Université de Caen Normandie

Rapport

Interpréteur de systèmes de Lindenmeyer

Étudiants :
Celina Bedjou
Wassim Djeha
Louheb Kaced
Yani Belkacemi

Enseignants :
Grégory Bonnet
Mihail STOJANOVSKI

Jury :
Mihail STOJANOVSKI
Alexis Lechervy

26 avril 2023

Table des matières

1 Introduction

1.1 Description générale du projet

Les systèmes de Lindenmeyer, ou L-systèmes, sont des outils mathématiques qui permettent de modéliser et de générer des structures complexes, telles que des arbres, des buissons ou des plantes, en utilisant des règles de réécriture. Ces systèmes sont largement utilisés dans les domaines du jeu vidéo, de l'animation et de la modélisation graphique pour créer des modèles végétaux réalistes et esthétiquement plaisants.

2 Objectifs du projet

L'objectif de ce projet est de concevoir et de développer un interpréteur de L-système qui prendra en entrée des règles de réécriture et produira une représentation visuelle en 2D (ou éventuellement en 3D) de l'objet généré par la simulation du système. Pour cela, plusieurs étapes devront être réalisées, notamment la conception d'un parser pour analyser les règles de réécriture, la mise en place d'un moteur de réécriture pour appliquer les règles successivement, et enfin la mise en œuvre d'un moteur de rendu graphique pour visualiser le résultat final.

Le projet pourra être étendu pour prendre en compte les L-systèmes stochastiques, où les règles sont appliquées avec une probabilité donnée, ainsi que les L-systèmes contextuels, où les règles sont appliquées en fonction des symboles environnants. La réalisation de cet interpréteur de L-système permettra ainsi de créer des modèles végétaux réalistes et diversifiés, offrant de nombreuses possibilités pour les applications dans le domaine du jeu vidéo, de l'animation et de la modélisation graphique.

2.1 Description des grandes étapes et fonctionnalités à implémenter

2.1.1 Partie 0 : réalisation de la version 1 diagramme UML

LA Figure 1 représente le tous premier diagramme UML réaliser par notre groupe , il a été modifier réguliarement.

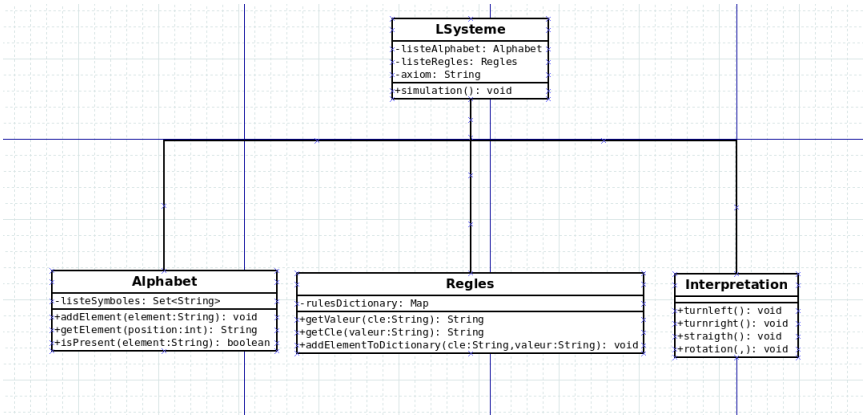

FIGURE 1 – Premier diagramme UML réaliser

2.1.2 Partie 1 : Définition d'un langage pour construire un système de Lindenmeyer classique

1. Description de la syntaxe et de la grammaire du langage utilisé pour représenter les règles de réécriture d'un L-système.Exemple **A,B,C.....Z** , **+,-,*,/,[,]** ;
2. Présentation des éléments de base d'un L-système, tels que les symboles, les règles de réécriture, les axiomes et les paramètres.
3. Ecrire les contrats de chaque fonction avant l'implémentation .
4. Implémentation d'un parser pour analyser les règles de réécriture d'un L-système en entrée et les représenter dans une structure de données adaptée dans le code .Le parser utilisé dans notre code est le **HashMap**(dictionnaire), exemple de régle avec **HashMap** :

```
{
"A" : "AB",
 "B" : "A"
}
```

2.1.3 Partie 2 : Implémentation d'un interpréteur pour une visualisation 2D

1. Explication du processus de réécriture itératif qui génère une séquence de symboles à partir de l'axiome initial en appliquant les règles de réécriture successivement.

   Le moteur représente la fonction "simulation(....)" dans notre code ,tels que cette fonction est une fonction hypothétique déclarer dans le package model/classe AbstractLsystem . Cette fonction représente le moteur de réécriture d'un L-système, qui prend en entrée un axiome initial (souvent sous forme de chaîne de caractères OU meme caractére), les règles de réécriture définies pour ce L-système la liste **HashMap**, ainsi que le nombre d'itérations souhaitées pour appliquer ces règles.
2. Interprétation des symboles générés comme des instructions de dessin pour tracer la structure arborescente sur un canvas 2D.En utilisant la fonction "rotation2D(....)" .
3. Présentation d'exemples de structures végétales générées à partir du L-système en utilisant des fichiers **.txt** contenant les coordonnées des points [x y] a dessiner sachant que les points sont calculer generer a partir des fonctions implémenter dans model comme "calculCoordinate(...)" puis copier apartir du terminal et coller dans differents fichiers textes pour differents Lsysteme .

