Université de Caen Normandie
UFR des Sciences
Département Informatique

2ème année de licence d'informatique

L-Systeme

Conception logicielle

Rapport de projet

Antonin Boyon
Thomas Lalong
Quentin Legot
Arthur Page

Table des matières

Table des figures

1 Introduction

1.1 Sujet et consignes

Ce projet a pour objectif de réaliser une application appliquant des principes de programmation orientée objet en langage de programmation Java. Nous avons eu le choix entre 6 sujets différents et, après études des propositions, notre choix s'est finalement porté sur le "Générateurs de flores vidéos-ludiques". Il consiste en la réalisation d'un simulateur de L-système végétal produisant une image 2D et 3D de l'objet par le biais de règles de réécritures.

> Pour cela nous avions quelques consignes à respecter :
>
> — Intégrer un parser de L-système.
>
> — Créer un moteur de réécriture.
>
> — Créer un moteur de rendu graphique.

Après lecture des consignes, nous avons pu entamer nos recherches.

1.2 Mise en place du projet

Nos recherches se sont premièrement portées sur le L-Système (principalement sur Wikipedia [1]) pour comprendre son fonctionnement et nous donner des informations sur la construction de notre parser et de notre moteur de réécriture. Nous nous sommes ensuite renseignés sur les différents moteurs de rendu graphique que nous pouvions utiliser et notre choix s'est finalement porté sur JOGL (Java Open Graphics Library [2]) qui était conseillé dans la liste des sujets, pouvant gérer un rendu 2D et un rendu 3D.

Suite à cela, nous avons réfléchi à la structure de notre code, ainsi qu'à une première ébauche sur laquelle nous pourrions nous baser pour débuter notre projet et un ordre de priorités ; certaines parties étant nécessaires pour que d'autres fonctionnent ou puissent être amorcées (comme le parser, les bases du système de réécriture ou encore les différents moteurs de rendu).
Puis, pour terminer notre mise en place, nous avons décidé que nous rajouterions une interface ainsi qu'une fenêtre d'aide à notre futur logiciel dans le but de faciliter son utilisation.

1. https://en.wikipedia.org/wiki/L-system
2. https://jogamp.org/jogl/www/

2 L-Système

2.1 Principe et fonctionnement

2.1.1 Qu'est-ce que le L-Système ?

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

2.1.2 Comment fonctionne-t-il ?

Ce système de réécriture fonctionne par le biais de plusieurs spécificités :

— Un alphabet : celui-ci est l'ensemble des variables et des constantes utilisées.

— Un axiome : il représente le point de départ, l'état initial du système.

— Des règles de réécriture : elles définissent les règles de développement du L-Système en utilisant l'alphabet donné dans le but de créer un mot.

En additionnant tous ces aspects, nous obtenons alors notre L-Système, commençant par l'axiome étant la base, puis, créant au fur et à mesure un mot grâce aux règles données (Dans la limite du nombre d'itérations imposés [3]), tout ceci étant possible grâce à l'alphabet qui les composent. Ce mot passera ensuite par un moteur graphique dans le but d'être modélisé.

2.2 Notre L-Système

2.2.1 Alphabet

Notre alphabet est composés de plusieurs règles et constantes :

— X permet de dessiner une branche et Y de ne rien dessiner, ils permettent avec certaines règles de contrôler l'évolution de notre L-Système.

— Il est possible de modifier l'angle d'une branche en utilisant par exemple les +, -, -35, +64y, qui donnera respectivement une orientation de 25° et -25° sur l'axe de rotation x, une rotation de -35° sur l'axe X et une orientation de 64° sur l'axe de rotation y ; il n'est pas possible de modifier l'orientation de l'axe de rotation Z.

> Exemple : +XYX appliquera une rotation de 25° a X et aucune à Y et au 2ème X

— Enfin il est possible d'utiliser les crochets [] pour contrôler l'évolution et obtenir des branches à vos arbres. Ces crochets vont conserver l'état, c'est-à-dire qu'une

1. Le système de Lindebmayer
2. Modèle de calcul transformant des objets syntaxiques comme des mots, des termes ou encore des graphes en appliquant des règles données.
3. Le nombre d'itérations ou nombre de générations correspond au nombre de réécritures de l'axiome pour obtenir le mot final

rotation appliquée aux crochets s'appliquera a tous les éléments étant à l'intérieur des crochets. Il est possible d'imbriquer des crochets.

