

RAPPORT DE CRÉATION D'UN PROJET EN PROGRAMMATION ORIENTÉE OBJET

mis en page par
Romain ANDRES, Marta BOSHKOVSKA, Ronan CARRÉ, Sara SALÉ

Licence 2 Informatique
Groupe 4A

Avril 2022



UE : Conception logicielle avancée

Projet choisi : *Générateur de flores vidéos-ludiques*

1 INTRODUCTION

Actuellement en deuxième année de Licence Informatique, au cours de l'UE Conception logicielle Avancée il nous a été demandé de réaliser une application parmi sept proposées en utilisant le langage de programmation orientée objet Java. Pour notre part, nous avons choisi le projet *Générateur de flores vidéos-ludiques*.

Ainsi tout au long de ce rapport nous vous présenterons, le processus de réalisation de notre projet en passant par le choix de celui-ci, les moyens mis en place pour la réalisation et la présentation de l'application en elle-même.

Table des matières

1 INTRODUCTION	2
2 LE PROCESSUS DE RÉALISATION	4
2.1 Choix du sujet : Générateur de flores vidéos-ludiques	5
2.2 Premiers essais	5
2.3 Re-considération du cahier de charge et élaboration de la version 2D	7
2.3.1 La récursivité et les moyens de réécriture	7
2.3.2 La construction d'un parser	10
2.3.3 Les éléments de dessin	10
2.4 Réalisation de la version 3D	11
2.4.1 Le choix de la bibliothèque pour l'interface graphique	11
2.4.2 Dessiner dans un espace 3D	11
3 L'ORGANISATION ET LE FONCTIONNEMENT DU PROGRAMME	13
3.1 Organisation architecturale du projet	14
3.1.1 Le répertoire	14
3.1.2 Diagramme de classe	14
3.2 Le fonctionnement du programme	17
3.2.1 Le manuel d'utilisation	17
3.2.2 Aperçu du lancement du programme	17
3.3 Répartition des tâches	20
3.4 Pistes d'amélioration	20
4 Conclusion	20
Appendices	21
A Bibliographie et sitographie	21

2 LE PROCESSUS DE RÉALISATION

2.1 Choix du sujet : Générateur de flores vidéos-ludiques

Au total, sept sujets étaient proposés. Ce sont toutefois les sujets *Générateur de flores vidéos-ludiques* et *Interpréteur de programmes chimiques* qui ont attiré notre attention. Les autres projets proposés nous semblaient soit linéaires soit un peu plus axés sur l'intelligence artificielle ou tout simplement la présentation et les moyens de réalisation semblaient complexes. Le point commun entre ces deux sujets est la mise en place d'un parser. Après une analyse de nos points forts et de la documentation, notre choix s'est au final porté sur le *générateur de flores vidéos-ludiques*. De plus, *l'interpréteur de programmes chimiques* devaient nécessiter des données extérieures dont nous n'avions pas vraiment connaissance.

L'objectif du *générateur de flores vidéos-ludiques* est la mise en place d'un système de réécriture par lequel un utilisateur va donner les caractéristiques de production d'une image de flore. Cette image doit apparaître à l'écran sous forme 2D et sous forme 3D. Le projet demande donc une bonne organisation puisqu'il faut :

- un parser de L-système,
- un moteur de réécriture,
- puis un moteur de rendu graphique pour visualiser ces plantes

Afin de mieux nous projeter, lors des premières séances, nous nous sommes attelés à comprendre le fonctionnement des fractales, de la récursivité et des différentes librairies graphiques de Java. Cela a directement mené à nos premiers essais.

2.2 Premiers essais

Dans un premier temps, nous nous sommes intéressés au fonctionnement des fractales et l'interface graphique. Une fractale est une forme géométrique qui se répète à l'identique quand on change d'échelle. Notre projet nous demande précisément des figures à l'aspect de plantes.¹ N'ayant pas auparavant utilisés les bibliothèques graphiques de Java, Abstract Window Toolkit communément appelée AWT et Swing nous avons commencé par réaliser des petits dessins. Il est important de noter que notre prise en main de l'interface graphique de Java et des moyens mis en place pour réaliser des images 2D n'a pas été difficile. Il nous a juste fallu nous documenter sur les différents fondements et nous exercer à leurs applications. C'était l'une des parties les plus intéressantes car nous avions la possibilité d'avoir un aperçu visuel de tout ce que nous faisions. Toutefois ces dessins n'entraient pas dans le domaine des fractales.

Nous avons essayé de mettre en place une application qui permettait à l'utilisateur de remplir quelques champs de texte afin d'obtenir la flore souhaitée. En renseignant ces champs, on pouvait déterminer quelle flore voulue et à partir de la classe de la flore en question dessiner sur la fenêtre graphique. En réalité, nous voulions une interface qui à l'aide d'un parser produisait des images 2D de flores.

Mais nous n'avions pas réussi à trouver lier correctement le parser et l'interface graphique. Et surtout ce n'était toujours pas de la récursivité et c'était un assez confus. Ci-dessous, les images et leurs descriptions montrent les différents résultats obtenus.

1. selon [Wikidia](#)



FIGURE 1 – Grâce à ce menu, l’utilisateur entre les caractéristiques suivantes :
-le type de flore souhaité,
-la hauteur de la flore
-l’angle d’inclinaison
-la densité souhaité qui équivaut au nombre d’itérations.

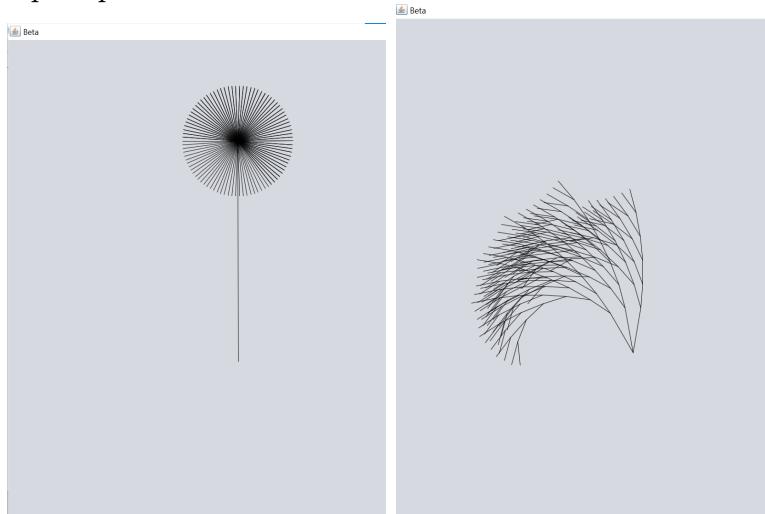


FIGURE 2 – L’utilisateur peut ainsi obtenir des figures de ce genre.

