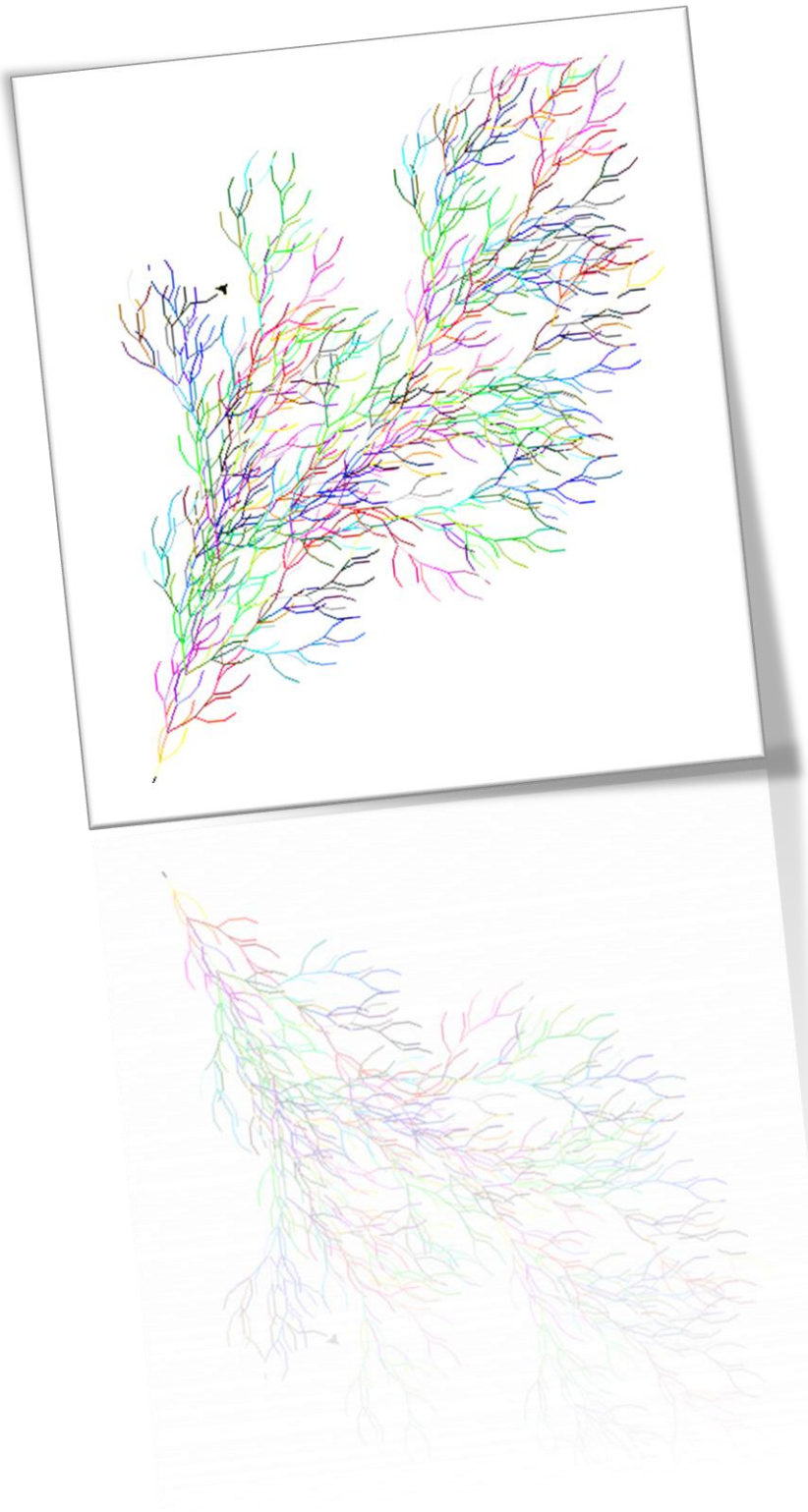
GASPAR Dorian / AUPHAN Gwendal / DOUNIES Arthur / RICHAUDEAU Alexandre
CPI103