> Exemple : +[XYX] appliquera une rotation de 25° à XYX.

2.2.2 Axiome, règles de réécritures et nombre d'itérations

Pour l'axiome, les règles de réécritures et le nombre d'itérations, ils seront définis par l'utilisateur dans les zones de textes de l'interface prévues a cet effet. Un bouton "Aide" est présent sur cette même interface aidant à comprendre et mettre en place le L-Système.

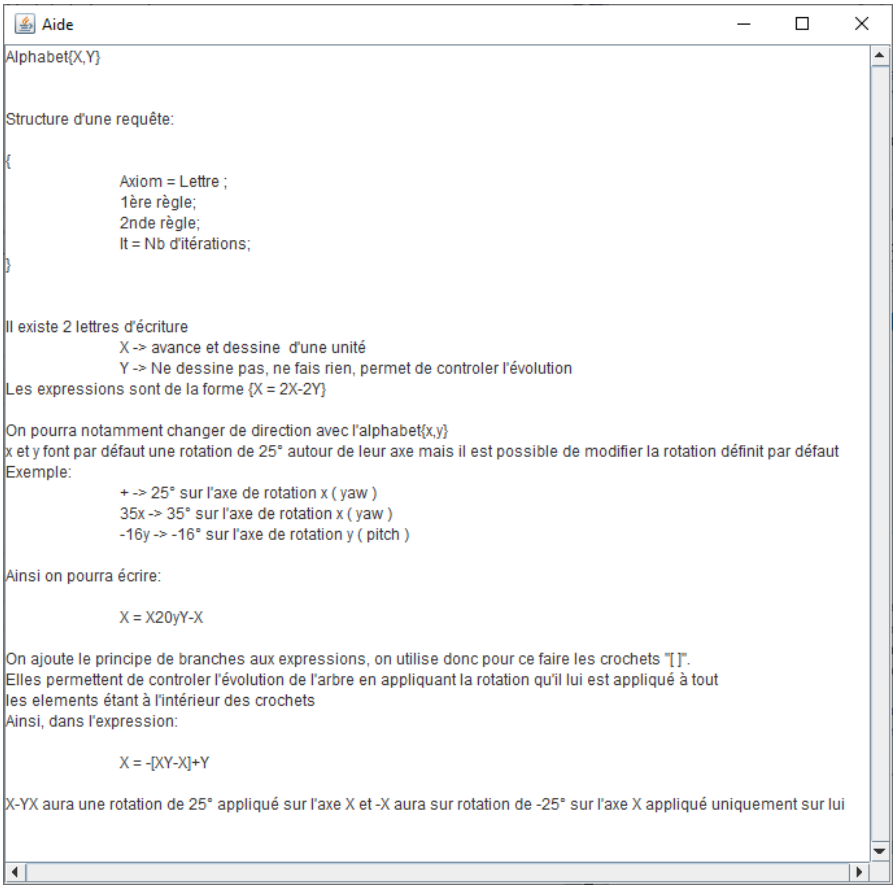

FIGURE 2.1 – Fenêtre d'aide

3 Organisation et structure

3.1 Organisation du sujet

3.1.1 Ordre de priorité

L'organisation du projet a été soumit a un ordre de priorité car, comme cité dans "Mise en place du projet" (voir section 1.2), certaines parties étaient nécessaires pour que d'autres fonctionnent ou puissent être amorcées :

- La création de l'alphabet de notre L-Système (voir section 2.2.1).
- La mise en place de la vérification des termes contenus dans les règles et l'axiome.
- La création du moteur de réécriture.
- La mise en place du Parser qui transforme le mot obtenu par le moteur de réécriture en une structure de données plus facilement lisible ; afin d'être affichée par le moteur graphique.
- La création du moteur graphique.
- Et, pour faciliter l'utilisation de notre code, la création d'une interface.

