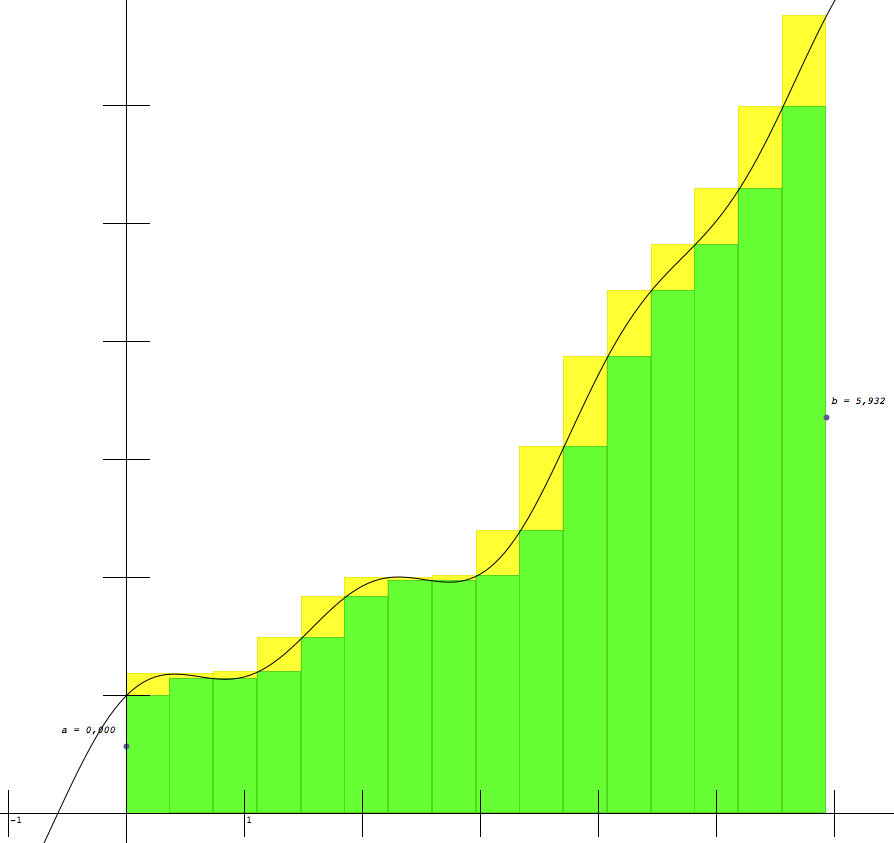
Travail de maturité n°013

Collège de Saussure

**« Comment créer un programme JAVA pour faire comprendre un concept mathématique ? »**



**Lomig Mégard**

Volée 2006-2008

*Professeur accompagnant :*

**Jean-Marie Delley 16 Novembre 2007**

Table des matières

Table des matières 1

Remerciements 2

Introduction 3

Le choix 3

Le but du travail 3

Déroulement de la création du programme 4

Problèmes rencontrés 8

Latex 8

Exemples de problèmes de programmation 8

Conclusion 9

Bibliographie 10

Annexes 11

Lexique 11

Log des versions de la partie graphique 11

Code source 12

Remerciements

Je voudrais remercier tout d’abord M. Delley pour son soutien et son suivi attentif. Les professeurs M. Roux, M. Rey ainsi que M. Perrin m’ont aussi aidé d’une façon ou d’une autre et je leur en suis très reconnaissant.

Merci à toute ma famille pour leurs avis extérieurs sur le programme et leur ténacité à m’inciter à travailler régulièrement.