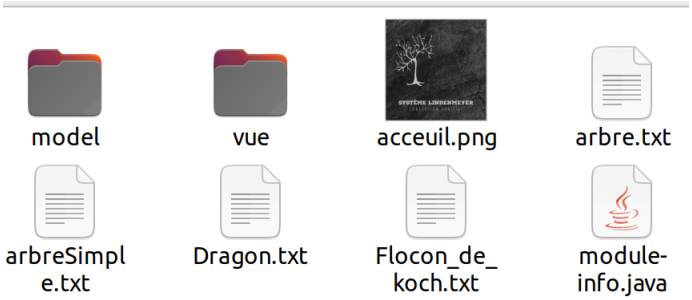

FIGURE 2 – Exemple du format des fichiers

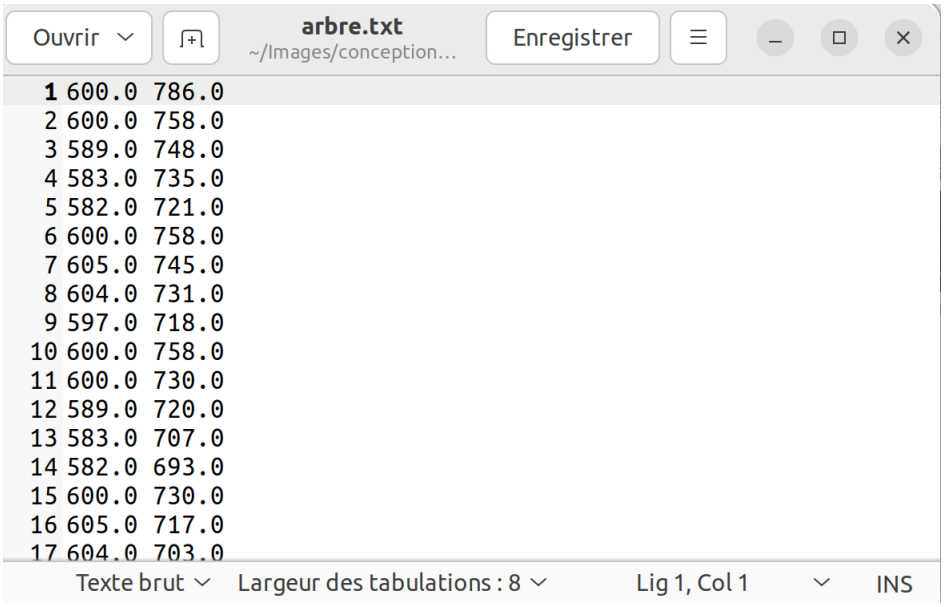

FIGURE 3 – Exemple contenu du fichier arbre.txt qui réalise le dessin d'un arbre .

Cette partie sera détaillier dans les prochaines sections.4.1.4

2.1.4 Partie 3 : Implémentation d'un interpréteur pour une visualisation 3D

1. Extension du moteur de réécriture pour générer des structures tridimensionnelles en utilisant des règles de réécriture spécifiques.En utilisant la fonction "rotation3D(....)"
2. Explication du processus de transformation des symboles générés en un modèle 3D en utilisant des primitives géométriques comme des cylindres, des sphères, etc.
3. Utilisation d'une bibliothèque de rendu 3D pour afficher et manipuler les modèles 3D générés à partir du L-système.Avec la bibliothéque **joagl**.

2.1.5 Partie 4 : Extension du langage et des interpréteurs pour des systèmes stochastiques et/ou contextuels

1. Description des L-systèmes stochastiques8, où les règles de réécriture ont une probabilité d'être appliquées, et des L-systèmes contextuels7, où les règlessont conditionnelles et dépendent du contexte environnant.

2. Extension du langage et du parser pour prendre en compte les règles de réécriture stochastiques et contextuelles, et adaptation du moteur de réécriture en conséquence.
3. Implémentation d'un moteur de réécriture stochastique qui applique les règles de réécriture avec une probabilité donnée pour générer des structures végétales avec une variation aléatoire.
4. Implémentation d'un moteur de réécriture contextuel qui prend en compte le contexte des symboles adjacents pour appliquer les règles de réécriture, permettant ainsi de générer des structures végétales avec des formes plus réalistes et complexes en fonction de l'environnement.

2.1.6 Partie 5 : Ajout de fonctionnalités avancées

1. Ajout de fonctionnalités avancées telles que la gestion des paramètres, des variables, des itérations et des règles conditionnelles pour permettre une plus grande flexibilité dans la génération de structures végétales.gérer par des **try and catch** dans le package "Vue.FenetreLSysteme"
2. Extension du langage et du parser pour prendre en charge ces fonctionnalités avancées et adaptation des moteurs de réécriture en conséquence.
3. Implémentation d'un système de gestion des paramètres pour permettre la modification dynamique des valeurs des paramètres du L-système pendant la génération des structures végétales.
4. Mise en œuvre d'un système d'itérations pour permettre la répétition d'un ensemble de règles de réécriture un certain nombre de fois, ce qui permet de générer des structures végétales avec des niveaux de détail différents.
5. Implémentation de règles conditionnelles pour permettre l'application de règles de réécriture en fonction de conditions spécifiques, offrant ainsi plus de contrôle sur la génération des structures végétales en fonction de certains critères.

2.2 Organisation et répartition des tâches

La répartition des tâches pour le projet de conception s'est organisée en deux binômes, l'un travaillant sur la partie console du projet et l'autre sur la partie interface utilisateur. Cependant,je tiens a vous signaler que le nombre de commits sur SVN peut sembler faible pour chaque membre du groupe, car la plupart du temps, le travail a été réalisé en collaboration et en binôme, ce qui implique que les contributions individuelles ne sont pas nécessairement reflétées uniquement par le nombre de commits.

3 Comment lancer le projet ?

3.1 Creation du fichier .jar

1. Étape 1 : Créer un projet Java dans Eclipse
   — Ouvrez Eclipse et créez un nouveau projet Java ou ouvrez un projet Java existant dans lequel vous souhaitez créer un fichier .jar exécutable.

2. Étape 2 : Configurer le fichier manifeste
   — Dans le projet Eclipse, créez un dossier nommé "META-INF" à la racine du projet, s'il n'existe pas déjà.
   — À l'intérieur du dossier "META-INF", créez un fichier appelé "MANIFEST.MF" (assurez-vous de respecter la casse dans le nom du fichier et de l'enregistrer en tant que fichier texte).
   — Ouvrez le fichier "MANIFEST.MF" dans un éditeur de texte.
   — Ajoutez les informations nécessaires au manifeste. Par exemple, vous pouvez inclure les lignes suivantes : makefile

     Manifest-Version : 1.0 Main-Class : package.MainClass

   — Assurez-vous de remplacer "package.MainClass" par le nom de la classe principale de votre application Java.
3. Étape 3 : Exporter le fichier .jar
   — Cliquez avec le bouton droit sur le projet dans la vue "Package Explorer" ou "Project Explorer" d'Eclipse.
   — Sélectionnez "Export" dans le menu contextuel.
   — Dans la boîte de dialogue "Export", développez la catégorie "Java" et sélectionnez "Runnable JAR file".
   — Cliquez sur le bouton "Next".
   — Sélectionnez le projet et la classe principale à utiliser comme point d'entrée de l'application.
   — Choisissez un emplacement et un nom de fichier pour le fichier .jar exporté.
   — Sélectionnez les options d'exportation appropriées (telles que l'exportation des bibliothèques externes dans le fichier .jar ou non).
   — Cliquez sur le bouton "Finish" pour exporter le fichier .jar.

3.2 Lancement du projet

1. On se position dans le dossier l-systems-renderer/dist
2. On lance le terminal
3. On tape la commande **java -jar run.jar**

4 Conception du projet

4.1 Architecture du projet

L'architecture du projet est basée sur une organisation classique d'un projet Java, avec une séparation claire entre les fichiers sources et les fichiers binaires, ainsi qu'une séparation entre la partie interface graphique et la partie console .

Le projet est organisé en package distincts pour les différentes parties du code, comme le package model, le package vue et les tests unitaires . Cette organisation permet de séparer clairement le code et facilite la maintenance, la compréhension et l'évolution du code.