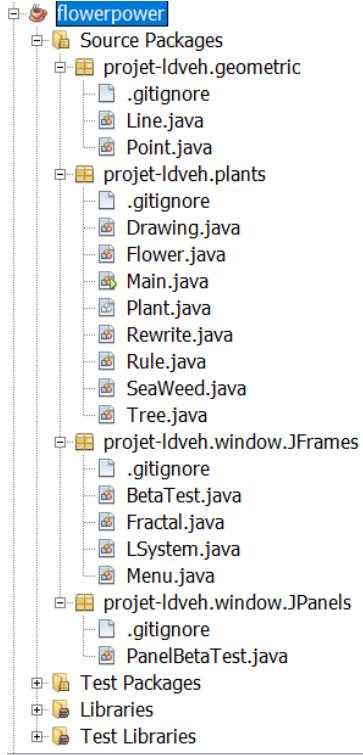


FIGURE 3 – Arborescence de notre projet

Parallèlement, l'organisation de nos packages et classes, comme le montre l'image ci-dessus, n'était pas appropriée. En effet, en plus du code redondant dans certaines classes, les classes n'étaient pas regroupées correctement. Quand on regarde l'arborescence par exemple, au lieu d'avoir un package destiné uniquement au parser et au dessin, tous se retrouvaient dans les packages destinés au élément de la flore.

Face à ce constat, nous avons effectué un modification complète de notre code, une réorganisation de notre travail et la redirection de nos objectifs.

2.3 Re-considération du cahier de charge et élaboration de la version 2D

2.3.1 La récursivité et les moyens de réécriture

Suite à notre première tentative, nous avons compris qu'il fallait réellement comprendre les notions de **récursivité** et de **réécriture** pour réussir notre projet.

La récursivité, c'est quand une fonction s'appelle elle-même jusqu'à atteindre une condition². Dans notre cas, la récursivité s'illustre par les fractales et les conditions d'arrêts sont par exemple le nombre d'itérations ou la longueur des plantes.

La création de fractales passe également nécessairement par la réécriture. L'utilisateur donne une chaîne de caractère et celle-ci doit être transformée selon des règles précises afin de générer une fractale. L'ensemble des règles de réécriture est appelé système de réécriture. Il comprend un alphabet (l'ensemble des variables du qu'on peut utiliser), un axiome de départ (l'état initial) et les différentes règles de réécriture (qui permettent de transformer l'axiome). Il existe plusieurs systèmes de réécriture :

- le L-Système, ou système de Lindenmayer : il permet la modélisation du processus de développement et de prolifération de plantes ou de bactéries ;

2. source :www.jesuisundev.com

- le DOL-Système : il est déterministe qui n'offre qu'une seule évolution possible depuis l'axiome à chaque génération ;
- et les SOL-Système : qui contrairement au DOL-système, un choix peut être opéré entre plusieurs transformations pour un symbole.

Nous avons choisi de mettre en place un DOL-Système. Par conséquence, nous n'avions plus besoin de certaines classes comme *Tree.java*, *Seaweed.java* et *Flower.java*. Nous devions plutôt définir une règle qui nous permettrait de réaliser n'importe quelle flore avec des caractéristiques définies. On utilise la classe *Rewrite.java* pour effectuer la réécriture. Sa méthode *replacement()* permet de réécrire une règle donnée par l'utilisateur.

L'algorithme se présente comme suit :

Algorithme 1 : Parse string to lines

Entrées : A string S, A double ANGLE, A integer WIDTH, A integer HEIGHT

Output : A finite set of lines

```
1 currentAngle ← 90
2 currentId ← 0
3 currentEndingPoint ← Point(0, 0)
4 res ← Array{Line}
5 pointsEncrages ← Array{Point}
6 pointsEncragesAngles ← Array{Double}
7 pointsEncragesId ← Array{Integer}
8 pointsEncrages.add(Point(width/2, height - 120))
9 pour i ← 0 to S.size faire
10   si S.charAt(i) ==' 1' alors
11     | currentId ← 1
12   fin
13   si S.charAt(i) ==' 2' alors
14     | currentId ← 2
15   fin
16   si S.charAt(i) ==' +' alors
17     | currentAngle ← currentAngle + ANGLE
18   fin
19   si S.charAt(i) ==' -' alors
20     | currentAngle ← currentAngle - ANGLE
21   fin
22   si S.charAt(i) ==' [' alors
23     | pointsEncrages.add(currentEndingPoint)
24     | pointsEncragesAngles.add(currentAngle)
25     | pointsEncragesId.add(currentId)
26   fin
27   si S.charAt(i) ==']' alors
28     | currentEndingPoint ← pointsEncrages.LastElement
29     | currentAngle ← pointsEncragesAngles.LastElement
30     | pointsEncrages.removeLastElement
31     | pointsEncragesAngles.removeLastElement
32     | pointsEncragesId.removeLastElement
33   fin
34   si S.charAt(i) ==' F' alors
35     | si je suis la première ligne de S alors
36       |   | F ← Line(currentId, pointsEncrages.get(0), LONGUEUR, 90)
37       |   | currentEndingPoint ← F.endingPoint
38       |   | res.add(F)
39     | fin
40   sinon
41     |   | F ← Line(currentId, currentEndingPoint, LONGUEUR, currentAngle)
42     |   | res.add(F)
43     |   | currentEndingPoint ← F.endingPoint
44     |   | currentAngle ← F.getAngle()
45   fin
46 fin
47 fin
48 retourner res
```

Selon le nombre d'itérations x, l'axiome est transformé x fois dans la règle.

2.3.2 La construction d'un parser

Parser, c'est faire une analyse syntaxique. Cette étape vient après la réécriture. C'est la partie traitement de données et c'est la classe *Parser.java* qui gère tout ces aspects.

Dans un premier temps, il faut déterminer les caractères que l'utilisateur a la possibilité d'employer et leurs significations à l'emploi. Nous avons donc un alphabet bien défini composé de huit caractères : "+-F[]012".