Introduction

Le choix

Dès le départ, je me suis orienté vers une réalisation « artistique » sur ordinateur. Cependant, je ne savais pas sur quel langage partir, puisque je connaissais déjà l’HTML, les bases du C++ ainsi que de Java. Le sujet lui aussi me manquait. Pour cela, j’ai discuté avec plusieurs membres de ma famille et avec M. Philippe Rey, professeur de mathématiques et d’application des mathématiques au Collège de Saussure. Il m’a vite aiguillé sur l’idée du cours, au début sous forme de page HTML. À partir de cette idée, je me suis décidé à créer effectivement un cours mais en application Java car je voulais réellement approfondir ce langage. En effet, je trouvais bien plus intéressant d’apprendre un nouveau langage. Le sujet général du cours a relativement été vite choisi : il me fallait une branche que j’appréciais. Je suis donc parti sur les mathématiques, mais le sujet précis n’était toujours pas trouvé.

La simple idée de cours a vite été confrontée à un mur : pourquoi refaire un cours de mathématique alors qu’il en existe déjà à foisons ? Face à cette question implacable, j’ai décidé d’utiliser les capacités, ou spécificités, qu’ont en plus les ordinateurs par rapport aux cours « standards » que l’on peut par exemple trouver au Collège. Les ordinateurs ont cette faculté de pouvoir modifier l’apparence de tous les éléments d’une application à n’importe quel moment. Deux mots sont très vite apparus dans mon esprit : dynamisme et interactivité. Mon futur cours devait donc être le plus dynamique et interactif possible, cela pour apporter quelque chose de plus à l’élève qu’un polycopié.

J’ai alors dû trouver le sujet précis de mon programme, dans les mathématiques. Mon choix s’est rapidement porté sur l’intégration puis enfin sur les Sommes de Riemann. J’ai choisi ce chapitre parce qu’il contenait déjà une partie visuelle, celle-ci était dans mon esprit plus simple à montrer, illustrer, dans un programme qu’une page de calculs algébriques.

Le but du travail

Je suis parti avec l’hypothèse qu’il est possible d’améliorer les cours que l’on nous donne au Collège de Saussure. Ma problématique devenait alors : *« Comment créer un programme JAVA pour faire comprendre un concept mathématique ? »*. Je l’ai abordée avec deux objectifs.

Le premier est du côté de la programmation : la question est bien *comment créer un programme*, soit de quelle manière et avec quels outils peut-on arriver à coder un tel logiciel ? Je cherche donc à savoir quel niveau de connaissance en Java il faut avoir et où trouver cette connaissance. Mon but dans le cas de ce travail n’était pas de créer un programme professionnel, mais de démontrer qu’avec quelques sources et de la patience, on peut arriver à un résultat convenable, sans être parfait.

J’ai abordé le mot *comment* dans le sens pratique, le deuxième but se tourne plus sur la façon de créer le cours *pour faire comprendre un concept mathématique*. En partant d’un cours polycopié, mon dessein est de l’améliorer, de le rendre plus interactif, moins figé pour l’élève. J’ai donc dû, durant l’élaboration du programme, penser le plus possible dans ce sens, en cherchant diverses idées pour modifier le cours.

Mon programme a donc pour but de démontrer qu’il est possible d’utiliser l’informatique pour rendre plus abordable un cours. Pour cela, je me suis concentré sur un sujet précis, les Sommes de Riemann, mais j’ai développé tout le programme pour qu’il puisse accueillir d’autres chapitres.

