Rapport Chromat'Ynk

Professeur encadrant: Zaouche DJAOUIDA

Olivier Compagnon–Minarro
Clément Delamotte
Matthias Franchini
Célian Mignot
Hénoch Xu

Année 2023-2024 - ING 1 GI 3 - Groupe 6

Tables des matières

Introduction

Nous groupe est composé de 5 étudiants de ING1 GI3 :

- Olivier Compagnon–Minarro
- Clément Delamotte
- Matthias Franchini
- Célian Mignot
- Hénoch Xu

Lors de la réunion de présentation, notre groupe hésitait entre le projet CodYngame et Chromat'Ynk. Après discussion, la majorité a choisi de faire Chromat'Ynk en raison des capacités techniques demandées par ce sujet, qui nous semblaient plus intéressantes par rapport aux autres sujets proposés.

Présentation du sujet

Le but de ce projet est de faire un programme qui va permettre de dessiner des formes quelconque à partir de ligne sur un écran à partir d'instructions simples, comme avancer d'un certain nombre de pixels, tourner d'un certain angle, … . Le tout en pouvant changer l'épaisseur et la couleur des traits dessinés. Ce projet nécessite donc la création d'un langage d'instructions (à l'aide d'une grammaire que l'on a créé), ainsi qu'un interpréteur de ce langage. Le programme pourra sauvegarder et charger des créations, et les dessiner en temps réel ou alors exécuter les instructions à un rythme donné (dont étape par étape).

Démarche

Notre programme comporte 2 points principaux : une grammaire (et son analyse) ainsi que l'affichage graphique des dessins.
L'application fonctionne de la façon suivante: l'utilisateur entre ses commandes dans une zone de texte sous un canevas où apparaîtront les dessins. Les commandes sont ensuite analysées et traitées par un analyseur lexical et sémantique, qui décident de la validité de la commande. Si elle est acceptée, la commande est alors enregistrée dans une classe qui contient ses paramètres et est envoyée au Controller qui exécute l'opération souhaitée. S'il y a une erreur, elle est affichée dans une console se situant en haut à droite de l'application et expliquant la cause de l'erreur. Certaines erreurs peuvent modifier l'interprétation du script et donc celui-ci n'est alors pas du tout exécuté.

Gestion du projet

Lors de la réalisation de ce projet, nous avons suivi le même procédé. Nous nous retrouvions tous les jours soit en présentiel ou en distanciel (Discord) pour avancer collectivement sur notre projet afin de ne laisser aucun membre de notre équipe sur le côté.
Nous avons utilisé la fonctionnalité “Code with me” de Intellij pour nous permettre de coder sur les mêmes fichiers, généralement par binôme/trinôme, et de nous faciliter la tâche si l’un d’entre nous rencontrait des difficultés.
Pour tout ce qui est de la partie orienté objet, nous avons fait des croquis de diagrammes sur feuilles que nous avons par la suite refait sur StarUML pour un rendu plus propre.
Pour la sauvegarde de l’avancement de notre projet, nous avons utilisé GitHub.

Première semaine

Pour commencer, nous nous sommes tous réunis en présentiel juste après le choix définitif du sujet et nous avons commencé par réaliser un diagramme des cas d’utilisation et de classe afin de nous donner une meilleure idée de comment nous allions nous organiser et de qu’est ce qui serait nécessaire d’implémenter afin de répondre au cahier des charges. Nous avons ensuite démarré le projet sur Intellij et démarré l’implémentation du code.
Notre première étape fût l’implémentation d’un interpréteur créé par nous même ainsi que des différentes classes de base comme celle pour les curseur et variables ainsi que les classes pour la gestion de leurs attributs et contextes. Lors de la réunion de suivi avec notre tutrice, nous avons échangé pour voir si nous avions compris ce qu’il nous était demandé dans le cahier des charges. Un de ses conseils étant “de ne pas réinventer la roue” en ce qui concerne l’analyse grammaticale, nous avons revu la manière de faire notre interpréteur. Durant le week-end nous avons trouvé une solution qui existait, à savoir ANTLR4, pour créer des classes à partir de notre grammaire pour analyser celle-ci.