3.1.2 Repartition des tâches

L'ordre de priorité étant fixé, nous sommes passé à la répartition des tâches. La création de l'alphabet s'est premièrement effectuée ensemble.
Ensuite nous nous sommes répartis les tâches, Arthur et Quentin travaillant sur le Parser, Antonin et Thomas commençant à faire des recherches sur le moteur graphique que nous pourrions utiliser.
Après cela, Quentin passa à la création du moteur graphique,aidé d'Arthur pour le déplacaement de la caméra, Antonin finalisant le Parser, Arthur et Thomas créant l'interface.
Une fois toutes les parties principales créées et avancées, tout le monde a travaillé dans l'amélioration de celles-ci.
Pour finir, la création du rapport s'est effectuée, elle aussi, en groupe.

3.2 Structure du projet

- engine
  - Rewrite : Moteur de réécriture
  - Element, ElementProperties et Parser : voir section 4.2
- screen
  - Gl3d : Tout les objets relatifs a l'affichage 3d du L-Systeme, voir la section 4.3
  - Main : Tout les objets relatifs au menu, voir la section 4.4
- utils : contient l'objet Pair qui est essentiel au fonctionnement du projet

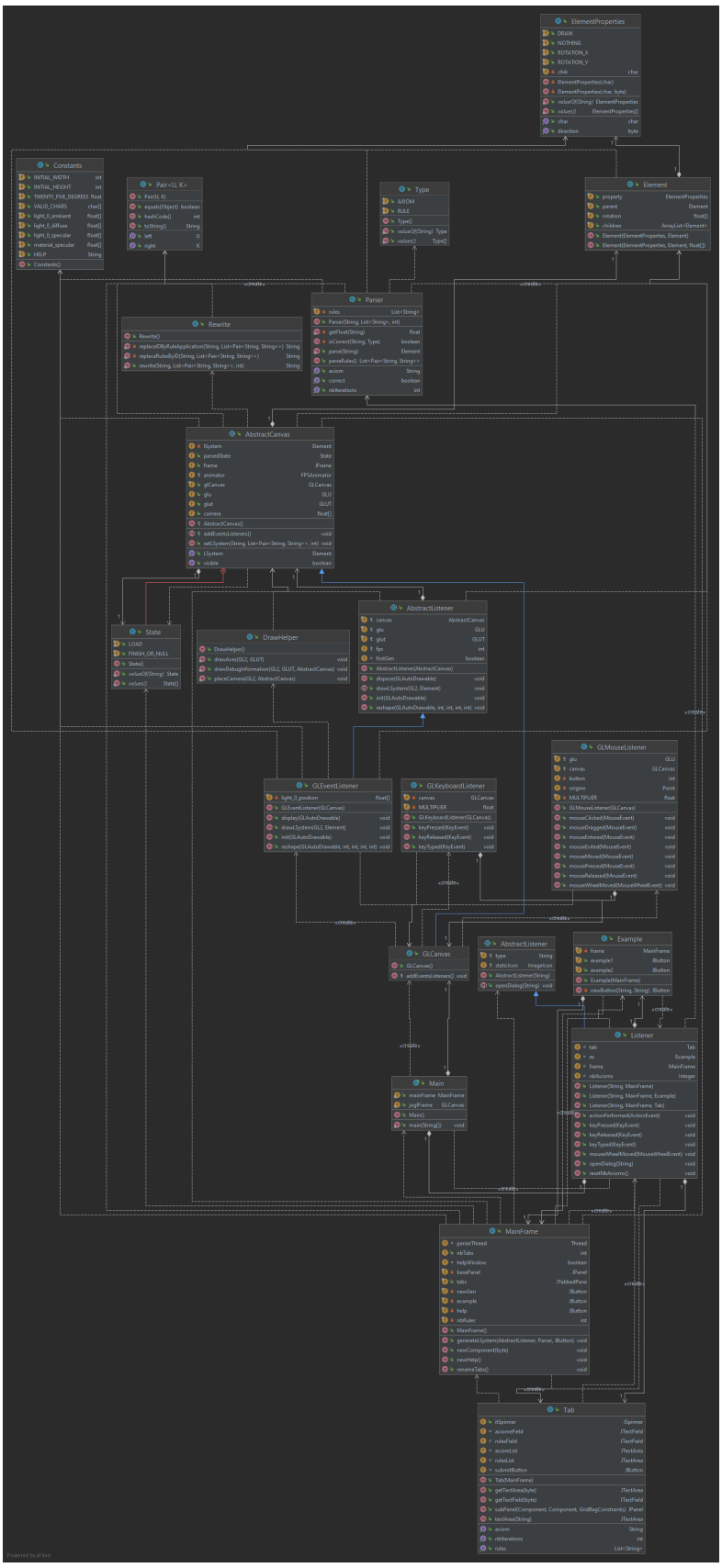

FIGURE 3.1 – Diagramme de classe de notre projet

4 Élements techniques

4.1 Moteur de réécriture

Le moteur de réécriture est chargé d'appliquer les règles de réécritures itérativement à partir de l'axiome pour obtenir le mot final. Nous avons d'un côté l'axiome qui est le mot de départ et une liste de règles qui contient 2 données : le mot à remplacer et son remplaçant. Le mot est réécrit en 2 temps :

— On change toutes les occurrences du mot à remplacer par ${identifiant de la règle}.

— On remplace toutes les occurrences de ${identifiant de la règle} par la règle associée.

On répète cette opération n fois, pour obtenir le mot final.

L'algorithme de réécriture est séparé en 3 parties :

Algorithme 1 : Rewrite Fonction principale appelée par la fenêtre principale

Entrées : axiom : $String$, rules : $List[Tuple[String, String]$, recurrence : $Integer$
Sortie : Le mot réécrit itérativement : $String$

$rewritten \leftarrow \emptyset$
pour $i \leftarrow 0$ ***à*** $recurrence$ **faire**
  $toRewrite \leftarrow replaceRulesByID(rewritten, rules)$