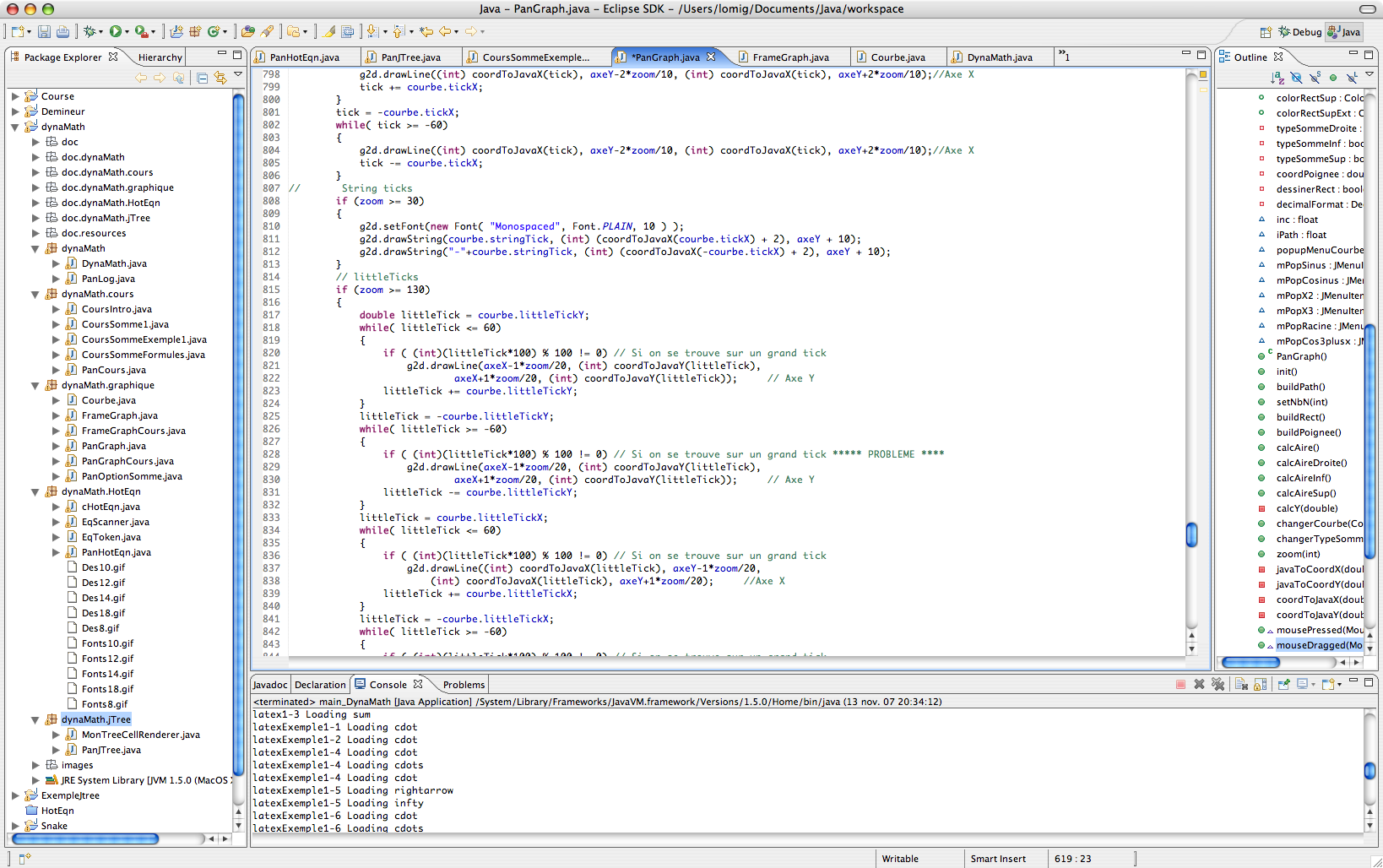
Déroulement de la création du programme

Mon premier réflexe a été de chercher sur Internet ce qu’il y avait déjà comme programme faisant environ la même chose que le mien. J’ai bien sûr trouvé le très bon logiciel GeoGebra[[1]](#footnote--1), lui aussi écrit en Java. Il a un niveau beaucoup plus élevé que celui que je souhaitais atteindre et il ne propose aucun cours à proprement parlé, mais seulement la possibilité de construire à la volée divers objets géométriques reliés entre eux. Cependant, il m’a donné une idée de ce qui peut être fait avec du temps.

