Les L-Systèmes

Lim Victoria et Lingeswaran Aranniy
`projet de XML`

2023-2024

1 Introduction

Dans le cadre de notre première année de Master, nous avons eu à réalisé un projet portant sur les L-Systèmes.

Ce rapport détaille nos choix de conception et d'implémentation, la répartition des tâches, les difficultés que nous avons rencontrées et les bugs non résolues !

2 Structure du Projet

- `XSD/` : `l-systems.xsd, tortue.xsd, traceur.xsd`
- `XSL/` : `svg.xsl, tortue.xsl, traceur.xsl`
- `generated/` : (fichiers générés)
- `l-systems.c`
- `l-systems.csv`
- `README`
- `makefile`
- `ce rapport`

3 Description du projet

3.1 Etape préliminaire

Notre première approche a été de décomposer toutes les informations du L-système en des balises distinctes. On avait donc une balise racine `<lsystems>` avec à l'intérieur une balise `<definition>` qui représentait chaque L-système. Dedans, on avait une balise pour le `<nom>`, l'`<alphabet>` du système représenté par plusieurs balises `<symbole>`, l'`<axiome>` également représenté par des balises `<symbole>`, et enfin la balise `<règles>` qui contenait - à nouveau - une balise `<symbole>` avec son association `<image>` et `<commande>` pour représenter la transformation du symbole.

Cependant, cette approche s'est avérée être trop complexe et peu pratique pour la suite, sans oublier les nombreuses redondances des balises symboles. De même, après une discussion avec notre chargé de TP, nous avons constaté que la balise `<alphabet>` n'était pas nécessaire, car toutes les informations concernant les symboles étaient déjà détaillées dans la balise `<axiome>`.

Nous avons ainsi simplifié davantage la structure XML en regroupant les informations relatives aux symboles et aux règles de production au sein d'une seule balise pour chaque L-système et en définissant des informations en tant qu'attributs.

- La balise racine est `<lsystems>`.
- À l'intérieur de `<lsystems>`, la balise `<lsystem>` avec un attribut `id` qui contient :
  - `<nom>`
  - `<axiome>` : qui contient les symboles initiaux de l'axiome.
    - Les symboles de l'axiome sont définis avec la balise `<symbole>`.
  - `<regles>` : qui définit les règles de production sous forme de balises `<regle>`.
    - Chaque règle utilise des attributs pour représenter les transformations :
      - `image` : pour le nouvel ensemble de symboles après la transformation.
      - `commande` : qui indique l'action à effectuer (e.g., `LINE`, `TURN`).
      - `angle` : qui spécifie l'angle de rotation.

Cette approche offre une structure plus concise et plus facile à lire, tout en préservant toutes les informations nécessaires pour définir un L-système.

Après cela, nous avons commencé à écrire le XSD validant notre format XML. Le seul problème rencontré est celui évoqué dans la section Difficultés Rencontrées du rapport. Nous avons simplement ajouter les contraintes qui nous semblaient nécessaire comme définir l'ordre des balises et leur nombre, préciser la nature des symboles et images ainsi que les valeurs min et max de l'attribut angle avec un restriction pattern. On a également énuméré les commandes possibles pour ne pas avoir de mauvaises implémentation, défini les élements qui seraient uniques comme une règle (pour éviter la redondance) ou l'id d'un L-Système. Cette partie a été très facile à réaliser. De même pour la création d'un xml complet de L-Systèmes à partir d'un programme en Language C.

3.2 La Tortue

Le template de la `racine` initialise le système en récupérant ses règles choisi à partir des paramètres d'entrée (nom et nombre d'itérations), puis il appelle le template `Iterate` pour traiter les symboles du système.

Le template `Iterate` utilise un `xsl:choose` pour gérer deux cas : lorsque le nombre d'itérations n'a pas été atteint et lorsque toutes les itérations ont été effectuées. Dans le premier cas, il applique les règles du L-système pour remplacer chaque symbole par son image correspondante, puis s'appelle récursivement avec les nouveaux symboles générés et l'itération suivante. Dans le second cas, il parcourt la séquence finale en traitant chaque symbole individuellement. Lorsqu'il rencontre les symboles '[' et ']', il génère les actions

d'attributs de commande `STORE` et `RESTORE` respectivement pour sauvegarder et restaurer l'état de la tortue. Pour les autres symboles, il utilise les commandes indiquées dans la balise `<regle>` du système.

On a également implémenté une fonction pour gérer les opérations sur les string, telles que la fonction `fun:explode()` qui divise une chaîne en caractères individuels.

3.3 Le Traceur

Le template `tortue` commence par la déclaration d'une variable `nbr_actions` pour compter les actions de notre tortue. Il définit ensuite des variables (x, y, angle, save) pour suivre sa position, son orientation et son état puis initialise le traçage avec une commande MOVETO à la position (0, 0), et ensuite appelle récursivement le template `Action`, en passant les paramètres nécessaires pour itérer sur chaque action et mettre à jour les données de la tortue.

Le template `Action` traite chaque action de la tortue. Tant que l'itération en cours est inférieure au nombre total d'actions, le template récupère l'action courante et utilise `xsl:choose` pour déterminer son type de commande (`TURN`, `LINE`, `MOVE`, `STORE`, `RESTORE`). Chaque commande est ensuite traitée en conséquence :

- `TURN` additionne l'angle courant à la nouvelle.
- `LINE` calcule de nouvelles coordonnées pour dessiner une ligne.
- `MOVE` déplace la tortue sans dessiner..
- `STORE` enregistre l'état actuel.
- `RESTORE` restaure le dernier état enregistré.