  $rewritten \leftarrow= replaceIDByRuleApplication(toRewrite, rules)$
fin
$rewritten = rewritten.replace("[","Y[")$
retourner $rewritten$

Algorithme 2 : replaceRulesByID remplace les occurrences des différentes règles par ${id}

Entrées : rewritten : $String$, rules : $List[Tuple[String, String]$
Sortie : Le mot partiellement réécrit

```
1 toRewrite ← rewritten
2 pour i ← 0 à rules.size() faire
3 |   pair ← rules.get(i)
4 |   toRewrite ← toRewrite.replace(pair[0], "${" + i + "}")
5 fin
6 retourner toRewrite
```

Algorithme 3 : replaceIDByRuleApplication remplace les occurrences ${id} par la définition des différentes règles

Entrées : toRewrite : $String$, rules : $List[Tuple[String, String]$
Sortie : Le mot entièrement réécrit

```
1 rewritten ← toRewrite
2 pour i ← 0 à rules.size() faire
3 |   rewritten ← rewritten.replace("${" + i + "}", rules.get(i)[1]
4 fin
5 retourner rewritten
```

4.2 Parser

Le rôle du parser est à partir d'un mot réécrit par le moteur de réécriture de convertir celui-ci en une structure de données plus facilement manipulable afin de l'afficher dans le moteur graphique. En appliquant les règles précédemment énoncées, il construit un arbre grâce à la structure de données suivante.

Algorithme 4 : Element Branche de l'arbre, si parent est vide alors, cet élément est la racine

```
1 struct {
2 |   propery : ElementProperty      ▷ enumérateur, valeurs possibles : DRAW ou
  |     NOTHING
3 |   parent : Element
4 |   rotation : Float[3]
5 |   children : List[Element]
6 } Element
```

4.3 Moteur graphique

Nous utilisons *Java* OpenGL pour afficher une fenêtre 3D, gl2 pour afficher les lignes de la grille, glu pour placer la caméra et GLUT pour afficher les cylindres.
Pour naviguer dans l'espace 3D, voir la section 5.1.4.
Pour afficher le L-Système nous utilisons une méthode récursive.

Les classes GLKeyboardListener et GLMouseListener écoutent respectivement le clavier et la souris.

GLCanvas crée la fenêtre (grâce à AbstractCanvas), GLEventListener initialise et affiche l'environnement 3D.

DrawHelper est une classe utilitaire qui permet d'afficher certains éléments afin d'alléger la méthode display située dans GLEventListener.