J’ai aussi dès le début acheté un livre pour apprendre les bases du langage Java : « Comment programmer en Java » de Deitel et Deitel[[2]](#footnote-0). J’ai donc commencé à le lire, tout en appliquant une partie de ce que j’apprenais au cours d’application des mathématiques. Ce livre contient de nombreux exemples très bien commentés qui m’ont énormément appris et sans lesquels je ne serais pas allé bien loin. Par contre, pour certains points plus précis que je devais utiliser pour mon programme, mes seules ressources étaient des sites spécialisés, notamment pour l’interface utilisateur (GUI, *Graphical User Interface*), où uniquement les bases étaient expliquées dans le livre.

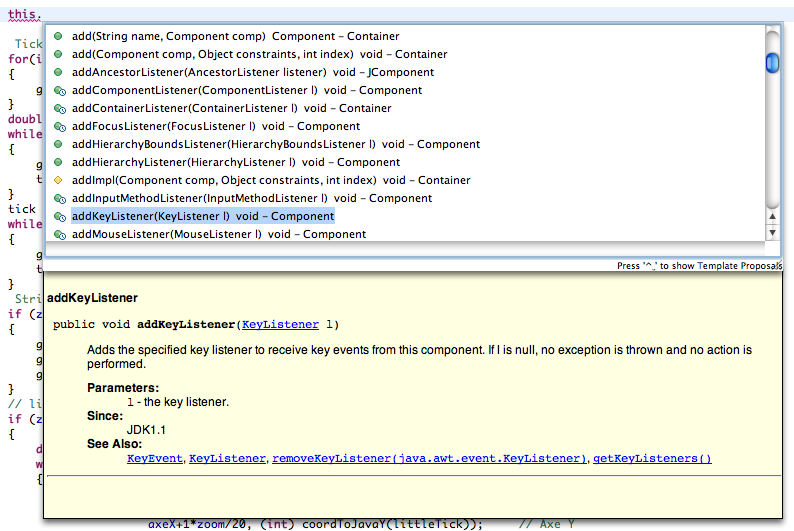
Avant de pouvoir commencer à programmer, j’ai dû choisir un IDE (en anglais pour *Integrated Development Environment*), ou EDI pour environnement de développement intégré. Ce type de programme offre un traitement de texte (avec de nombreuses aides comme l’indentation automatique, la coloration syntaxique, etc.), un compilateur qui donne des résultats d’erreur précis et le plus souvent un débogueur. J’ai adopté le logiciel Eclipse[[3]](#footnote-1) qui permet vraiment de gérer un projet tout en offrant tout d’un EDI normal. Une de ses fonctions qui m’a été très utile est qu’il donne une liste de possibilité lorsque l’on frappe (dans des cas précis), ce qui permet tout d’abord de connaître les possibilités qui existent et en plus de compléter plus rapidement le mot que l’on écrit. En fait, Eclipse intègre toute la JavaDoc des packages standards ce qui permet à tout moment d’avoir plus d’informations sur l’utilisation d’une classe comprise dans ces packages.

La fenêtre principale d’Eclipse, avec à gauche la liste des packages et des classes, au centre le traitement de texte, en bas la console et les problèmes et à droite la liste des variables et méthodes :



Eclipse : fenêtre principale

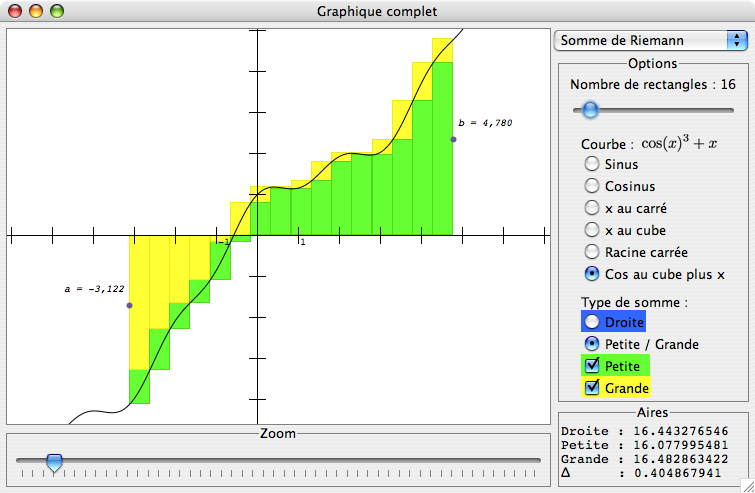
Un exemple de la complétion que fait Eclipse et des informations de la JavaDoc :