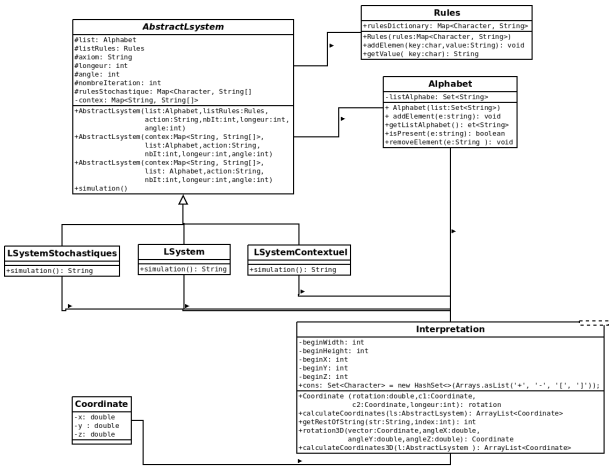

FIGURE 4 – diagramme de classe de la partie console

4.1.1 Console

Les classes du package "model" sont chargées de gérer les différents aspects des L-systèmes, notamment les alphabets, les coordonnées, les règles et leur interprétation. Elles sont utilisées pour interpréter les règles et les alphabets, afin de les transformer en éléments visuels qui peuvent être traduits ultérieurement en une représentation visuelle.

4.1.2 Interface graphique

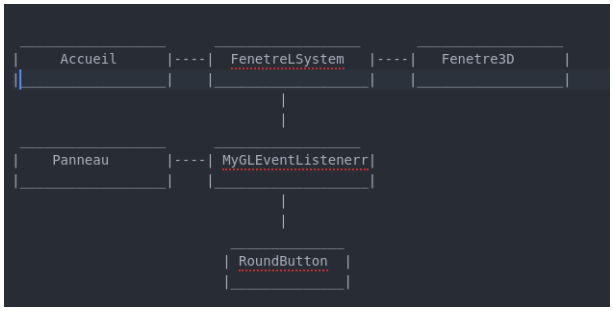

FIGURE 5 – diagramme de classe de la partie interface graphique

La partie interface graphique du programme est regroupée dans le package "vue", qui contient des classes Java dédiées à la gestion de l'affichage et de l'interaction avec l'utilisateur, telles que "Accueil", "Fenetre3D", "FenetreLSystem", "Panneau" et "RoundButton". Cette séparation permet de gérer l'interface graphique de manière indépendante de la console, et son objectif est de traduire les données générées par la partie console en éléments visuels, tout en gérant les interactions de l'utilisateur avec le programme, en 2D et en 3D.

4.1.3 Tests unitaires

Le dossier "tests unitaires" contient généralement les tests destinés à vérifier le bon fonctionnement du code et à détecter d'éventuelles erreurs. Dans notre cas, nous avons

tenté de tester la fonction "rotation(...)" en raison de problèmes rencontrés lors de l'importation de la bibliothèque JUnit pour utiliser la méthode "assertEquals". Malheureusement, ces problèmes n'ont pas été résolus avec succès jusqu'à présent, ce qui a empêché les tests des autres fonctions du projet.

4.1.4 Ressources

Les fichiers texte et images utilisés dans le projet sont situés dans le répertoire "src" du projet. Ces fichiers sont utilisés comme ressources par l'application, tels que des éléments graphiques pour l'interface utilisateur ou des exemples de L-systèmes générés.

Le code dans 4.1.4 montre une fonction appelée "ecrireCoordonneesDansFichier" qui prend en paramètre une liste d'objets "Coordinate" et écrit les coordonnées contenues dans cette liste dans un fichier texte appelé "arbre.txt" situé dans le répertoire "src".

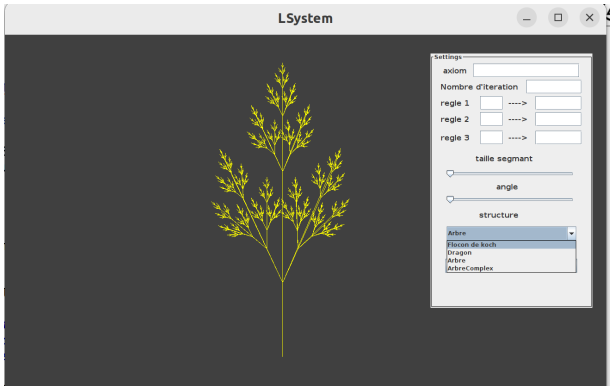

FIGURE 6 – exemple generation arbre avec coordonnées stockées dans un fichier.

Voici comment cette fonction travaille en détail :

1. Elle utilise un bloc try-catch pour gérer les exceptions, en particulier les exceptions de type "FileNotFoundException" qui peuvent être levées si le fichier "arbre.txt" n'est pas trouvé.
2. Elle crée un objet "PrintWriter" qui est utilisé pour écrire dans un fichier texte. L'objet "PrintWriter" prend en paramètre un nouvel objet "File" qui représente le fichier "arbre.txt".
3. Elle utilise une boucle "for-each" pour parcourir chaque objet "Coordinate" dans la liste "coordonnees".
4. Pour chaque objet "Coordinate", elle utilise les méthodes "getX()" et "getY()" pour obtenir les valeurs des coordonnées en X et Y respectivement.
5. Elle utilise la méthode "println()" de l'objet "PrintWriter" pour écrire les coordonnées en X et Y séparées par un espace dans une nouvelle ligne du fichier "arbre.txt".
6. Enfin, elle ferme l'objet "PrintWriter" en utilisant la méthode "close()" pour libérer les ressources et enregistrer les modifications dans le fichier "arbre.txt".