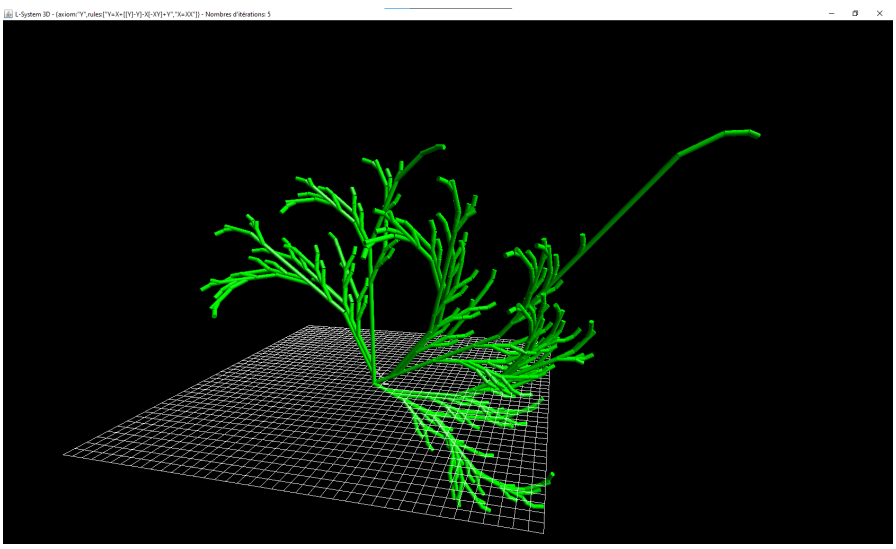

FIGURE 4.1 – Fenêtre 3D

4.4 Interface principale

4.4.1 Composition de l'interface

L'interface utilisateur de notre logiciel a été conçu grâce à la bibliothèque *Swing* de *Java*. Elle se compose de trois classes, une contenant la fenêtre principale ***MainFrame***, une autre permettant de créer des onglets, ***Tab*** et une troisième classe gérant les événements, ***Listener***.

4.4.2 Classes de l'interface

MainFrame

La classe *MainFrame* est une classe héritant de la classe JFrame de *Swing*. Elle permet de créer une fenêtre de base, de taille prédéfinie, dans laquelle peuvent être placés des composants graphiques. Elle comprend aussi un bouton de fermeture qui, une fois cliqué, permet l'arrêt du programme.
Elle comporte aussi une instance de la classe JTabbedPane , un conteneur graphique dont le but est de disposer ses composants sous la forme d'onglets.

Tab

La classe *Tab* est une classe héritant de la classe JPanel de *Swing*. JPanel est un composant de base dans lequel il est possible d'ajouter d'autres composants graphiques. Les instances de ***Tab*** créées sont ensuite ajoutées par la classe ***MainFrame*** à son composant de la classe JTabbedPane 4.4.2.

Listener

La classe *Listener* est une classe implémentant certaines classes Listener de *Swing*(***ActionListener, KeyListener et MouseWheelListener***). Elle permet de capter toutes les actions effectuées par l'utilisateur et d'appeler les méthodes correspondantes des classes de l'interface. Elle permet ainsi de créer de nouveaux onglets (Nouvelles instances de Tab) mais aussi d'en fermer ou bien encore de lancer la génération du modèle.

4.5 Pair ou un tuple a 2 entrées en java

Nous avons utilisé les classes génériques afin de créer une classe nommé Pair, elle possède 2 paramètres génériques nous permettant de lui assigner des éléments de n'importe quel type.
Une fois instanciée avec ces 2 éléments, son contenu n'est plus modifiable, ce qui en fait un uplet à 2 éléments (couple).
Nous l'utilisons ici pour stocker les règles, la partie gauche du couple stocke le contenu qui va être remplacé dans l'axiome et la partie droite le remplaçant.
Nous l'utilisons seulement dans ce projet pour stocker des mais il est possible de stocker n'importe quelle instance d'un objet.

5 Experimentations et Usages

5.1 Manuel d'utilisation

5.1.1 Préambule

Notre application a été développé et pensé pour les versions de java supérieures ou égales à la version 8u281. L'application fonctionne sur Linux avec une interface tournant sur les moteurs graphiques Xorg et Wayland et sur Windows 10. Notez que pour linux, notre programme ne fonctionne que pour openjdk 8

Les archives jar de Jogl doivent se trouver dans le dossier lib selon le modèle ci-dessous (image)

> Nous ne pouvons pas vous garantir si l'application fonctionne sur Mac OS X, aucun des membres de notre n'en possède un.