Deuxième semaine

Nous avons donc décidé de changer la manière dont nous implémentons la grammaire, réalisant qu’il existait des outils bien plus pratiques et beaucoup plus faciles à prendre en main que ce sur quoi nous nous dirigions. Au début, nous avons fait un prototypage de grammaire sur feuille en utilisant nos connaissances en théorie des langages. ANTLR nous permet de construire notre propre grammaire et de la traduire en un lexer (analyseur lexical) et un parser(analyseur sémantique) comprises par java qui seront ensuite utilisées pour établir nos propres règles : lorsque l’analyseur reconnaît une phrase qui est correcte, il

détermine automatiquement la syntaxe concernée et il va ensuite chercher pour cette syntaxe donnée quels sont les consignes que nous avons entrées dans le cas reconnu. Par exemple, si l'utilisateur rentre "FWD 5", l'analyseur reconnaît qu'il s'agit d'un "forwardStatement" et va chercher la méthode qui lui est liée, qui ici, avance le curseur de 5 pixels dans la direction qu'il pointe.

La majeure partie de la semaine à consisté en la compréhension du fonctionnement des différentes classes générées à partir de la grammaire afin de pouvoir les utiliser et en l'implémentation des différentes instructions.

Nous avons rencontré plusieurs défis pour l'implémentation des instructions, dont les principaux ont été de détecter les erreurs et les gérer étant donné que l'outil ANTLR a un cas qui retourne une "extraneous input" ce qui arrive quand une instruction n'a pas de paramètres mais que d'autres instructions sont écrites en dessous. De plus ANTLR va chercher ce paramètre parmi les paramètres des autres instructions, par exemple, lorsque l'on écrit FWD puis BWD 4, ANTLR reconnait FWD 4 comme une instruction et BWD est compris comme un intrus au milieu de l'instruction. Pour pallier ce problème, on a rajouté les méthodes qu' ANTLR utilise pour notifier cette erreur ainsi que d'autres (voir syntaxError) et ainsi générer une erreur dite critique puisque l'input n'est pas reconnu comme on le souhaite.

De plus, la gestion des pourcentages à aussi poser problème puisque notre première grammaire ne les reconnaissait pas efficacement, typiquement dans le cas où il y avait plusieurs pourcentages entrés. Il a donc fallu rajouter dans celle-ci une spécification pour connaître la position de ceux-ci (s'ils sont en premier paramètre ou deuxième par exemple). Une fois ceux-ci reconnus, il suffisait alors simplement de retourner les dimensions du canevas concerné (dimension sur la largeur pour le premier paramètre et dans la hauteur pour le deuxième) multiplié par le pourcentage entré en tant que paramètre dans les instructions.

A part cela, les expressions arithmétiques et booléennes nous ont donné un peu de fil à retordre puisque les opérations et comparaisons entre variables avaient des types différents mais l'interpréteur était incapable de déterminer le type des variables, le problème venait en fait de la grammaire dont on a prioriser la gestion des variables en écrivant une règle dédiée pour gérer les différents cas d'erreurs.

A la fin de la semaine, nous avons pratiquement fini l'implémentation des méthodes et nous nous sommes mis à réaliser l'application en javaFX à l'aide du scene Builder.

Troisième semaine

Lors de la dernière semaine, nous avons terminé l'implémentation des instructions ainsi que la gestion des erreurs. Nous nous sommes donc mis à plein temps sur le développement de la partie en javaFX. Nous avons rencontré plusieurs problèmes dont la difficulté de faire le pont entre la partie graphique et la partie technique développée précédemment. Mais nous avons aussi eu de nombreuses difficultés sur la partie graphique en elle-même. Pour commencer, nous ne savions pas comment utiliser un canevas pour dessiner les lignes créées par notre console : les lignes suivant la première avaient les mêmes caractéristiques alors que nous modifions leur aspect.
Les problèmes situés au-dessus sont également arrivés pour les curseurs. Pour les images des curseurs, nous avons alors utilisé un ImageView, liée à un Pane créé dans SceneBuilder, nous permettant de modifier plus facilement l'image du curseur présent sur le canevas. Pour les lignes, nous utilisons un GraphicsContext, nous sommes alors allés lire la documentation qui lui est liée pour mieux comprendre son fonctionnement et nous avons ainsi réussi à régler nos problèmes sur les lignes.
Suite à la dernière réunion avec notre encadrante, cette dernière nous a suggéré un diagramme de classe qui est plus orienté objet, cette suggestion nous a surpris au niveau du nombre de classe à créer mais cette méthode de diviser pour mieux régner nous a aidée à mieux cerner le sujet. Et par une partition-fusion des classes nous avons pu optimiser le projet.

Conclusion

Ce projet nous a permis d'utiliser nos compétences que nous avons pu apprendre au cours de notre première année d'ingénieur, aussi bien du premier semestre que du deuxième semestre et approfondit notre connaissance sur le JAVAFX ainsi que découvert de nouvelles bibliothèques de JAVA. Ce projet fut notamment l'occasion pour nous de comprendre comment il était nécessaire d'organiser une équipe pour produire un résultat convenable dans un délai imposé.