Rapport Projet Turtle



GASPAR Dorian / AUPHAN Gwendal / DOUNIES Arthur / RICHAUDEAU Alexandre
CPI103

Table des matières

GASPAR Dorian / AUPHAN Gwendal / DOUNIES Arthur / RICHAUDEAU Alexandre
CPI103

1) Introduction et Structure du projet

En cette fin de premier semestre de seconde année du cycle préparatoire intégré, il nous a été demandé de réaliser un projet sur les L-System. Chaque partie de la consigne a été séparée dans 8 fonctions à l'intérieur du fichier main.py. Un autre fichier a été créé pour contenir les tests unitaires. Dans le dossier du projet, vous trouverez deux dossiers avec les deux fichiers mentionnés plus haut :

- example : dossier contenant des exemples d'entrée possible
- tests : dossier contenant des entrées utilisé dans les tests unitaires

Notre programme permet de traduire des entrées de L-System classique et de Context Sensitive L-System, il supporte aussi le changement de taille du trait (avec trois choix possible : petit, moyen, grand) et la couleur (avec trois couleurs possibles : rouge, vert, bleu). On s'est aussi permis de changer la vitesse de déplacement de la turtle à 0 pour que le dessin se fasse le plus vite possible.

Note :

Quand il est mentionné chemin ou emplacement dans le contexte des fichiers d'entrée et de sortie, il est question de chemin relatif au fichier main.py.

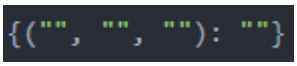
2) Présentation des fonctions et de l'algorithme

L'algorithme commence dans la fonction main() qui a pour seul objectif d'appeler toutes les autres fonctions et d'arrêter le programme si nécessaire.

La première fonction de l'algorithme est setFileName(args), elle prend comme argument les paramètres passés au programme lors de l'exécution et renvoie l'emplacement et le nom du fichier en entrée et même chose pour le fichier de sortie si spécifié, "resultat.py" sinon. Si un argument n'est pas reconnu, il notifie à l'utilisateur que l'argument entré est invalide mais cela n'affecte pas l'exécution du programme. Si la sortie de la fonction comporte un emplacement de fichier d'entrée vide, le programme est arrêté par la fonction main.

La seconde fonction est readData(inputFileName), elle prend en argument l'emplacement du fichier d'entrée et renvoie une liste contenant tous les paramètres. Les paramètres seront stockés dans cette fonction et dans le reste du programme dans une liste structurée de la façon suivante : `config = ["", {}, 0, 0, 0, []]`

Le premier index est une chaîne de caractères contenant l'axiome, le second un dictionnaire des règles, le troisième est un réel contenant la valeur de l'angle, le quatrième est un entier contenant la valeur de la taille, le cinquième index est un entier correspondant au niveau et le sixième index correspond à la liste des caractères ignorés.

Les règles sont structurées de la façon suivante : 

La clef est un tuple contenant dans l'ordre, la lettre concernée, le contexte avant la lettre et le contexte après la lettre enfin la valeur de la clef est un string contenant la nouvelle séquence.

Afin de lire le fichier en entrée, la fonction découpe le fichier ligne par ligne et regarde dans chaque ligne si il trouve un mot clef (axiome, taille, angle, …).

La fonction readData(inputFileName) appelle deux fonctions :

- fileIsValid(data) qui permet de vérifier si un fichier passé en entrée répond aux critères suivants: il y a un unique axiome, il y a une valeur d'angle, il y a une taille et au moins une règle. Si au moins un de ces critères n'est pas respecté, la fonction renverra False, ce qui aura pour effet de renvoyer la variable config avec les valeurs par défaut et causera la fonction main de stopper le programme. Pour chaque erreur qui se produit, le programme affiche ce qui ne va pas.