Et chaque caractère permet d'orienter la création de nos fractales :

```
'+' : signifie que l'angle est positif  
'-' : signifie que l'angle est négatif  
'F' : représente une ligne  
'[' : marque le début de branche  
'0' : représente la couleur 1  
'1' : représente la couleur 2  
'2' : représente la couleur 3
```

L'axiome par défaut est 'F'. L'utilisateur mette en place sa règle à l'aide des autres caractères de l'alphabet. Cependant, les caractères doivent être utilisés selon un ordre pour avoir des figures logiques. Notamment, on ne peut pas utiliser ']' avant d'utiliser '['. La méthode *isLexical()* de notre classe vérifie la concordance des crochets. La méthode *isSyntactic()* de la même classe vérifie si la règle fournie a un sens. *Parser.java* grâce à la méthode *parseStringtoLines()* permet de convertir la chaîne de caractères en tableau de lignes. Ces L'utilisateur doit aussi indiquer l'angle souhaité pour l'inclinaison des branches, la longueur des branches (la taille des branches) et le nombre d'itérations qu'il souhaite pour sa figure. Le nombre d'itérations correspond au nombre de fois où l'opération de "dessin" doit être répétée(on peut dire simplement qu'il correspond au n étages de branches souhaité). Ce nombre d'itération est aussi utilisé dans la réécriture. Toutes ces données sont également récupérées à l'aide des méthodes *getAngle()*, *getNbAxiomes()* et la longueur est utilisée pour les lignes dans *parseStringtoLines()*.

2.3.3 Les éléments de dessin

Plusieurs bibliothèques existent en Java pour réaliser des interfaces graphiques. Nous avons préféré utiliser [Swing](#) et [AWT](#) parce que nous en avions déjà entendu parler en cours. Aussi, il existe une relation entre ces deux classes et cela a facilité notre usage. Plus tard, on a découvert JavaFX. *Parser.java* à travers *parseStringtoLines()* permet de convertir la chaîne de caractères en tableau de lignes. Ces lignes sont générées et gérées à l'aide des classes *Point.java* et *Line.java*. Pour chaque figure, on a un tableau de lignes structurées. Chaque ligne a un id, et les lignes se succèdent les unes aux autres. Leurs inclinaisons les unes par rapport aux autres se fait selon l'angle donnée par l'utilisateur. Chaque figure est matérialisée comme un *ArrayList* de lignes et chacune de ces lignes est dessinée grâce à *javax.awt.Graphics*. Nous avons produit deux *JFrame*, l'une pour la fenêtre de démarrage et l'autre pour la génération de notre flore vidéo-ludique. Et cette dernière contient deux *JPanel* : la première qui permet la mise en place d'un menu de contrôle et la deuxième qui sert de zone de dessin. Puisque Swing permet de dessiner sur sa composante *JPanel* avec AWT. C'est sur cette dernière que nous avons redéfini la méthode *paint()* afin dessiner correctement nos lignes. Après cela n'est qu'une question d'agencement.

Dans notre processus de réalisation en 2D, nous avons remarqué que l'interface graphique après la prise connaissance de la documentation ne nous a pas posé de problème. Cependant,

la non-compréhension des attendus au niveau de la réécriture et du parsing nous a énormément perturbé.

2.4 Réalisation de la version 3D

Une fois la partie 2D terminée, nous nous sommes penchés sur la modélisation en 3D de nos flores.

2.4.1 Le choix de la bibliothèque pour l'interface graphique

Nous avons donc cherché un moyen d'afficher des formes 3D dans notre JPanel. La bibliothèque Swing ne permettant pas de faire cela et se limitant à la 2D avec la méthode *drawLine*, une implémentation de la 3D avec lignes 2D aurait été très laborieuse et longue. Donc, nous avons donc opté pour trouver une bibliothèque Java qui permettait d'afficher des figures 3D. Nous avons premièrement voulu implémenter la bibliothèque OPEN GL. Elle permet de faire tout ce que l'on souhaite en 3D mais nous avons constaté que la bibliothèque était assez ancienne et donc incompatible avec nos versions de Java actuelles.

Nous sommes ensuite tournés vers JavaFX puisqu'elle permet d'être utilisée dans Swing. La bibliothèque est mise à jour jusqu'à la dernière version de Java actuelle donc pour la compatibilité il n'y avait aucun problème. Cependant, nous avons pris une semaine pour enfin pouvoir utiliser JavaFX car nous n'arrivions pas à l'installer convenablement. Nous avons essayé de l'implémenter dans notre répertoire **lib** mais de nombreuses erreurs survenaient. Une partie du problème venait de la compilation. Nous avons alors créé un Makefile qui compilait avec succès la librairie en ajoutant les modules requis et le chemin relatif de la librairie. Mais même après cela, il y avait encore des erreurs.

Nous avons donc essayé une autre approche en essayant d'utiliser le logiciel Maven, qui s'occupe d'implémenter la librairie automatiquement au projet avec une connexion internet mais nous n'avons pas réussi à le faire fonctionner.

Après un retour aux sources, nous avons finalement constaté que la version de la librairie JavaFX que nous avons mis dans notre répertoire **lib** était la mauvaise version. Une fois la bonne version utilisée, le seul inconvénient qui survient est la détection de la librairie et donc de toutes les lignes de code liées à la librairie par l'éditeur de texte ou de l'environnement de développement puisque dans certains cas, celle-ci n'est pas détectée. Codant sur le logiciel Visual Studio Code, nous n'avons pas réussi à résoudre ce problème et avons du faire avec.

2.4.2 Dessiner dans un espace 3D

Une fois l'installation de JavaFX maîtrisée, plusieurs questions se sont posées : Comment dessiner nos flores, avec quoi, et dans quoi ? Nous avons donc trouvé le composant JFXPanel qui pouvait contenir des groupes d'objets 3D avec des scènes 3D. La meilleure option que nous avons trouvé consiste donc à procéder en plusieurs étapes.

Premièrement, il fallait afficher l'arbre tel que représenté dans la version 2D dans un monde 3D. C'est-à-dire, avoir un arbre tout plat dessiné uniquement en fonction de X et de Y dans un monde avec X Y Z. Nous avons donc un arbre qui possède une hauteur et une largeur mais une profondeur fixe avec ce procédé. Pour faire cela, nous avons donc cherché à afficher nos lignes en 3D avec un radius. Après avoir cherché dans la documentation nous avons trouvé que la manière la plus simple était d'utiliser des objets déjà créés dans la librairie, ainsi nous avons utilisé la classe *Cylinder* comprise dans JavaFX pour créer nos lignes. Ici, nos lignes sont des cylindres. Nous avons trouvé sur internet une fonction qui permettait de tracer un Cylindre d'un point A à un point B et avons donc décider d'utiliser cette fonction. Ainsi pour afficher

notre arbre, il ne nous restait plus qu'à parcourir comme dans la version 2D notre arraylist de lignes. Pour chaque ligne, tracer un cylindre qui commence au point départ de cette ligne et qui se termine au point de fin de celle-ci. Avec ce procédé, nous avons pu avoir une figure en 3D qui s'approchait pas mal de notre version 2D. Toutefois, la représentation au niveau des angles était assez déformée.