Si le fichier "arbre.txt" n'est pas trouvé, l'exception "FileNotFoundException" sera attrapée et une trace de la pile d'erreur sera imprimée à l'aide de la méthode "printStackTrace()" de l'objet d'exception "e".

```
public void ecrireCoordonneesDansFichier(List<Coordinate> coordonnees) {
    try {
        PrintWriter writer = new PrintWriter(new File("src/arbre.txt"));
        for (Coordinate coordonnee : coordonnees) {
            writer.println(coordonnee.getX() + "␣" + coordonnee.getY());
        }
        writer.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

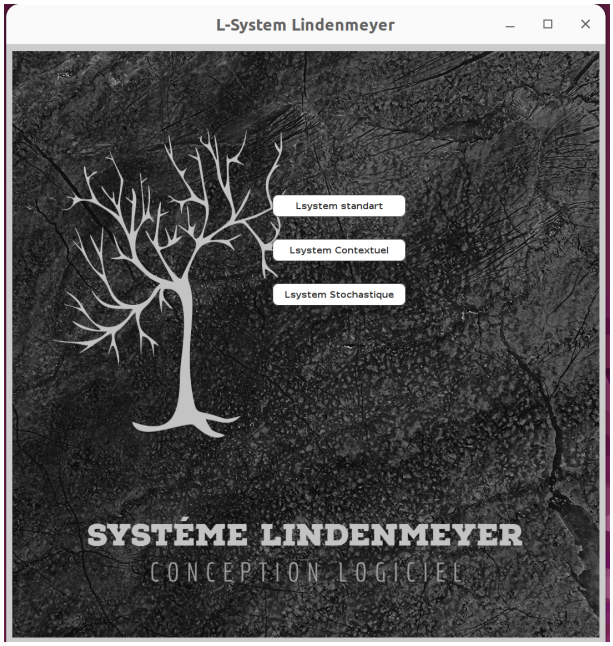

FIGURE 7 – exemple d'utilisation de l'image dans la fenetre du projet.

4.2 L'Organisation des Fichiers/Dossiers

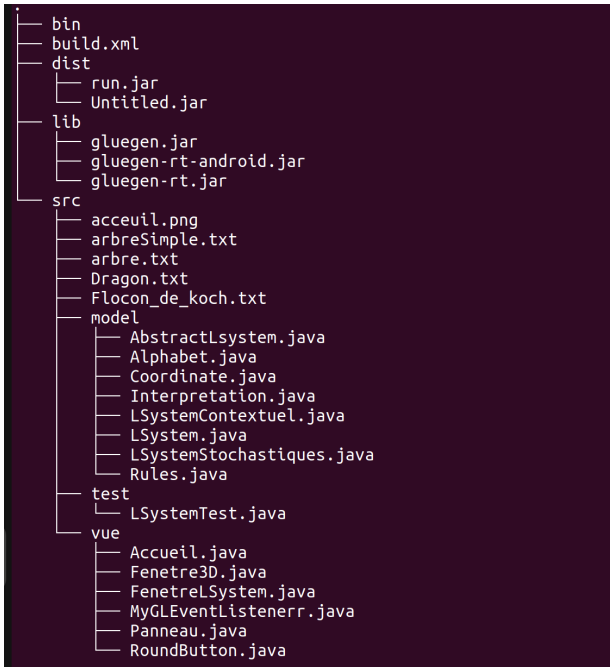

FIGURE 8 – Organisation du projet Fichiers/Dossiers

4.2.1 Organisations des dossiers

1. Le dossier bin contient les fichiers compilés du projet, les fichiers générés par la compilation des fichiers sources Java.
2. le dossier lib contient les bibliotéques joagl et gungael.
3. le dossier dist contient l'executable "Untitled.jar" pour lancer le projet.[1]
4. Le dossier src contient les fichiers sources Java (.java) du projet, ainsi que les fichiers texte et images mentionnés précédemment :[4.1]
   -Le dossier model contient des fichiers pour différentes classes liées aux L-systèmes.
   -Le dossier vue contient les fichiers pour différentes classes de l'interface utilisateur.
   -Le dossier testeunitaire contient les fichiers sources Java pour les tests unitaires, comme TesteUnitaire.java.

5 Revue de la littérature

5.1 Revue des travaux de recherche existants

Avant de commencer notre projet, nous avons effectué une revue des travaux de recherche existants dans le domaine des L-systèmes. Nous avons consulté les ressources mises à disposition par notre professeur, ce qui s'est avéré très utile pour notre projet. Cette recherche nous a permis de découvrir la théorie des L-systèmes développée par Aristid Lindenmayer, un biologiste hongrois dans les années 60. Cette théorie permet de décrire

la croissance des plantes et des organismes biologiques. Nous avons également découvert qu'il existe plusieurs types de L-systèmes qui diffèrent d'un système à un autre, principalement par la manière dont un système donné interprète les règles fournies par l'utilisateur.

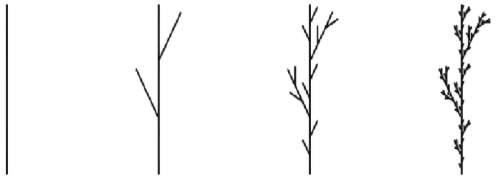

FIGURE 9 – generation d'une plante avec un L-systeme

En approfondissant nos recherches, nous avons découvert qu'il existe plusieurs algorithmes pour la réécriture de chaînes et la génération de graphiques 2D et 3D à partir des L-systèmes. Nous avons également constaté que les L-systèmes sont largement utilisés dans divers domaines tels que les jeux vidéos, les films, la biologie, l'architecture et l'art. Nous avons également identifié les tendances actuelles dans la recherche sur les L-systèmes, notamment les L-systèmes paramétriques, les L-systèmes interactifs et les L-systèmes évolutifs. Ces types de L-systèmes sont plus complexes et visent à créer des structures plus grandes et plus détaillées.

voici quelque exemple :

— **L-systèmes déterministes :** Les L-systèmes déterministes sont les plus simples et les plus courants. Ils utilisent un ensemble de règles de production fixes et déterministes pour remplacer les symboles de l'axiome initial. La croissance de la structure est entièrement déterminée par les règles de production et l'état initial.
Exemple : Le flocon de von Koch

Axiome : F–F–F
Règles : F -> F+F–F+F
Angle : 60 degrés

— **L-systèmes stochastiques :**
Les L-systèmes stochastiques introduisent une composante aléatoire dans les règles de production. Chaque symbole peut avoir plusieurs règles de production, chacune avec une certaine probabilité d'être choisie. Cela permet de générer des structures plus variées et plus naturelles.
Exemple : Plante stochastique

Axiome : X
Règles : X -> F-[[X]+X]+F[+FX]-X avec probabilité 0.5
X -> F+[[X]-X]-F[-FX]+X avec probabilité 0.5
F -> FF
Angle : 25 degrés

— **L-systèmes contextuels :**
Les L-systèmes contextuels tiennent compte du contexte des symboles pour déterminer les règles de production à appliquer. Les règles de production ont des préconditions qui doivent être remplies pour qu'elles puissent être appliquées. Cela permet de créer des structures plus complexes et plus réalistes.

Exemple : L-système contextuel simple

Axiome : A[B]C Règles : B -> X dans le contexte A _ C (B est remplacé par X si B est entre A et C) Angle : Non applicable

5.2 Présentation des concepts clés des L-systèmes

- Alphabet : Un L-système utilise un ensemble fini de symboles, appelé alphabet, pour représenter les éléments de base de la structure à modéliser ,les alphabets sont représenter dans notre projet sous forme d'un set ou une liste de "A,.....,Z" dans la classe "Alphabet.java".
- Axiome : L'axiome est la chaîne initiale à partir de laquelle le L-système commence son processus de réécriture. Il représente généralement l'état initial de la structure à modéliser , représenter par un string dans notre projet.
- Règles de production : Les règles de production définissent comment les symboles de l'axiome ou des chaînes dérivées sont remplacés par d'autres symboles ou séquences de symboles. Ces règles déterminent la manière dont la structure évolue au fil des réécritures successives , ce qu'on appelle le parser représenter par un dictionnaire (cle,valeur) "HashMap" dans la classe "Rule.java".[4]
- Processus de réécriture : Le processus de réécriture consiste à appliquer les règles de production à la chaîne actuelle de manière répétée, en remplaçant chaque symbole par la séquence de symboles correspondante définie par les règles de production. Chaque application des règles de production est appelée une "itération" ou "génération".

6 Les L-Systems Standard

Un L-système standard est un type de L-système qui utilise un ensemble de règles de production fixes et déterministes pour remplacer les symboles de l'axiome initial. La croissance de la structure est entièrement déterminée par les règles de production et l'état initial, sans aucune composante aléatoire ou contextuelle. dans le cas de notre projet on creer une class qui herite de la la classe AbstractLsystem. cette classe prend en argument une liste d'alphabet et une liste des regles , un axiome et un nombre d'iteration. avec la methode simulation elle genere une chaine de symboles selon le nombre d'iteration en utilisant une simple boucle. et l'interieur de cette boucle elle repete le processus de remplacement de chaque carctere de la chaine generée avec les regles de production associees.

Voici la fonctions implémenter pour générer la chaine finale associer au LSystem Standard :