5.1.2 Lancement de l'application

> Vous devez ouvrir un terminal à l'emplacement du dossier contenu le projet

Pour lancer l'application, exécutez la commande *ant run*

Si vous souhaitez seulement compiler les fichiers sources dans le répertoire **bin**/, executez la commande : *ant compile*

Pour générer une archive jar dans le répertoire **build**/, executez la commande : *ant packaging*

Pour générer la javadoc dans le dossier **doc**/, executez la commande : *ant javadoc*

> Ouvrez ensuite le fichier **doc/index.html** ou **doc/overview-summary.html** dans un navigateur.

Pour effectuer les tests, executez la commande : *ant tests*

> Un fichier **result.txt** sera générée affichant les résultats ainsi qu'une copie de la sortie standard.

5.1.3 Utilisation de l'interface utilisateur

Une fois l'application lancée, une fenêtre s'affiche (figure 5.1). Elle contient une barre de navigation grâce à laquelle vous pouvez ouvrir soit une nouvelle génération, soit une fenêtre d'aide, ainsi qu'un onglet de génération. Il ne vous reste ensuite plus qu'à renseigner votre axiome, ainsi que vos règles et de cliquer sur le bouton *Générer en 3D*. Le bouton *Close* permet de fermer l'onglet de génération et le bouton *Clear* de

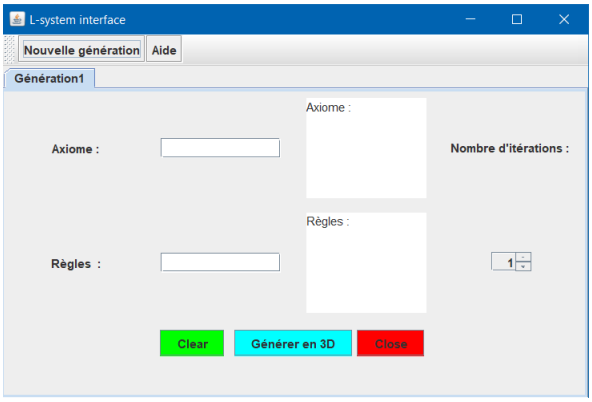

FIGURE 5.1 – Fenêtre principale

supprimer votre axiome et vos règles précédemment écrites. Grâce au compteur à droite, vous êtes en mesure de définir le nombre d'itérations de votre génération.

> Vous pouvez ouvrir de nouveaux onglets de génération grâce au bouton *Nouvelle génération* mais sachez qu'un maximum de trois fenêtres est accepté

5.1.4 Navigation dans l'interface graphique en 3D

Pour naviguer dans l'espace 3D, vous pouvez utiliser votre clavier ainsi que votre souris.

> La souris n'est pas essentielle, elle permet de se déplacer facilement dans l'environnement mais un clavier suffit amplement

Liste des commandes au clavier :

- **Z** → *Avancer*
- **S** → *Reculer*
- **Q** → *Aller à gauche*
- **D** → *Aller à droite*
- **A** → *Tourner la caméra à gauche*
- **E** → *Tourner la caméra à droite*
- **W** → *Prendre de la hauteur*
- **X** → *Perde de la hauteur*

Liste des commandes à la souris :

- **Mollette Avant** → *Zommer*
- **Mollette Arrière** → *Dézoomer*
- **Clic Droit** → *Maintenir puis bouger la souris pour changer l'orientation de la caméra*

> Vous ne pouvez pas utiliser 2 touches ou plus en même temps pour naviguer. Par exemple, enfoncer les touches **Z** et **D** pour aller la direction nord-est est impossible, il vous faut tourner votre caméra dans la direction où vous voulez aller puis appuyer sur **Z**.

Fermez la fenêtre 3D pour pouvoir générer un nouveau L-Système sans avoir à rouvrir l'application

5.2 Tests de notre logiciel

5.2.1 Possibles problèmes

Lorsque vous tentez de générer un L-Système, celui-ci est affiché en 3D en utilisant une méthode récursive, si celui-ci est trop long cela peut entraîner une erreur de type *StackOverflowError*, la fenêtre 3D restera alors blanche. Fermez la puis retentez une génération avec moins d'itérations ou tentez une génération avec d'autres règles.