Pour boucher le trou à l'intersection des cylindres, nous avons trouvé comme solution d'y placer des sphères 3D grâce à la classe *Sphere* de JavaFX. Nous avons donc mis des sphères à tous les points de fin de nos cylindres et avons obtenu un beau résultat.

Après cette première étape terminée, nous avons mis en place une caméra afin de pouvoir mieux observer notre arbre dans notre scène. Il a fallu créer une fonction d'initialisation de la caméra pour l'utiliser dans le constructeur. Nous avons par la suite ajouté des évènements clavier pour diriger la caméra. Les flèches directionnelles du clavier servent à aller de gauche à droite et de haut en bas ; la molette de la souris sert à avancer et reculer et les numéros du pavé tactile '2' et '8' servent à incliner vers l'avant ou vers l'arrière la caméra. Nous n'avons malheureusement pas réussi à faire pivoter la caméra.

Parallèlement, nous nous sommes attaqués à l'étape de la mise en place de la modélisation en 3D de nos flores. autrement dit, faire des rotations de nos points entre X et Z pour donner de la profondeur à notre arbre. Nous avons donc repris ce que nous avons fait pour les rotations sur l'espace 2D et l'avons adapté à l'axe X et Z. Nous avons donc réussi à obtenir des rotations cycliques autour de nos points sur X et Z sans que Y bouge. Cela signifie qu'on peut désormais faire pivoter une ligne en arc de cercle autour de son point de départ et seul son point de fin bouge.

En dernière position, il nous reste à orienter nos branches. Nous avons essayé l'approche suivante : à chaque nouvelle intersection, c'est-à-dire quand notre point est le début de plusieurs branches, on positionne la première branche à 90 degré sur le plan XZ, puis pour chaque nouvelle branche, on fait une rotation proportionnelle au nombre de branches restantes. Par exemple, si on a deux branches, on aurait donc deux branches à l'opposé, une à 90 et l'autre à 180 degrés. Si on aurait par exemple au contraire 4 branches, cela serait donc successivement 45, 90, 135 et 180 degrés, vu que nos angles vont jusqu'à 180 degrés maximum.

Nous avons essayé d'implémenter une nouvelle version de notre Parser pour réaliser cela mais nous n'avons pas eu le temps de terminer avant la date finale.

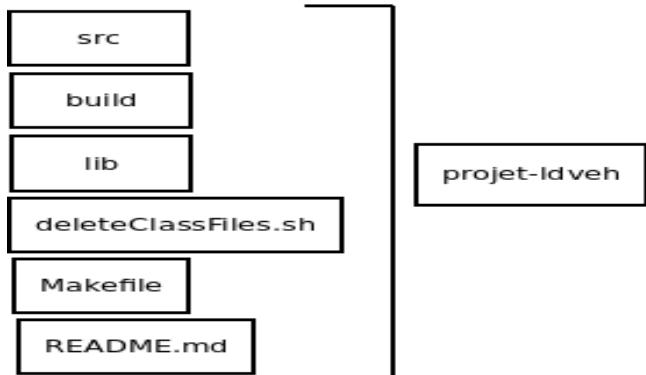
L'élaboration de notre projet en version 3D nous a demandé beaucoup plus de technicité que la version 2D. La gestion de l'interface graphique qui au départ nous semblait la plus facile, s'est avéré bien plus complexe. À présent, nous vous présenterons le rendu de notre programme au terme du processus de réalisation.

3 L'ORGANISATION ET LE FONCTIONNEMENT DU PROGRAMME

3.1 Organisation architecturale du projet

3.1.1 Le répertoire

En accédant notre répertoire de travail intitulé **projet-ldveh**, il se présente comme suit :



Le Makefile, nous permet de compiler et d'exécuter plus facilement notre code et pendant l'exécution de celui-ci, rendre le terminal plus attractif.

Notre projet se subdivise en 3 packages :

- **lSystem**,
- **geometric**,
- et **window**.

lSystem comporte les classes *Rule.java*, *Parser.java* et *Rewrite.java*. Ce package dans son ensemble regroupe les moyens de parsing et de réécriture.

geometric se compose de *Point.java*, *Line.java* et *Drawing.java* qui permettent de générer les éléments constitutifs de l'espace géométrique.

Et enfin, **window** regroupe toutes les classes concernant l'interface graphique. Ce sont *Display2D.java*, *Display3D.java*, *LSystem.java*, *Sphere3D.java*, *Cylinder.java* et *Menu.java*.

Malgré, cette organisation notre projet ne répond pas au motif d'architecture MVC car nous ne disposons pas véritablement de contrôleur.

3.1.2 Diagramme de classe

Avec le diagramme UML ci-dessous, on peut observer la structure des différents packages et les différentes relations entre les classes. Les classes ne sont pas interdépendantes. Chacune a un rôle bien précis. Cependant, au niveau de la complexité certaines méthodes de certaines classes ont un coût important en temps et en espace.