```
public String simulation(int nombreIteration) {
                StringBuilder chaine = new StringBuilder(axiom);
                List<Coordinate> listC = new ArrayList<>();
                double x = 100;
                double y = 100;
                //listC.add(new Coordinate(x, y));

                for (int i = 0; i < nombreIteration; i++) {
                        StringBuilder newChaine = new StringBuilder();
                        char[] charArray = chaine.toString().toCharArray();
                        for (char c : charArray) {
                                String replacement = listRules.getValue(c);
                                newChaine.append(replacement);
                        }
                        chaine = newChaine;
                        y += longeur;
                        //listC.add(new Coordinate(x, y));
                }
                return chaine.toString();
        }
```

6.1 Explication du code

1. La méthode simulation(int nombreIteration) est utilisée pour générer la chaîne de symboles en utilisant les règles de production et le nombre d'itérations donné.
2. L'axiome initial, la liste d'alphabet et la liste de règles de production sont passés en arguments dans le constructeur de la classe LSysteme.
3. La variable chaine représente la chaîne de symboles générée, et elle est mise à jour à chaque itération de la boucle en remplaçant les symboles selon les règles de production.
4. La variable newChaine est utilisée pour construire la nouvelle chaîne de symboles à partir de l'ancienne chaîne en appliquant les règles de production.
5. Les coordonnées x et y sont utilisées pour représenter la position des symboles générés, mais leur utilisation n'est pas claire dans le code, car elles sont mises à jour sans être utilisées ultérieurement.
6. Code commenter dans les classes .java pour clarifier leur rôle.

7 Les L-Systems Contextuels

Les L-systèmes contextuels sont une extension des L-systèmes de base où les règles de production dépendent non seulement du symbole à remplacer, mais aussi des symboles voisins (contexte). Cela permet de créer des modèles plus complexes et réalistes. la classe LSystemContextuel, qui hérite de la classe AbstractLsystem implémente un L-système contextuel, où les règles de production sont appliquées en fonction du contexte.

Voici la fonctions implémenter pour générer la chaine finale associer au LSystem Contextuels :

```
public String simulation() {
                String resultat = axiom;

                for (int i = 0; i < this.nombreIteration; i++) {
                        StringBuilder sb = new StringBuilder();

                        for (int j = 0; j < resultat.length(); j++) {
                                String actuel = String.valueOf(resultat.charAt(j
   ));
                                String contexte = "";
                                String predecesseurStrict = actuel;

                                if (j > 0) {
                                        contexte += resultat.charAt(j - 1);
                                        predecesseurStrict = resultat.substring(
   j - 1, j + 1);
                                }
                                if (j < resultat.length() - 1) {
                                        contexte += resultat.charAt(j + 1);
                                }

                                String cle = predecesseurStrict + ":" + contexte
   ;
                                String[] production = this.contex.get(cle);


                                if (production == null) {
                                        production = this.contex.get(actuel);
                                }


                                if (production == null) {
                                        sb.append(actuel);
                                } else {
                                        sb.append(String.join("", production));
                                }
                        }

                        resultat = sb.toString();
                }

                return resultat;
        }
```

7.1 Explication du code

La fonction simulation() est la méthode principale de la classe LSystemContextuel qui effectue la simulation de l'L-système contextuel en appliquant les règles de production en fonction du contexte. Voici une explication détaillée de son fonctionnement :

1. resultat est initialisée avec la valeur de l'axiome, qui est la chaîne de départ de la simulation.
2. Une boucle for est utilisée pour itérer sur le nombre d'itérations spécifié dans la variable nombreIteration. Cela permet d'appliquer les règles de production pour chaque itération de la simulation.
3. À chaque itération, une nouvelle instance de StringBuilder appelée sb est créée pour construire la nouvelle chaîne de caractères résultante.
4. Une autre boucle for est utilisée pour parcourir les caractères de la chaîne resultat à chaque itération. Chaque caractère est considéré comme le caractère actuel pour lequel les règles de production sont appliquées.
5. Le contexte du caractère actuel est déterminé en examinant les caractères voisins dans la chaîne resultat. Les caractères précédent et suivant sont utilisés pour former le contexte, et le prédécesseur strict est également déterminé en utilisant une sous-chaîne de la chaîne resultat.
6. Une clé de recherche cle est construite en combinant le prédécesseur strict et le contexte. Cette clé est utilisée pour rechercher dans la structure de données contex (qui est supposée être une map) les règles de production associées.
7. Si aucune règle de production n'est trouvée pour la clé de contexte, une recherche est effectuée uniquement avec le caractère actuel comme clé.
8. Si une règle de production est trouvée pour la clé de contexte, le résultat de la règle de production est ajouté à la nouvelle chaîne de caractères sb.
9. Si aucune règle de production n'est trouvée pour le caractère actuel ou pour la clé de contexte, le caractère actuel est simplement ajouté à la nouvelle chaîne de caractères sb.
10. Une fois que tous les caractères de la chaîne resultat ont été traités, la nouvelle chaîne de caractères sb est mise à jour comme la nouvelle chaîne resultat, prête pour l'itération suivante.
11. Une fois que toutes les itérations sont terminées, la chaîne de résultat finale est retournée comme résultat de la simulation.

8 Les L-Systems Stochastiques

Les L-systèmes stochastiques sont une variante des L-systèmes où plusieurs règles de production peuvent être associées à un même symbole, chacune avec une probabilité associée. Cette approche permet de générer des structures plus variées et naturelles. Pour implémenter cette fonctionnalité dans notre projet, nous avons suivi le même principe que celui utilisé dans un L-système standard. Cependant, lorsqu'il s'agit de remplacer un caractère présent dans la liste des alphabets, nous le remplaçons par une règle de production choisie aléatoirement parmi les règles disponibles pour ce caractère. Cela introduit

de l'aléatoire dans le processus de génération, ce qui conduit à une plus grande diversité dans les structures générées et à un aspect plus naturel dans les résultats obtenus.

Supposons que nous ayons une situation où un événement stochastique se produit avec deux options possibles : A et B. La probabilité de l'événement A se produise est de 0,3 (ou 30%), tandis que la probabilité de l'événement B se produise est de 0,7 (ou 70%).

Mathématiquement, nous pouvons représenter cela de la manière suivante :

$$P(A) = 0,3$$
$$P(B) = 0,7$$

Où $P(A)$ est la probabilité de l'événement A et $P(B)$ est la probabilité de l'événement B.

En utilisant ces probabilités, nous pouvons modéliser la composante stochastique de notre système en utilisant des fonctions de génération de nombres aléatoires pour générer un nombre entre 0 et 1, et en comparant ce nombre avec les probabilités pour déterminer quel événement se produit.

Par exemple, en utilisant du pseudocode :