Eclipse : le programme donne les possibilités lors de la frappe

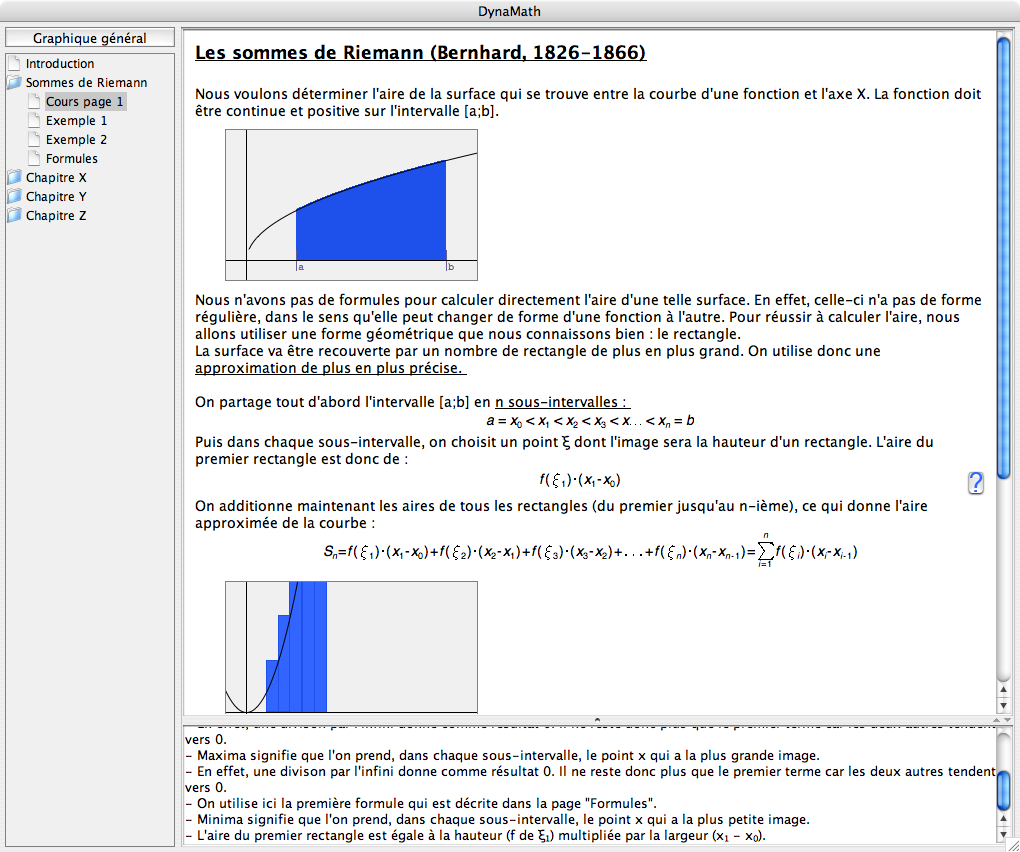
La première chose que j’ai codée est le graphique. Celui-ci a déjà demandé de longues heures de recherches rien que pour dessiner sa base : le repère, la courbe et les rectangles, le tout totalement immobile. J’ai ensuite rajouté petit à petit de nouvelles fonctionnalités (voir le log de version du graphique en annexe) jusqu’à arriver à un graphique qui permet de changer de courbe, de modifier l’emplacement des bornes, de montrer aussi bien la petite somme que la grande et de voir la valeur de chaque somme. Le graphique est lui aussi conçu pour qu’il puisse illustrer d’autres sujets que les Sommes de Riemann : la partie des options contient un menu défilant qui permettrait de choisir entre les sujets et dès que cela serait fait, le graphique changerait ainsi que les options qui lui sont associées.