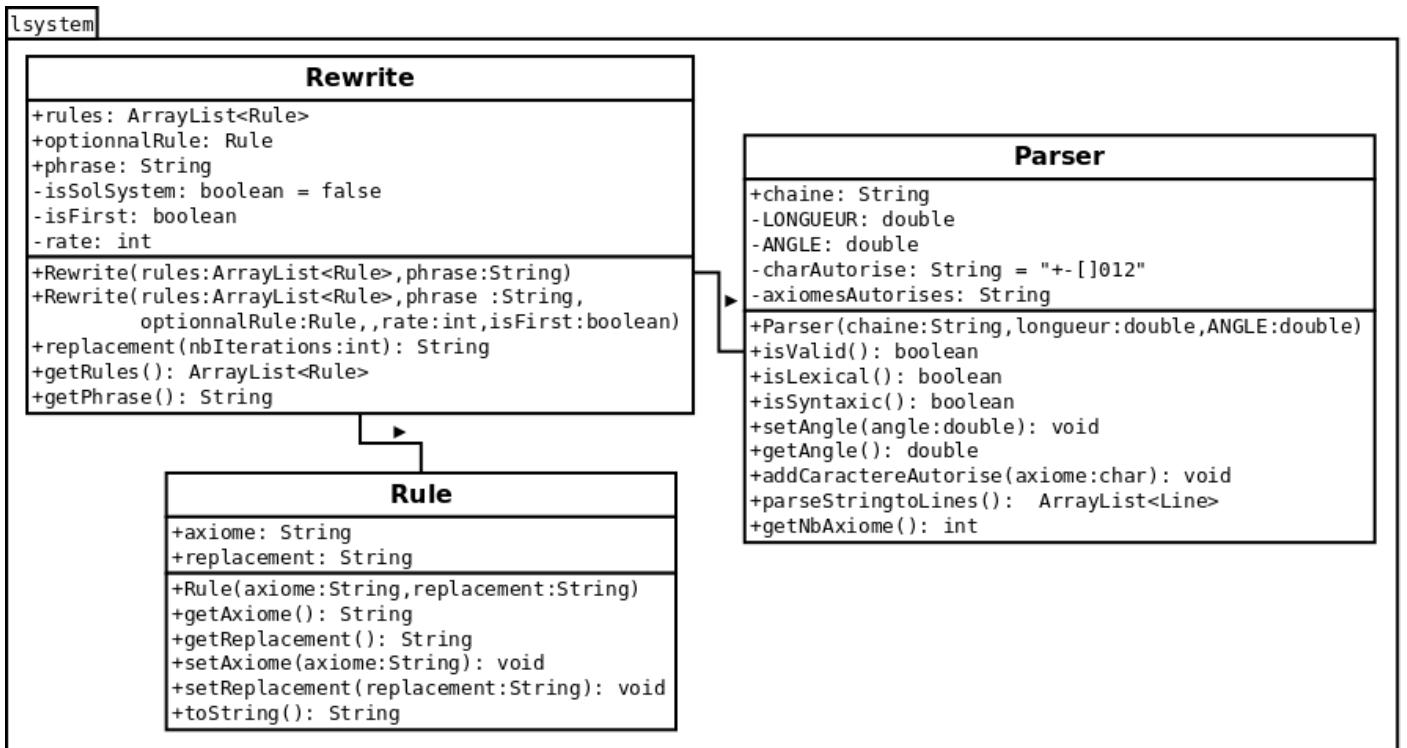


Diagramme UML du package lSystem

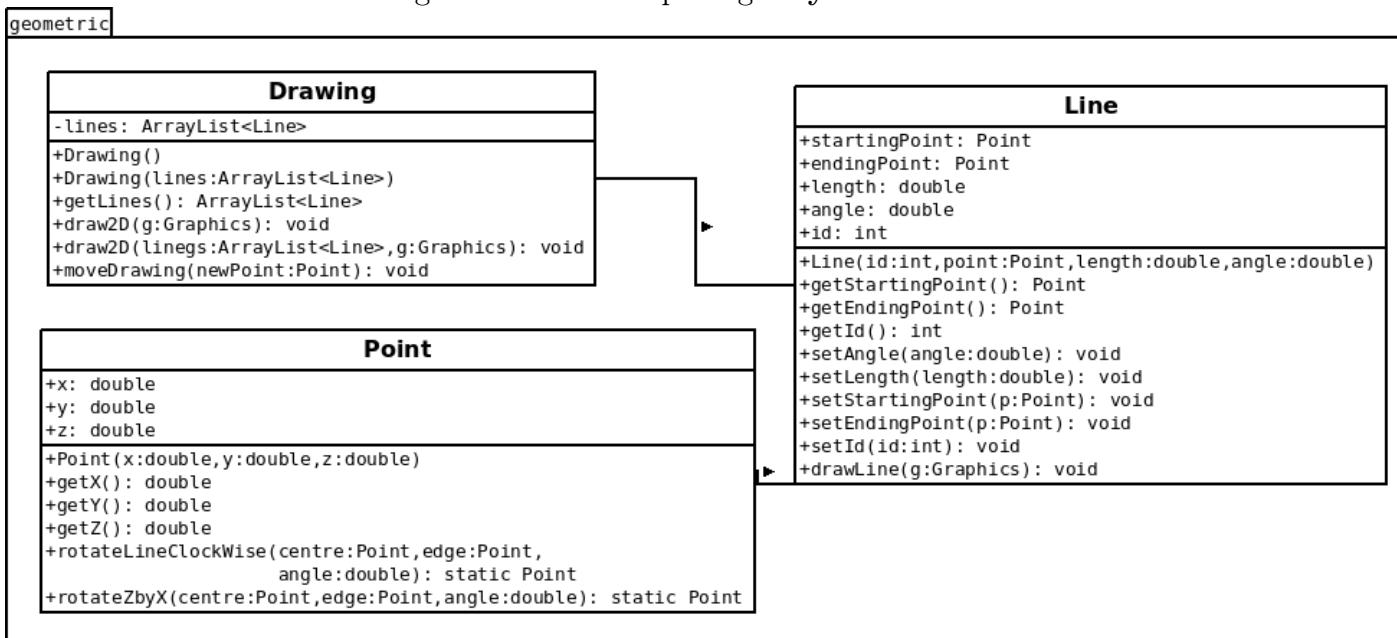


Diagramme UML du package geometric

window

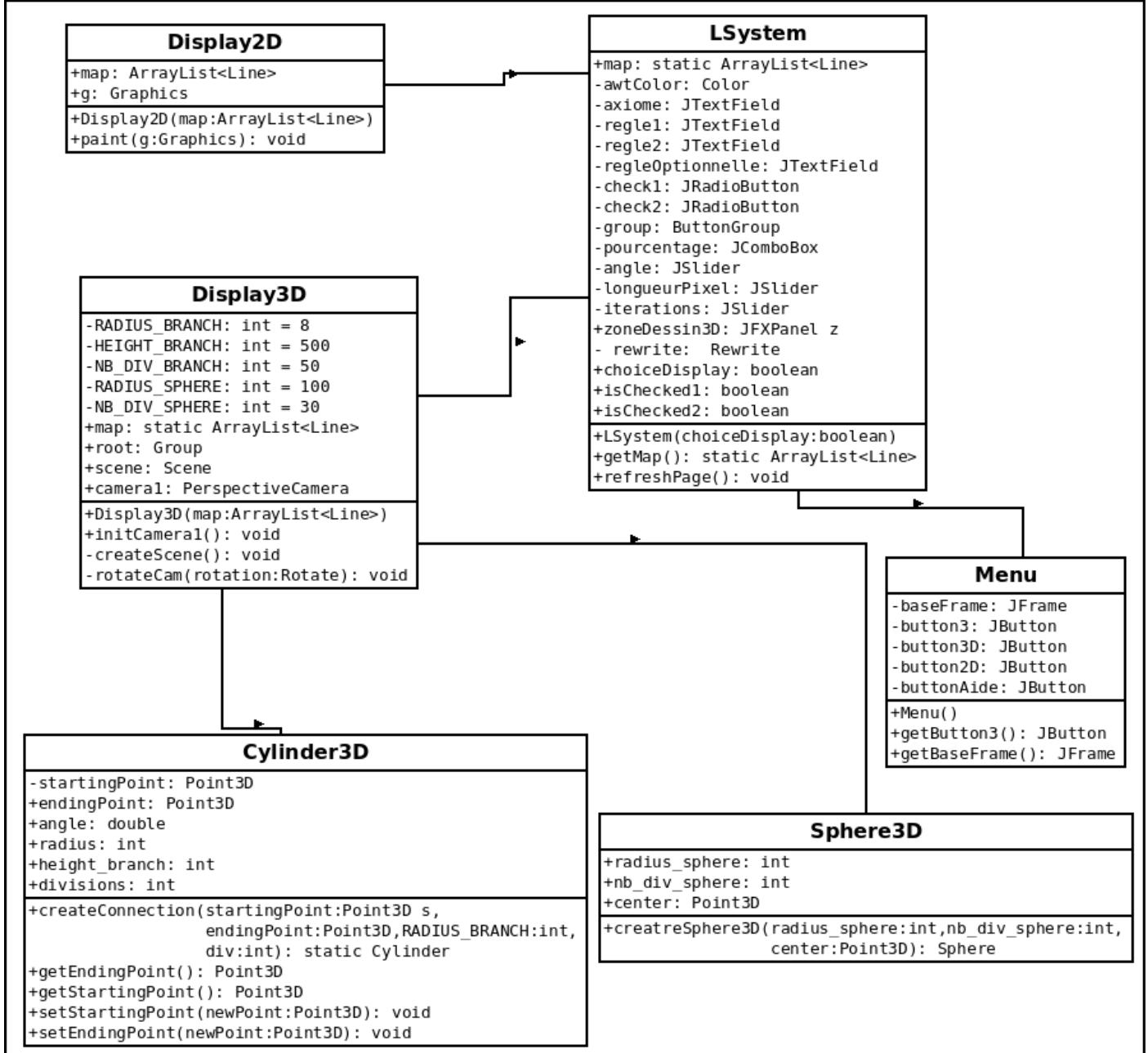


Diagramme UML du package window

3.2 Le fonctionnement du programme

3.2.1 Le manuel d'utilisation

Le générateur de flore vidéo-ludiques fonctionne de manière structurée. Les étapes suivantes doivent être bien suivies au risque de ne produire aucune figure. La génération de flore en 2D et 3D fonctionne de la même manière au niveau des barres de menu.

La manipulation de la barre de menu se fait comme suit :

- 1- Choix de son axiome de départ;
- 2- Saisie de la règle de réécriture.

Il faut respecter l'utilisation de l'alphabet ci-contre sinon il n'y aura pas de réécriture donc pas de flore.

