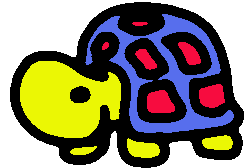
|  |  |  |
| --- | --- | --- |
| Rapport de projet | July 7  2012 | |
| Le projet Logo fait parti des projets de compilation du cours de M. Dechelle, enseignant à l’ESIEA. Nous avons donc réalisé un interpréteur Logo capable de lire et comprendre une grammaire propre au language Logo. | |  |



Contents

[Choix du sujet 2](#_Toc329535088)

[Introduction aux technologies utilisées 3](#_Toc329535089)

[Première approche 4](#_Toc329535090)

[Solution technique 5](#_Toc329535091)

[Problèmes rencontrés 7](#_Toc329535092)

[Conclusion 8](#_Toc329535093)

## Choix du sujet

Au départ nous hésitions entre 2 sujets le sujet avec le tableur CSV et l’interpréteur logo java.

Nous hésitions entre ces 2 sujets à cause du langage de programmation que nous apprécions mais ce qui nous a fais trancher est que dans un projet précédent nous avions déjà utilisé un fichier CSV ou nous devions traiter les données.

Donc voulant découvrir de nouvelle technologie nous avons optés pour l’interpréteur logo.

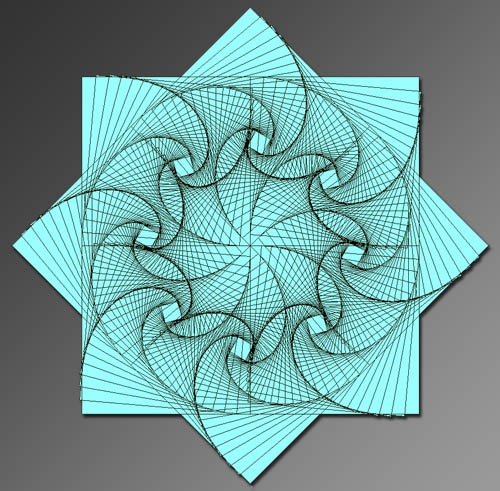
## Introduction aux technologies utilisées

Je vais faire une brève introduction du langage logo et surtout de l’aspect que nous avons travaillé.

Le langage logo est un logiciel de dessin qui a un curseur avec 2 paramètres la position en X Y ainsi que l’angle de son curseur.

Il est utilisé pour faire des figures géométriques et le plus souvent des fractals.

Exemple :



Il possède plusieurs fonctions comme le changement de la couleur du tracé ainsi que le fond d’écran.

Moi et Nicolas avons utilisés JavaCC qui nous facilite la tache car on l’utilise pour générer un parser.

Nous nous sommes vite rendu compte que si nous ne l’utilisions pas pour le langage logo nous allions devoir faire plusieurs classes très lourdes qui se composeront d’un switch case énorme. Ainsi qu’un temps d’exécutions loin d’être exceptionnel.

Au lieu de cela nous utilisons une grammaire sous l’extension de fichier .jj . Avec cette grammaire javaCC produit un parser descendant récursif.

Pour l’interface graphique nous avons utilisés swing qui est pour nous déjà connue.

## Première approche

Suite à cela nous avons commencé nos recherches sur internet pour appréhender ce qu’était le langage logo ainsi que les différentes réalisations que l’on pouvait faire avec.

Nous n’avons pas réalisé la difficulté de ce projet au premier abord.

Après avoir appréhendé le sujet nous avons commencé à coder pour avoir une grammaire qui puisse lire des instructions logo simple et qu’ils nous les retournaient de la bonne manière sans erreur.

Une fois cette étape passée, nous avons décidé de nous atteler à la partie graphique du programme java qui s’exécutera comme l’interpréteur logo.

Après cela on a décidé de pousser la complexité des instructions que le langage logo nous offrais et tentions de les interprétés.

Nous avons interpréter une partie de ces instructions pour montrer que nous pouvions le faire.

Suite à cela on a ajouté au programme une partie fun que vous découvrirez en le lançant.