Ci-dessous se trouve la capture de la fenêtre du graphique. On voit à droite le panneau qui contient toutes les options avec en haut le menu qui permettrait de changer de chapitre. L’utilisateur peut changer le nombre de rectangles, la courbe et le type de somme. J’ai aussi affiché les différentes aires obtenues avec en plus la différence entre la petite et la grande aire (delta). Les petits cercles qui se trouvent au milieu de la hauteur du premier et du dernier rectangle permettent de modifier respectivement les bornes d’intégration a et b.



DynaMath : fenêtre du graphique

Dès que la partie graphique a commencé à prendre forme, je me suis attelé à la conception de la fenêtre principale. Je suis parti avec quelques croquis sur papier pour l’apparence générale. Une partie à gauche est un « arbre » (*tree*, en anglais) dont le but est la navigation entre les différentes pages du cours. Là aussi, il permettrait d’afficher plusieurs sujets regroupés en dossiers. Au centre se trouve le cours en lui-même, c’est cette partie qui, tant au niveau de la programmation que du contenu, m’a posé le plus de problèmes. Ceux-ci sont développés dans le chapitre du même nom.



DynaMath : fenêtre principale

Pour chaque partie du programme, j’alternais en permanence entre des phases de programmation et des phases de recherches sur divers sites. En effet, je devais me renseigner sur chaque classe (ou presque) que j’utilisais pour cerner les possibilités qu’elle offrait. De plus, j’avais parfois une idée bien précise dans la tête, mais je ne savais pas comment la réaliser en java. Le plus souvent c’est grâce aux questions déjà posées sur des forums que j’ai pu avancer. Certaines idées ne se sont jamais réalisées.

Une fois l’armature du logiciel posée, j’ai enfin commencé à rédiger le cours en lui-même. Pour cela, je me suis basé sur le polycopié de M. Perrin, mon professeur de mathématiques en 3ème et 4ème année. J’ai relu de nombreuses fois les quelques pages qui traitent des Sommes de Riemann en les annotant. J’ai décidé que le texte de mon cours serait très ressemblant à celui du polycopié mais en permettant à l’élève d’avoir plus d’informations sur un mot ou sur la résolution d’une équation en intégrant au cours des petits boutons qui, lorsque l’on clique dessus, écrivent un texte explicatif dans une zone de la fenêtre principale prévue à cet effet. Les images du polycopié ont été complètement repensées avec l’ajout au cours de graphiques interactifs et non pas d’images statiques. J’ai repris mon code qui servait au graphique complet pour créer des petits graphiques affichants plus ou moins d’informations suivant les cas. Ces graphiques intégrés à même le cours permettent à l’élève de voir directement, en changeant lui-même les paramètres, ce qui est expliqué sous forme de texte dans le cours. J’ai aussi relié des calculs à des graphiques pour qu’ils changent suivant le nombre de rectangles affichés.

Problèmes rencontrés

Latex

Faire un cours sur un sujet de mathématique implique forcément l’apparition de formules. À l’aide d’un simple traitement de texte, il est impossible d’écrire des formules, ainsi qu’avec les packages standards Java. Plusieurs possibilités s’offraient à moi : Premièrement, il aurait été possible de créer les formules avec un logiciel tiers puis de les exporter dans un format d’image et de les intégrer au cours tel quel. Cependant, utiliser cette technique offrait très peu de souplesse, par exemple si je devais changer une formule, tout le travail de création-exportation-importation aurait été à refaire. De plus, je voulais que certaines formules soient liées à un graphique, et qu’elles changent en fonction du nombre de rectangles, ce qui impliquait de faire un grand nombre d’images différentes. Il me fallait donc créer les images directement dans le programme java, pour tout automatiser, ce qui amène l’autre possibilité.