```
générer un nombre aléatoire entre 0 et 1
si le nombre aléatoire est inférieur à 0,3 :
    l'événement A se produit
sinon :
    l'événement B se produit
```

Cela introduit une probabilité dans notre système, où l'événement A se produira avec une probabilité de 0,3 et l'événement B se produira avec une probabilité de 0,7, rendant le système plus réaliste et imprévisible.

Voici la fonctions implémenter pour générer la chaine finale associer au LSystem stochastique :

```
    public String simulation() {

                StringBuilder chaineCourrante = new StringBuilder(axiom);
                for (int i = 0; i < this.nombreIteration; i++) {
                        StringBuilder next = new StringBuilder();
                        char[] chars = chaineCourrante.toString().toCharArray();
                        for (char c : chars) {
                                if (this.list.isPresent(String.valueOf(c))) {
                                        String[] replacements = this.
    rulesStochastique.get(c);
                                        String replacement = replacements[new
    Random().nextInt(replacements.length)];
                                        next.append(replacement);
                                } else {
                                        next.append(c);
                                }
                        }
                        chaineCourrante = next;
                }

                return chaineCourrante.toString();

        }
```

8.1 Explication du code

La fonction simulation() est la méthode principale de la classe LSystemStochastique qui effectue la simulation de l'L-système stochastique en appliquant les règles de production en fonction du contexte. Voici une explication détaillée de son fonctionnement :

1. La fonction prend en compte un L-système stochastique avec des règles de production associées à des probabilités pour chaque symbole.
2. Elle utilise un StringBuilder pour créer la chaîne résultante de la génération, en commençant par l'axiome initial.
3. Elle itère sur le nombre d'itérations spécifié pour la génération de la chaîne.
4. Pour chaque caractère dans la chaîne courante, elle vérifie s'il est présent dans la liste des alphabets du système à l'aide de la méthode isPresent() de l'objet this.list.
5. Si le caractère est présent dans la liste des alphabets, alors elle récupère les règles de production associées à ce caractère à partir du dictionnaire this.rulesStochastique en utilisant this.rulesStochastique.get(c), où c est le caractère courant.
6. Elle choisit ensuite aléatoirement une règle de production parmi les règles disponibles pour ce caractère en utilisant new Random().nextInt(replacements.length) pour obtenir un index aléatoire et accéder à la règle correspondante dans le tableau replacements.
7. Elle ajoute le résultat de cette règle de production dans le StringBuilder next.

8. Si le caractère n'est pas présent dans la liste des alphabets, elle l'ajoute directement à la chaîne résultante sans effectuer de remplacement.
9. Enfin, elle met à jour la chaîne courante avec la chaîne générée à l'itération en cours et répète le processus pour le nombre d'itérations spécifié.
10. À la fin, elle retourne la chaîne résultante de la génération comme résultat de la fonction.

9 Interactions avec les différentes vues

Dans notre projet, nous avons implémenté deux types d'interactions avec les différentes vues : le Console et l'Interface.

9.1 Terminal

Dans ce projet, un L-système est utilisé pour générer des arbres. Voici comment le projet fonctionne lorsque vous l'exécutez dans une console :

Initialisation : Les classes nécessaires sont initialisés comme l'alphabet, les règles,et un les L-systèmes (déterministes, contextuels et stochastiques), et les paramètres tels que le nombre d'itérations, la longueur et l'angle et l'axiome.

Simulation : Le L-système sélectionné effectue la simulation en appliquant les règles de production sur l'axiome pour un nombre spécifié d'itérations. Cette simulation produit une chaîne de caractères représentant la structure de l'arbre.

Interprétation : La chaîne générée par la simulation est interprétée pour générer une liste de coordonnées. Les caractères de la chaîne sont sont traduit par les fonctions de rotation, de déplacement et de sauvegarde ou restauration de l'état pour obtenir les coordonnées des points formant la structure de l'arbre.

9.2 2D/3D

9.3 pourqoui pas des fenétres 2D/3D qui se chevauchent entre elles ?

l'interface graphique de ce projet est construite en utilisant Java Swing pour créer et organiser les éléments de l'interface utilisateur 2D ,3D. Les interactions de l'utilisateur avec ces éléments déclenchent des événements qui sont gérés par des écouteurs d'événements, lesquels exécutent le code pour générer et afficher les fractales en utilisant l'algorithme L-Système.

La classe principale FenetreLSystem hérite de JFrame, la classe Swing représentant la fenêtre de l'application. Dans le constructeur, divers éléments d'interface utilisateur (JButton, JTextField, JLabel, JComboBox, JPanel) sont initialisés et configurés.
Les éléments sont ajoutés à différents panneaux (par exemple, PanneauManipulation) qui sont ensuite ajoutés à la fenêtre principale avec des gestionnaires de disposition.
Des écouteurs d'événements sont attachés pour détecter les interactions utilisateur et exécuter le code correspondant.
La classe PanneauPrincipale dessine les arbres L-Système en utilisant les cordonnees generer et se met à jour lorsque l'utilisateur interagit avec l'interface. La méthode main initialise l'application en créant une instance de FenetreLSystem.