Après chaque action, le template s'appelle lui-même avec les paramètres mis à jour pour traiter la prochaine action. Les calculs ont été réalisés en utilisant des formules trigonométriques.

On a sauvegardé l'état intermédiaire en utilisant un système de pile. Lorsque l'action `STORE` est rencontrée, les coordonnées actuelles de la tortue ainsi que son angle sont stockées dans un élément `state` et ajoutées à la pile des états sauvegardés. Cette pile est représentée par la variable `save`, qui est une séquence de ces états. Lorsque l'action `RESTORE` est rencontrée, la tortue récupère les coordonnées et l'angle du dernier état sauvegardé et les retire simplement de la pile. Ainsi, la tortue peut restaurer son état précédent à tout moment en défilant simplement à travers cette pile.

3.4 Le SVG

Ici, pour générer un fichier `.svg`, nous avons opté pour une approche modulaire en divisant le code en trois templates distincts.

Le premier template est dédié à l'élément `<traceur>` qui définit la structure de base du SVG en incluant les balises `<svg>` et `<path>`. Un attribut `d` est ajouté à l'élément `<path>` en appelant le template `tracage` avec la liste des actions comme paramètre.

Le template `tracage` itère ensuite sur chaque action à l'aide d'une boucle `for-each`, appelant la dernière template, `trace-action`, pour chaque action.

Cette dernière template est chargée de gérer les cas de déplacement (`MOVETO`) et de ligne (`LINETO`), en concaténant les coordonnées correspondantes avec les préfixes "M" ou "L" selon le cas. La séparation de ces trois templates facilite l'ajout de nouvelles actions à l'avenir. Par exemple, si une action de courbure de ligne était ajoutée, il serait alors plus facile de l'intégrer en l'ajoutant uniquement dans la troisième template.

Pour garantir que le dessin obtenu soit bien au centre de l'écran, nous avons inclus des variables au début du fichier qui calculent respectivement les coordonnées minimales et maximales des actions de la tortue. Ces coordonnées sont multipliées par 5 pour agrandir les dimensions du dessin. Ensuite, deux autres variables sont utilisées pour déterminer la largeur et la hauteur du dessin en soustrayant les coordonnées minimales des coordonnées maximales. Cela permet donc de centrer le dessin.

4 Répartition et Chronologie des Tâches

- **1ère étape : Modélisation en XML d'un L-système**
  Période : Début du projet jusqu'au 6 mars. *Répartition* : Victoria et Aranniy
- **2ème étape : XSD du L-système**
  Période : Du 6 mars au 8 mars. *Répartition* : Aranniy
- **3ème étape : Programme C générant le fichier XML complet**
  Période : Mars-Avril. *Répartition* : Aranniy
- **4ème étape : XSD de la Tortue**
  Période : Avril. *Répartition* : Victoria
- **5ème étape : XSD du Traceur**
  Période : Avril. *Répartition* : Aranniy et Victoria
- **6ème étape : XSLT de la Tortue**
  Période : Mai. *Répartition* : Victoria
- **7ème étape : XSLT du Traceur**
  Période : Mai. *Répartition* : Victoria
- **8ème étape : XSLT du SVG**
  Période : Mai. *Répartition* : Aranniy et Victoria
- **9ème étape : Rapport**
  Période : En continu pendant le projet. *Répartition* : Aranniy

5 Difficultés Rencontrées

- **Se mettre d'accord sur les formats XML** : Il nous a fallu du temps pour nous décider sur les formats XML. En effet, nous avions toutes les deux deux avis totalement différents : l'une préférait mettre toutes les informations dans des attributs,

tandis que l'autre proposait d'utiliser des balises pour chaque information. Il a fallu du temps pour trouver une solution qui nous satisfaisait toutes les deux.

- **Choix des contraintes XSD** : Nous voulions que nos schéma XSD soit le plus stricts possible. Cependant, nous avons rapidement constaté que certaines contraintes que nous voulions imposer étaient impossibles à implémenter. Par exemple, nous souhaitions que chaque lettre de l'axiome soit présente parmi les symboles possibles (ex : pas de « C » dans l'axiome si « C » n'est pas un symbole). Cependant, cela s'est avéré être un mauvais choix car certains symboles qui ne sont pas dans l'axiome peuvent ne jamais être produits à partir de l'axiome (par exemple, un axiome A, avec des règles A -> B, B -> A, et une règle pour un symbole C qui ne sera jamais utilisée).

- **Comprendre la structure de la Tortue et du Traceur** : Nous avons eu du mal à assimiler ce que nous devions faire pour cette partie et comment la réaliser. La gestion de l'état intermédiaire notamment nous a causé de nombreux soucis pour son implémentation. La solution adopté a été d'utiliser un système de pile.

- **Centrage du SVG** : Nous avons mit du temps à comprendre comment redimensionner l'image généré.

- **Pression des examens et autres projets** : La pression des examens et des autres projets a également eu un impact. Cela nous a contraints à délaisser temporairement ce projet.

6 Bugs Non Résolues

- **Tortue** : Chosir un nombre trop élevé d'itérations peut causé des problèmes de mémoires selon le système choisi.