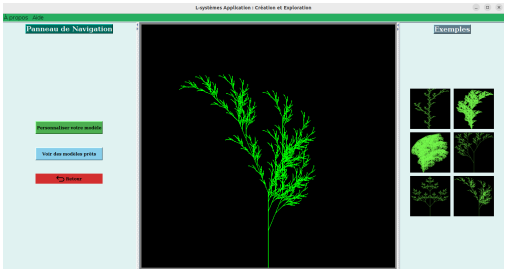

FIGURE 5.3 – Visualisation de modèles prêts

6 Tests Unitaires et Validation du Projet

Ce chapitre détaille les tests unitaires conçus pour vérifier la fiabilité et la robustesse des différentes composantes de notre application, notamment les classes du modèle et de l'interface utilisateur. Ces tests sont essentiels pour assurer que toutes les fonctionnalités répondent correctement aux exigences spécifiées et fonctionnent comme prévu dans divers scénarios d'utilisation.

6.1 Tests des Méthodes du Modèle

Les classes du modèle, qui incluent la gestion des L-systèmes et des tortues graphiques, ont été rigoureusement testées pour garantir leur correct fonctionnement. Voici les principales méthodes testées :

— **Tests de la classe Lsysteme** : Ces tests vérifient que les méthodes `replace` et `generate` fonctionnent correctement, en s'assurant que les caractères sont remplacés selon les règles définies et que la génération de séquences se déroule comme

attendu.

— **Tests de la classe Tortue2D** et **Tortue3D** : Ces tests s'assurent que les tortues peuvent correctement avancer, tourner, et manipuler leur état graphique en fonction des commandes reçues, avec un focus particulier sur les interactions complexes comme le pivotement et le basculement dans l'espace 3D.

6.2 Tests de Stress sur l'Application

Un test de stress a également été réalisé sur la classe `MainConsole` pour évaluer la performance et la stabilité de l'application lors de la manipulation de grands ensembles de données et de paramètres complexes en mode console. Le test consiste à exécuter le programme avec des paramètres de grande taille, mesurant ainsi le temps nécessaire pour compléter le processus sans erreurs.

6.3 Résultats des Tests

Les résultats des tests unitaires ont confirmé que les composantes du système se comportent conformément aux attentes, avec une précision et une fiabilité élevées dans toutes les opérations testées. Le test de stress, quant à lui, a démontré que l'application peut gérer efficacement des situations à haute charge sans dégradation significative des performances.

6.4 Possibles améliorations

— Étendre le langage des L-systèmes pour inclure des variantes stochastiques et contextuelles, permettant une modélisation plus riche et adaptée.

— Optimiser la complexité algorithmique du moteur de réécriture en utilisant des structures de données plus efficaces telles que `HashMap`, `LinkedList`, et des itérateurs pour améliorer les performances et la gestion de la mémoire.

— Améliorer le réalisme de la modélisation 3D pour produire des visualisations plus détaillées et esthétiquement plaisantes.

Conclusion

Ce projet nous a permis d'approfondir notre compréhension des L-systèmes et de leur application dans la modélisation végétale. Nous avons développé une application qui simule des structures naturelles avec précision et esthétique, et envisageons des améliorations futures pour augmenter sa performance et sa réalisme.

Nous remercions sincèrement notre encadrant, M. Antoine Boiteau, pour son soutien et ses conseils précieux qui ont été essentiels à la réussite de notre travail.

Nous envisageons de continuer à améliorer l'application, en espérant qu'elle servira utilement dans le domaine de la visualisation scientifique et éducative.

7 Sources

- The Algorithmic Beauty of Plants :http://algorithmicbotany.org/papers/abop/abop.pdf
- Wikipedia L-Système (FR) : https://fr.wikipedia.org/wiki/L-Système
- Developpez.com - Tutoriel Swing : https://baptiste-wicht.developpez.com/tutoriels/java/swing/debutant
- Java doc - Swing https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html
- JOGL : https://jogamp.org/jogl/www/
- JUnit : https://junit.org/junit4/javadoc/latest/
- JOML : https://github.com/JOML-CI/JOML