Nous avons décidé d'utiliser deux fenêtres distinctes pour les modes 2D et 3D de notre projet pour plusieurs raisons. Tout d'abord, les modes 2D et 3D de notre application nécessitent des rendus graphiques différents pour afficher correctement les éléments visuels. Le mode 2D utilise des images et des sprites en deux dimensions, tandis que le mode 3D utilise des modèles 3D en trois dimensions. Pour permettre l'affichage approprié de ces éléments graphiques, nous avons opté pour des fenêtres distinctes pour chaque mode afin de garantir que les éléments graphiques soient rendus de manière appropriée et esthétiquement agréable dans chaque mode.
Dans la figure suivante la fenétre adroite représente la fenétre 2D avec comme L system "Dragon" ,et celle agauche une fenetre 3D avec comme L system axiom :F et Regle :F+F avec une longeur de 5 et un angle de 60.

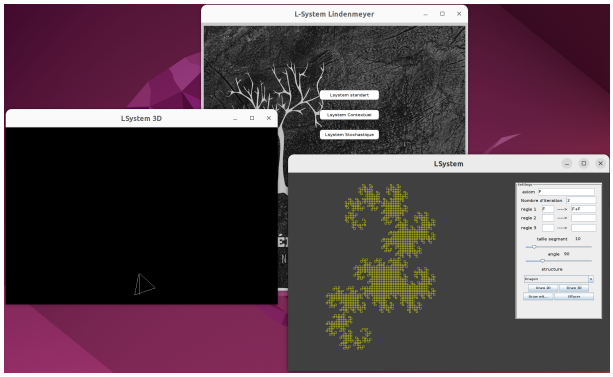

FIGURE 10 – Exemple d'affichage de notre projet 2D/3D

9.4 Comment calculer les coordoonées des points dessiner sur les Fenétres 2D/3D ?

L'une des principales fonctionnalités de notre projet est l'interpréteur, qui est représenté sous la forme de la fonction "calculateCoordinates()" dans model.Interpretation. Cette fonction est utilisée pour calculer les coordonnées des points qui composent la fractale. Elle prend en entrée un L-système et utilise la méthode "simulation()" de l'objet L-système pour obtenir une chaîne de caractères représentant la fractale. Ensuite, elle parcourt cette chaîne de caractères et calcule les coordonnées de chaque point en utilisant la méthode "rotation()" et en suivant les règles de l'objet L-système. Les coordonnées sont stockées dans une liste et renvoyées en sortie de la méthode. Pour gérer les crochets "[" et "]" dans la chaîne de caractères, qui sont utilisés pour stocker et récupérer les coordonnées et l'angle lorsqu'une branche est créée ou terminée, la méthode utilise également une pile.

9.5 Génération du fichier Fenetre3D

La classe "Fenetre3D" du package Vue a été créée dans le but de permettre la création d'une fenêtre graphique en 3D utilisant OpenGL et d'afficher des coordonnées 3D à partir d'un ArrayList d'objets "Coordinate". Elle offre plusieurs fonctionnalités essentielles, notamment la gestion des capacités OpenGL, la création d'un canvas OpenGL, l'ajout d'un écouteur d'événements OpenGL, et la gestion de l'animation avec un FPSAnimator. De

plus, elle utilise la bibliothèque externe "com.jogamp.opengl" pour faciliter la gestion des fonctionnalités OpenGL. La méthode "getCoord()" permet d'obtenir la liste de coordonnées utilisée dans la scène 3D. La classe est organisée de manière à faciliter la création de la fenêtre 3D et la gestion des événements, bien que certains choix de conception, tels que l'utilisation de variables statiques, méritent d'être notés. La classe "Fenetre3D" joue un rôle clé dans la création de l'interface utilisateur graphique en 3D de notre projet, permettant d'afficher les coordonnées 3D de manière visuelle et interactive.

9.6 Gérer les exceptions avec des try and catch

Voici comment le programme gère les différents types de règles en fonction de l'option sélectionnée dans l'interface utilisateur :

1. Si l'option "lsystemStandart" est sélectionnée, le programme utilise un objet Rules pour stocker les règles de production. Les symboles et les productions sont extraits des champs de texte symboleField1, symboleField2, symboleField3, productionField1, productionField2, productionField3. Les valeurs de ces champs sont d'abord vérifiées pour s'assurer qu'elles ne contiennent que des entiers à l'aide de la méthode contientEntiers de l'objet panneauPrincipale, puis elles sont ajoutées à l'objet Rules en utilisant la méthode addElement avec le symbole extrait du champ de texte comme clé et la production extraite du champ de texte comme valeur. Enfin, l'objet Rules est défini comme règle de production dans l'objet panneauPrincipale en utilisant la méthode setRs.
2. Si l'option "contextuel" est sélectionnée, le programme utilise un objet Map avec des clés de type String et des valeurs de type String[] pour stocker les règles de production. Les symboles et les productions sont extraits des champs de texte symboleField1, symboleField2, symboleField3, productionField1, productionField2, productionField3 de la même manière que dans le cas précédent. Les productions sont ensuite stockées dans un tableau de chaînes de caractères chaine dans l'ordre des champs de texte. Enfin, le tableau chaine est associé à la clé extraite du champ de texte symboleField1 dans l'objet Map ruleContex, qui est défini comme règle de production dans l'objet panneauPrincipale en utilisant la méthode setRuleContext.
3. Si aucune des options précédentes n'est sélectionnée, le programme utilise un objet Map avec des clés de type Character et des valeurs de type String[] pour stocker les règles de production. Les symboles et les productions sont extraits des champs de texte symboleField1, symboleField2, symboleField3, productionField1, productionField2, productionField3 de la même manière que dans les cas précédents. Les productions sont ensuite stockées dans un tableau de chaînes de caractères chaine dans l'ordre des champs de texte. Enfin, le tableau chaine est associé à la clé extraite du champ de texte symboleField1 converti en caractère dans l'objet Map ruleStoch, qui est défini comme règle de production dans l'objet panneauPrincipale en utilisant la méthode setRuleStoch.

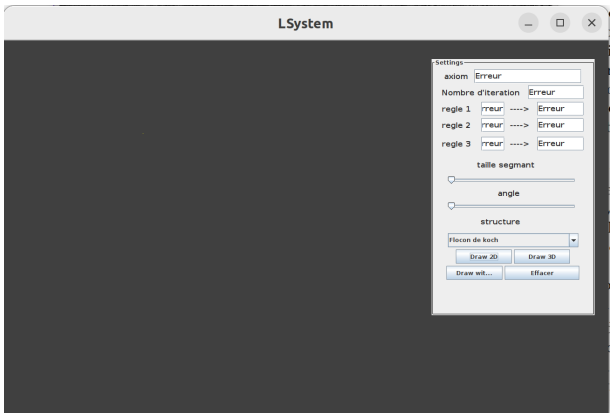

FIGURE 11 – Exemple de génération d'une fenêtre 2D avec des champs vides pour permettre l'essai de différentes fonctionnalités

10 Présentation des bibliothèques utilisés

10.1 Description détaillée d'un exemple d'implémentation de l'interpréteur