- readRule(i, data) qui permet de lire et d'interpréter les règles contenues dans le fichier d'entrée. Comme dit plus haut, les règles sont définies comme étant les lignes commençant par des espaces. La fonction est donc un boucle while qui, tant qu'il y a des espaces comme premier caractère continue à lire ligne par ligne. L'algorithme reconnaît la syntaxe des Context Sensitive L-system via les opérateurs "<" et ">". L'algorithme permet aussi d'interpréter les L-systems plus classiques.

  Cette fonction est appelée lorsque le mot clef détecté est "regle", il est transmis à la fonction l'index de la ligne dans laquelle est apparu ce mot clé.

La troisième fonction appelée par la fonction main est generate(config), elle prend comme argument la configuration du fichier d'entrée et renvoie l'axiome généré au niveau définis dans le fichier en entrée. Elle est constituée d'une boucle for afin de répéter les instructions le nombre de fois qu'il faut pour obtenir le bon niveau, ensuite la fonction regarde pour chaque règle quels sont les emplacements où elle s'applique avec la fonction checkContext(path, rule).

La fonction checkContext(path, rule) prend en entrée l'axiome au niveau n et une règle et regarde quels sont les endroits où cette règle peut être appliquée pour obtenir la chaîne au niveau n+1. Elle est en deux parties, s'il y a un contexte à droite OU à gauche et s'il y a un contexte à droite ET à gauche. La seconde partie est simplement la réunion de la condition à droite et de celle à gauche. S'il y a un contexte à droite ou à gauche, la fonction parcourt la plante (branches incluses) pour déterminer si les conditions sont réunies. Si c'est le cas, cette position fera partie du retour final de la fonction.

Pour revenir sur la fonction generate(config), une fois que toutes les règles ont été passées en revue, le nouvel axiome est enregistré dans newPath et tout ce qui n'as pas été changé reprend la valeur du niveau précédent (copie des constantes).

Enfin, la dernière fonction à être appelée est translate(processed, config), elle prend en entrée la chaîne qui vient d'être générée au niveau demandé et transcrit les symboles en instructions turtle. Pour ce faire, la fonction utilise un dictionnaire contenant toutes les équivalences symbole/code et les ajoutés les uns aux autres en plaçant des retours à la ligne entre chaque instruction pour donner le résultat final qui est renvoyé à la fonction main, se chargeant de l'affichage et de la sauvegarde du fichier.

3) Tests unitaires

Concernant les tests unitaires, nous avons choisi d'utiliser la librairie unittest. En effet, unittest permet de définir des cas de tests, avec notamment un jeu d'assertions assez complet et pratique. Un scénario de test est créé comme classe-fille de unittest.TestCase. Par convention, chaque fonction de cette classe commencera par "test", ceci permet de reconnaître les fonctions étant des tests.

A chaque test, nous faisons alors appel à 2 méthodes différentes :

- assertEqual pour vérifier un résultat attendu
- assertTrue() ou assertFalse() pour vérifier une condition

```
def test_fileIsValid(self):
    with open("tests/buisson.txt", "r") as f:
        self.assertTrue(fileIsValid(f.read()))
    with open("tests/fougère_sans_taille.txt", "r") as f:
        self.assertFalse(fileIsValid(f.read()))
```

Dans le cas ci-dessus, on teste la validité des fichiers .txt. Dans 1 premier temps, on vérifie que le fichier buisson est valide, dans le 2ème on vérifie que fougère est faux.