'+' : l'angle est positif
'-' : l'angle est négatif
'F' et 'X' : tracé de ligne et marquent les axiomes
'[' : début de branche
]' : fin de branche
'0' : couleur marron
'1' : couleur vert foncé
'2' : couleur vert clair

On peut effectuer la réécriture selon le 'F' ou le 'X'
d'où les zones de texte intitulées règle 1 ou règle 2.

C'est le DOL-System.

On peut avoir un SOL-System en paramétrant la règle optionnelle et en choisissant le pourcentage de son action;

- 3- Choisir le nombre d'itérations de la règle;
- 4- Choisir l'angle de rotation des branches;\n
- 5- Choisir la longueur souhaitée pour les branches\n
- 6- Il ne reste plus qu'à générer votre flore vidéo-ludique.

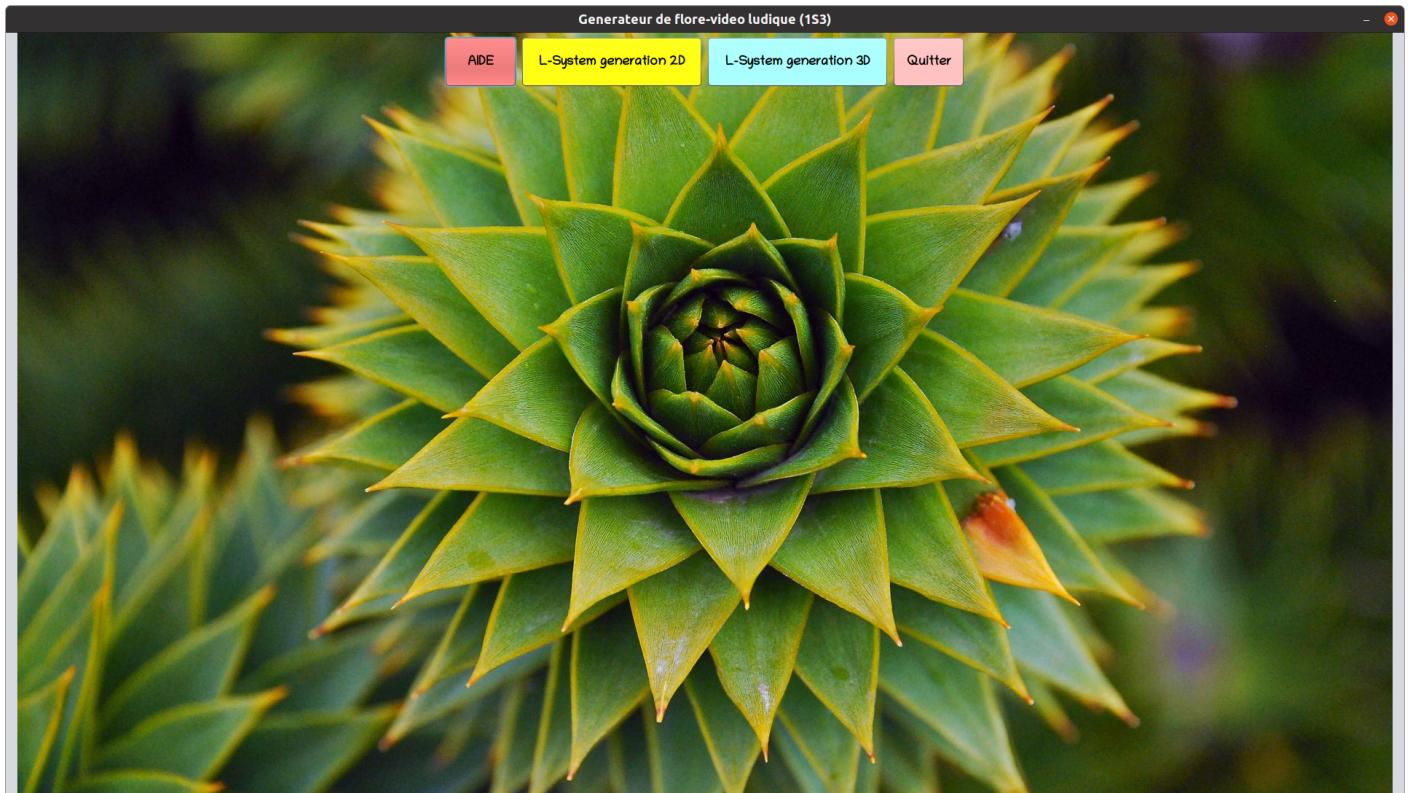
Bon à savoir:

Pour le contrôle de la caméra dans la version 3D:

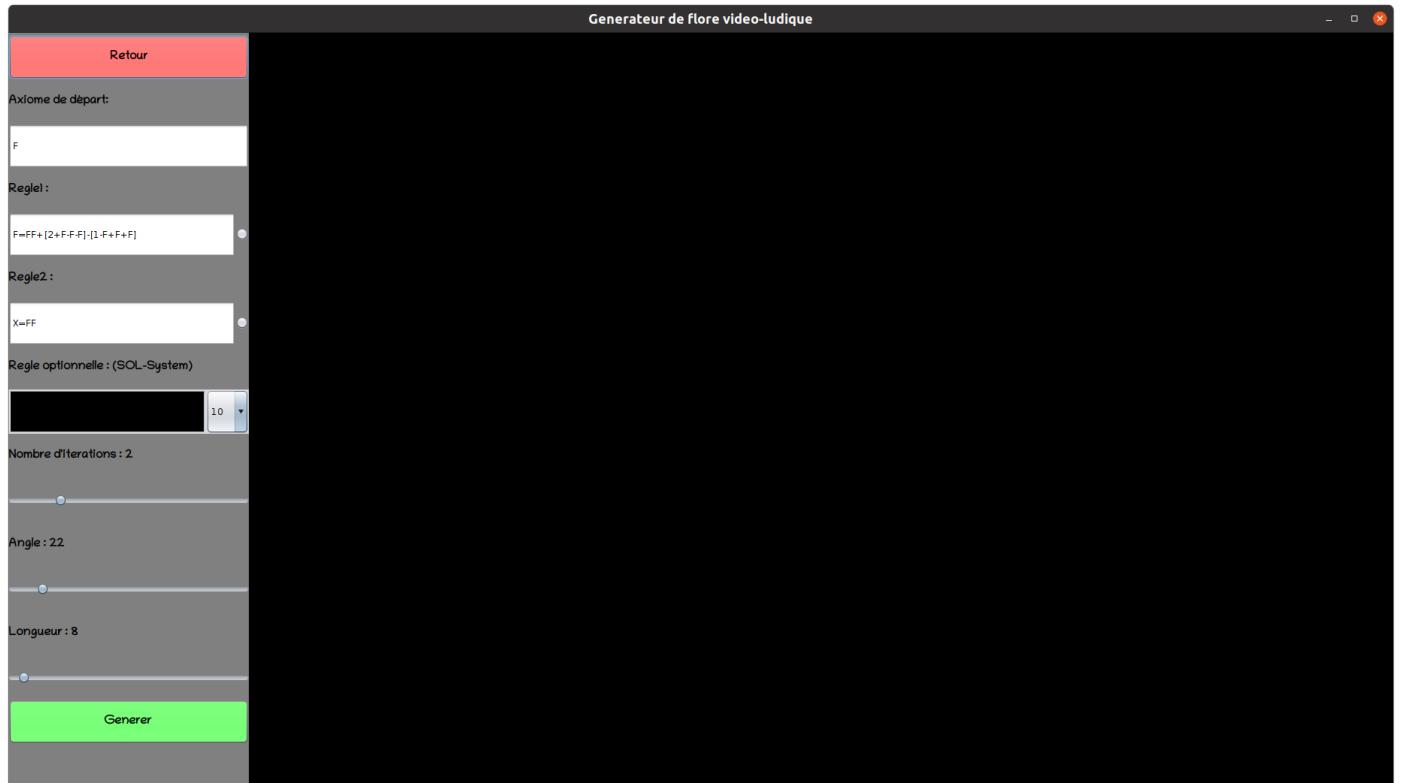
- flèche du bas: aller vers le bas
- flèche du haut: aller vers le haut
- flèche de gauche: aller vers la gauche
- flèche de droite: aller vers la droite
- molette/pavé tactile: aller vers l'avant ou l'arrière
- NUM8 du pavé tactile: incliner la caméra vers l'avant
- NUM2 du pavé tactile: incliner la caméra vers l'arrière

3.2.2 Aperçu du lancement du programme

Cette section montre en images, le lancement du programme.



Présentation de la page principale au lancement du projet. En cliquant sur le bouton L-System generation 2D, on lance la fenêtre du générateur de flore vidéo ludique en version 2D. Le bouton L-System generation 3D lance la version 3D. Le bouton Aide permet l'affichage du manuel d'aide et en cliquant sur Quitter, on ferme le programme.



Fenêtre du générateur de flore vidéo ludique composée d'une barre de menu et d'un espace d'affichage de dessin. La barre de menu permet à l'utilisateur d'entrer sa règle de réécriture et les caractéristiques souhaitées pour sa flore.