Alphabet : A, B
Axiome : A
Règles : $A \rightarrow AB$
Règles : $B \rightarrow A$
Angle : 60°
Itérations : 6

* n = 0, A
* n = 1, AB
* n = 2, AB A
* n = 3, AB A AB
* n = 4, AB A AB AB A
* n = 5, AB A AB AB A AB A AB
* n = 6, AB A AB AB A AB A AB AB A AB AB A

10.2 Présentation des bibliothèques utilisés

Swing.JFrame : cette bibliothèque est souvent utilisée pour créer des interfaces graphiques en Java. Par exemple pour créer notre projet on a utiliser JFrame pour créer la fenêtre principale de l'application et y ajouter des boutons, des champs de texte, etc.

Com.jogamp.opengl :La bibliothèque OpenJOGEL est une librairie de programmation en Java qui permet de créer et de manipuler des objets géométriques en 2D et 3D. Elle offre un large éventail de fonctionnalités pour la modélisation, la visualisation et la

manipulation d'objets géométriques complexes.

OpenJOGEL est basée sur le système de coordonnées cartésiennes, et fournit des classes pour représenter des points, des vecteurs, des courbes, des surfaces, des polygones, des transformations géométriques, et bien plus encore. Elle offre également des fonctionnalités pour le rendu graphique, l'interaction utilisateur, la gestion des événements et la gestion des primitives graphiques.

Java.awt.Graphics : cette bibliothèque est utilisée pour dessiner des formes et des images dans des applications Java. elle permet de dessiner des graphiques en temps réel, on peut utiliser Graphics pour dessiner les données sur un canevas en utilisant des formes telles que des lignes, des cercles, des courbes, etc.

Java.util.Random : cette bibliothèque est utilisée pour générer des nombres aléatoires dans des applications Java. elle nous permet des nombres aléatoire elle utile dans le Système stochastique.

11 Extensions du projet

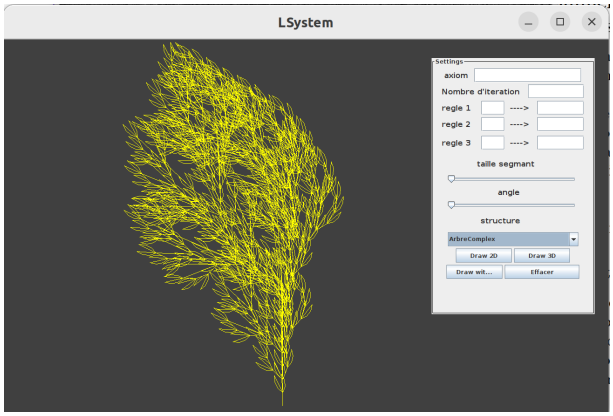

FIGURE 12 – Interface graphique 2D

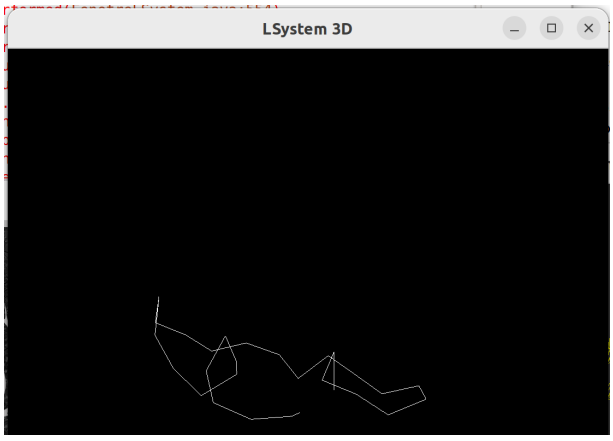

FIGURE 13 – Interface graphique 3D

12 Améliorations

12.1 Améliorations faites

Dans le cadre de notre projet, nous avons cherché à améliorer la convivialité et l'esthétique de notre application en ajoutant une fonctionnalité d'animation. Nous avons ajouté une fonctionnalité d'animation à notre projet en utilisant un timer pour améliorer l'expérience utilisateur. Nous avons créé un objet Timer avec une période de rafraîchissement de 200 ms, qui a été associé à un ActionListener pour gérer les événements du timer. Dans la méthode actionPerformed(), nous avons incrémenté l'index de ligne actuel à chaque tick du timer, ce qui a permis de mettre à jour l'affichage des nouvelles lignes dans la zone de dessin à chaque itération. Nous avons également arrêté le timer lorsque toutes les lignes ont été dessinées, en vérifiant si l'index de ligne actuel était supérieur au nombre d'itérations souhaitées. De plus, nous avons redémarré le timer lorsque l'index de ligne actuel était égal à zéro, pour permettre une boucle continue de l'animation. Cette animation a rendu notre projet plus interactif et visuellement attrayant, offrant ainsi une meilleure expérience utilisateur. Toutefois, nous reconnaissons que des améliorations pourraient être apportées à l'avenir, notamment en termes de performances pour un grand nombre d'itérations et de fonctionnalités supplémentaires pour enrichir davantage l'animation.

12.2 Améliorations possibles

- Amélioration de la fluidité de l'animation : en ajustant ajuster la période de rafraîchissement du timer pour obtenir une animation plus fluide, en testant différentes valeurs et en choisissant celle qui offre la meilleure expérience utilisateur en termes de fluidité des mouvements.
- Personnalisation de l'animation : tels que la vitesse, la direction, la couleur, etc. Cela peut ajouter de l'interactivité à l'animation et permettre à l'utilisateur de créer des animations personnalisées en fonction de ses préférences.
- Ajout de fonctionnalités avancées : telles que la gestion des collisions, la simulation physique, ou l'intégration de données externes pour influencer le comportement de l'animation.

13 Conclusion

En conclusion, ce rapport présente le développement d'un interpréteur de L-système en Java pour générer des images 2D/3D de structures végétales à partir de règles de réécriture. Le projet comprend la conception d'un parser, d'un moteur de réécriture et d'un moteur de rendu graphique, ainsi que des expérimentations pour évaluer la performance et la qualité des images générées. Les extensions possibles du projet, telles que la prise en charge des L-systèmes stochastiques et/ou contextuels, sont également discutées. Ce projet offre ainsi une base solide pour la génération de structures végétales réalistes dans des applications de jeux vidéos, d'animation, de modélisation graphique, et peut être étendu pour répondre à des besoins spécifiques. Les résultats obtenus dans ce projet de groupe ont démontré la capacité des membres à collaborer, à mettre en œuvre leurs compétences individuelles, et à atteindre les objectifs communs. Les améliorations apportées tout au

long du projet, notamment l'ajout d'une animation avec un timer, ont permis d'enrichir l'expérience utilisateur et d'améliorer la qualité du produit final.

14 Reference

— Prusinkiewicz, P., & Lindenmayer, A. (1990). The Algorithmic Beauty of Plants. Springer-Verlag. [Livre][5.1]
— Prusinkiewicz, P., & Hammel, M. (1993). A Fractal Model of Mountains with Rivers. Proceedings of Graphics Interface '93. [Article][5.1]
— `https://en.wikipedia.org/wiki/L-system` [Article Wikipédia sur les L-systèmes] [5.1]
— `https://fr.wikipedia.org/wiki/L-Syst%C3%A8me`
— `http://paulbourke.net/fractals/lsys/`
— `https://www1.biologie.uni-hamburg.de/b-online/e28_3/lsys.html`