4) Bonus

Comme dit dans l'introduction, nous avons choisi d'implémenter quelques bonus, parmi eux, les variantes plus complexes de L-system, déjà mentionnées dans la section 2. Voici le détail des autres bonus que nous avons réalisés :

- Ajout de commutateurs : le -i permet de spécifier le chemin vers le fichier en entrée, s'il est absent, le programme se stoppe et affiche l'erreur, il y a aussi le commutateur -o qui permet de spécifier le chemin et le nom vers le fichier de sortie.
- Ajout d'options pour contrôler la taille du trait tracé par la turtle: s pour petit (1px), m pour moyen (3px) et l pour large (6px)
- Ajout d'options pour contrôler la couleur du tracé : r pour rouge (#FF0000), g pour vert (#00FF00) et b pour bleu (#0000FF)

5) Répartition des tâches

Afin de collaborer le plus simplement possible à distance, nous avons utilisé plusieurs outils tels que GitLab et LiveShare. GitLab permet de partager et de versionner le code entre les collaborateurs (voici le lien: https://gitlab.etude.eisti.fr/gaspardori/projet-l-system). LiveShare permet de coder à deux en même temps sur un même projet et permet donc d'avoir une meilleure productivité. Nous avons découpé le projet en deux : l'exercice 1 et l'exercice 2. Alexandre Richaudeau et Arthur Dounies se sont chargés de l'exercice 1 dans un premier temps et Gwendal Auphan et Dorian Gaspar se sont chargés de l'ajout des fonctionnalités de l'exercice 2. Nous avons choisi cette répartition du travail par rapport au niveau, en effet, Arthur et Alexandre ont commencé le Python cette année alors que Dorian et Gwendal le pratiquent depuis plusieurs années.

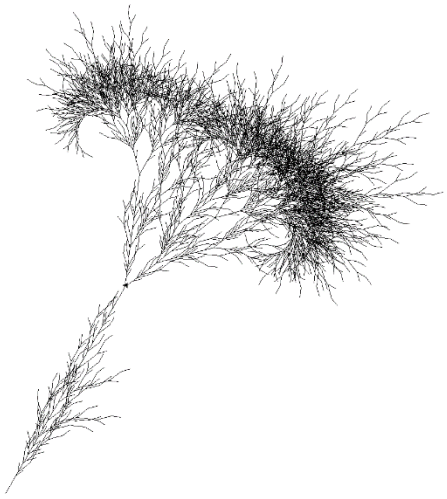
6) Bilan du Projet

Ce projet a été une expérience enrichissante où l'on a pu découvrir de nouvelles techniques et une nouvelle librairie graphique ainsi que des tests. La compréhension du sujet n'a pas été facile puisque tout n'était pas très détaillé. Nous avons alors progressé par ajout de fonctions au cours du projet. L'algorithme en lui-même n'était pas très compliqué, la difficulté résidait dans l'ajout des Bonus où nous devions vérifier pas mal de conditions.

Le dessin des figures est assez satisfaisant et nous avons pu prendre plaisir à voir prendre forme différentes fractales ou arbres de la vie. Nous sommes assez contents du rendu du projet car il est efficace et compact, de plus le travail de groupe est toujours un vrai engouement.

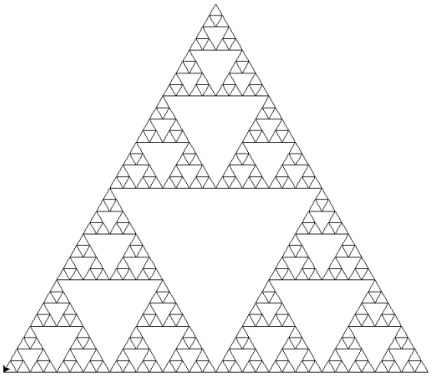
7) Exemples

Context-sensitive



```
axiome = "aYaYaY"
regles =
    "X<X>X=X"
    "X<X>X=X"
    "X<X>Y=Y[+aYaY]"
    "X<Y>X=Y"
    "X<Y>Y=Y"
    "Y<X>X=X"
    "Y<X>Y=YaY"
    "Y<Y>X=X"
    "Y<Y>Y=X"
    "+=-"
    "-=+"
angle = 22.5
taille = 5
niveau = 29
```

Triangle de Sierpinsky



```
axiome = "aXa--aa--aa"
regles =
    "a=aa"
    "X=--aXa++aXa++aXa--"
angle = 60
taille = 10
niveau = 5
```