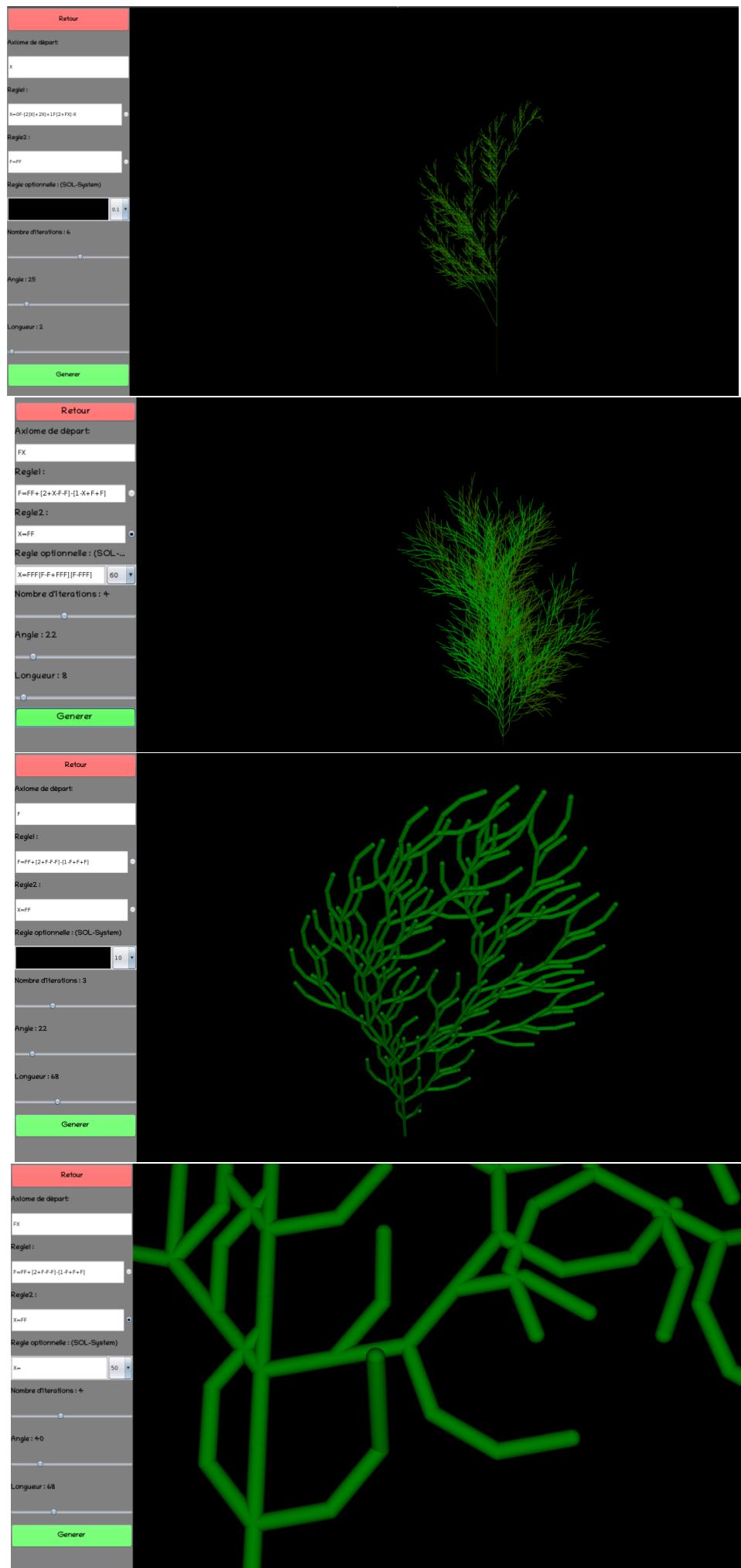


FIGURE 4 – Exemples de flores générées en 2D puis en 3D

3.3 Répartition des tâches

Étant, l'un des premiers projets avec lequel nous avons commencé ce semestre, nous avons réellement eu du mal à trouvé notre équilibre. Premièrement, nous n'étions pas tous familiers à l'usage des logiciels permettant l'hébergement des projets, notamment la forge. Nous nous envoyions nos travaux par mail ou attendions d'être ensemble pour comparer nos travaux. En outre, pendant les séances de travaux pratiques, nous nous rassemblions autour d'une seule machine pour effectuer le travail. Même si ce n'était pas la meilleure méthode de travail, celle-ci nous permettait de nous mettre d'accord et d'avancer plus ou moins au même rythme. Mais il est de noter que la partie 3D nous a un peu déconcerté. Point culminant de notre travail, il a coïncidé avec l'accumulation d'autres projets. Malheureusement pour nous, toutes les deadlines coïncidaient. La charge de travail de chaque membre a été totalement déséquilibrée.

3.4 Pistes d'amélioration

Si on passe en revue la modélisation 3D presque complète de nos flores, nous avons constaté que le programme avait pas mal de latences. Avant tout, la première amélioration consisterait à optimiser les performances, en faisant pourquoi pas du multi threading. Cela permettrait une meilleure répartition des calculs et un véritable gain en fluidité. En effet, à profondeur 6 autrement dit le nombre d'itération 6, nos arbres prennent du temps à se créer, et en version 3D ralentissent fortement le programme en raison du nombre élevé de cylindres. Une piste serait peut-être d'utiliser une autre manière pour faire nos cylindres avec par exemple, les [Triangles Mesh](#) ou alors pixel par pixel avec la classe *PixelBuffer*. Plus nous utilisons des objets déjà faits, moins c'est optimisé. Ce gain de performances aurait effectivement participé à l'amélioration de nos flores et l'environnement. Aussi, pourrait-on ajouter de nouveaux éléments à nos flores pour les rendre plus attractifs, notamment la conception de pétales pour créer des fleurs, ajouter des feuilles pour nos arbres, etc. Au niveau de l'environnement, ce serait la création d'un sol avec un ciel et donc la gestion de plusieurs flores en même temps.

Nous pourrions également générer une flore avec la caméra qui tournerait donc autour de cette flore. De plus, nous pourrions passer d'une flore à une autre en appuyant sur les flèches directionnelles. Une autre perspective serait la mise en place d'un positionnement aléatoire de nos flores sur le terrain avec un algorithme intelligent qui placerait nos flores à des distances rationnelles les unes des autres. Ainsi avec ce procédé, nous pourrions petit à petit procéder à la création de champs de fleurs, de forêts, et donc se rapprocher de la création d'environnements aléatoires.

4 Conclusion

En conclusion, nous avons réalisé un générateur de flore-vidéo ludique d'une part dans une version 2D et d'autre part dans une version 3D. Ce travail nous a demandé un grand effort de documentation, d'expérimentation mais surtout de cohésion. Ce fut l'occasion pour nous d'apprendre d'avantage sur le langage Java et ses subtilités. Mais aussi, de nous pencher davantage sur l'importance de la compréhension des objectifs. Finalement, ce projet nous aura marqué parce qu'**on ne vera plus les arbres de la même façon**.

Appendices

A Bibliographie et sitographie

Références

- [1] Aristid Lindenmayer et Przemysław Prusinkiewicz *The Algorithmic Beauty Of Plants*, 1990.
- [2] [Wikipédia](#)
- [3] [JavaFX 3D Line](#).