## Solution technique

* Parseur

Pour parser notre chaine nous avons utilisé JavaCC. Il possible de définir avec cet analyseur syntaxique descendant récursif, une grammaire ou un ensemble de règles syntaxique que la chaine en entrée doit respecter.

On peut scinder le parseur en 3 parties :

* Les opérations : (Ex : REPETE n [] ) peuvent être composées d’opération et de fonctions. L’unique opération implémentée est « REPETE » qui exécute n fois l’intérieur des crochets. Nous pouvons donc voir sous forme d’un arbre les opérations.
* Les fonctions : (Ex : AV 14 | RE 85 ) possèdent un argument de type entier et peuvent dessiner des figures sur la fenêtre.
* Les variables (Ex : x=4 AV :x ) sont déclarées au début de la chaine et stocké dans une HashMap. Vous devez terminer la déclaration des variables par un point virgule.
* Les procédures ( Ex : Pour :Carré Fin) sont déclarées et peuvent être appelé dans le programme. C’est fonctionnalité n’a malheureusement pas été implémenté.
* IHM

Lors de la récupération de notre graphe d’objet, il est important de le manipuler et de l’exécuter dans coté IHM pour dessiner ce qui est souhaiter. Etant donnés que nous ne sommes pas des spécialistes SWING, je tiens à citer les travaux de Christophe Dujardin sur developpez.com qui nous ont beaucoup aidé dans la mise en place de la solution technique.

La classe « interpreteur.java » se base sur le patron de conception « Interpeter ». L’interpréteur fait la correspondance entre les fonction qu’il connait « AV | RE |TG | TD | FCC » dans sa  « map » . La clef est le nom de fonction qui mène vers un la classe correspondante (ex : AvPrimitive)

L’interface graphique est composé d’une fenêtre : deux panneaux l’a compose. Le panneau interpréteur est le champ de saisie par laquelle nous rentrons les instructions logo. Le panneau dessin qui permet de dessiner dans le programme.

Une section « Aide » a été ajouté la barre de menu en haut du programme ainsi qu’une section « A propos » pour recadrer dans quel contexte se place ce logiciel.

* Livraison

Nous avons pris soin de livrer dans l’archive les éléments suivants :

* Les sources « packager » au mieux.
* Les librairies employées (JavaCC) ainsi que les échantillons.
* Un exécutable jar.

## Problèmes rencontrés

Nous avons rencontré plusieurs problèmes tout d’abord niveau organisationnel. Quand l’un de nous modifiais une partie de programme chez nous surtout la grammaire qui est difficile a comprendre.

Nous n’ajoutions pas d’explication à notre code donc une difficulté de compréhension et des ambiguïtés s’installais.

Nous avons finalement résolue cela à l’aide de skype pour communiquer.

Il y a eu aussi de nombreux problème technique car nous étions complètement perdus avec JavaCC qui est une nouvelle technologie pour nous. Et avec les grammaires qui sont un aspect que nous n’avions jamais appréhender auparavant. Mais avec l’aide de notre professeur nous avons pue commencer l’ébauche de notre projet ce qui nous a mis sur la voix.

Coté technique, il s’agissait de créer un parseur mais également un « désérializer » dans notre partie « interpreteur ». Nous n’avons pas été en mesure de composer les opérations REPETE car le patron de conception « interpreter » (concept de génie logiciel avancée) implémenté ne suffisait plus à la situation.

## Conclusion

Le cours et le projet de compilation de M. Dechelle nous ont permis de comprendre les enjeux de la compilation. Transformer une chaine en entrée fait appel à des concepts mathématiques tel que les langages formels. Derrière le java qui nous était familier, nous devions utiliser un analyseur syntaxique descendant qui nous imposait de respecter des règles stricts de code.

Nous tenons à mettre en valeur les projets de M. Dechelle qui permettaient de toucher à beaucoup de technologies différentes. Les projets étaient nombreux et très instructifs. Nous pensons avoir été introduits sur le concept de l’analyse syntaxicale.

Enfin nous espérons que vous avez apprécier Freddy Mercury !