Si vous lancez une génération avec beaucoup d'itérations, celle-ci peut mettre un certain temps à se générer et peut provoquer une erreur *OutOfMemoryError*.

5.3 Mesures de performances

Nous avons mis en place un stress-test qui réécrit et convertit en arbre le L-Système. Ce stress-test nécessite 4GB de RAM de libre et construira un mot de quasiment 26 millions de caractères puis celui-ci sera converti en un arbre (notez qu'il est impossible de l'afficher dans le moteur graphique car cela donnera *StackOverflowError*).

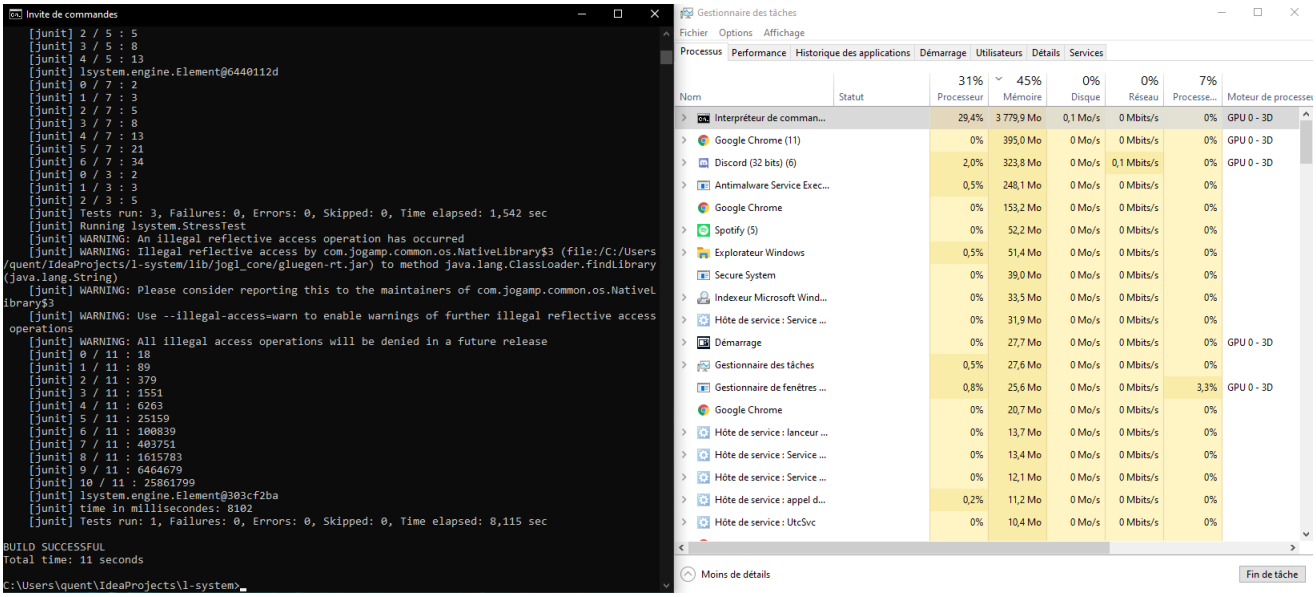

FIGURE 5.2 – Mesure de performances

(Test de performances effectué avec un Ryzen 5 2400g et une gtx 1050ti.)

5.4 Possibles améliorations

- Améliorer l'optimisation.
- Rendre l'interface graphique plus conviviale.
- Réaliser les L-Système 2D dans un vrai environnement 2D, et non dans un moteur 3D comme actuellement.
- Améliorer le réalisme de la modélisation 3D.
- Exporter les modèles d'arbres afin de les implémenter facilement dans des applications comme des jeux.

— Utilisation des méthodes de OpenGL 3 ou 4 au lieu de ceux de OpenGL 2 afin de profiter des dernières améliorations technologiques des cartes graphiques (notamment au niveau des performances).

6 Annexes

6.1 Sources

- Wikipedia L-Système (EN) : https://en.wikipedia.org/wiki/L-system
- 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/
- Javadoc : https://junit.org/junit4/javadoc/latest/

6.2 Remerciements

Triss Jacquiot qui nous a permit de s'inspirer de son rapport pour la mise en forme.