J’ai découvert peu de temps avant le début du travail le langage Latex, qui permet justement à partir de lignes de code de créer une image, le plus souvent illustrant une formule mathématique. Par exemple, pour écrire la fraction 1 sur 2, la syntaxe est *\frac{1}{2}*. J’ai donc commencé à chercher une application java qui interprète le langage Latex. Je n’ai pas pensé à la faire moi-même puisqu’un tel interpréteur pourrait être à lui seul un sujet de Travail de Maturité, vu sa complexité. L’applet HotEqn créée par Christian Schmid, professeur à l’université de Ruhr, faisait presque ce que je voulais. Croyant qu’elle était sous copyright, j’ai contacté l’auteur par e-mail, et il s’avère en fait qu’elle est sous licence GNU, soit *« open source »*. J’ai alors modifié en quelques endroits le code pour qu’il s’adapte à mon programme.

Grâce à cet interpréteur, j’ai pu intégrer à mon cours des formules avec une bonne présentation et surtout modifiables à tout moment en donnant simplement une nouvelle ligne de Latex à la formule.

Le site de l’applet HotEqn est :

http://www.atp.rub.de/VCLab/software/HotEqn/HotEqn.html

Et l’on peut télécharger les sources à cette adresse :

<http://vclab.atp.rub.de/Src/GPL/HotEqn/HotEqnSrc.zip>

Exemples de problèmes de programmation

Mon premier exemple est la mise en page du cours qui m’a réellement donné du fil à retordre. En effet, j’ai dû trouver un moyen d’écrire le cours, tout en lui intégrant des graphiques, des boutons et des images. J’ai utilisé la classe JTextPane, qui m’a semblé adaptée à mes besoins. Cependant, j’ai très vite eu quelques problèmes : le retour à la ligne automatique ne marchait pas et changer le style du texte (gras, italique, taille, etc.) était particulièrement compliqué. Pour le retour à la ligne (*wrapline*, en anglais), j’ai passé de nombreuses heures à chercher sur toute la toile, aussi bien en français qu’en anglais. Partout les programmeurs cherchaient à enlever ce retour à la ligne (qui pour eux était bien automatique) ! J’ai finalement trouvé le problème qui venait de l’ordre dans lequel « j’encapsulais » chaque composant dans l’autre. En effet, de l’intérieur à l’extérieur j’avais la zone de texte (JTextPane), puis la « pile de cours » (comme un jeu de cartes, où chacune est une page du cours) et enfin la « boite de l’ascenseur » (JSrollPane). C’est celle-ci qui provoque le retour à la ligne mais dans mon cas elle n’englobait pas directement la zone de texte et donc le retour ne marchait pas. J’ai donc tout simplement changé l’ordre en JTextPane, JScrollPane et « pile de cours » pour que tout marche convenablement. Ce problème était au-delà de la syntaxe et impliquait de bien comprendre comment chaque partie du programme interagit avec une autre pour pouvoir le résoudre.

Le contenu du cours était au début tiré uniquement d’un fichier texte qui contenait une syntaxe bien spéciale que j’ai inventée. Le programme java se contentait d’interpréter le fichier pour remplir chaque page du cours. Le premier mot de chaque ligne annonçait au programme ce qui allait suivre : du texte, du latex, un graphique ou encore une image. La suite de la ligne contenait les paramètres. Ce système avait l’avantage de pouvoir modifier chaque page du cours avec un traitement de texte. Tout marchait parfaitement, mais dès que j’ai dû lier divers composants à d’autres, des problèmes se sont posés. Il devenait bien trop compliqué de continuer avec le système de fichier texte, j’ai donc tout repassé en code source java, où les liens entre les composants se font facilement. J’ai fait ce changement car le but de mon Travail de Maturité n’est pas de programmer un interpréteur ni d’inventer un nouveau langage. Cependant cette étape m’a permis d’apprendre à lire dans un fichier texte, soit de savoir gérer les flux d’entrée et de sortie. Bien que je ne les utilise pas dans mon programme, les flux sont importants à connaître et je pourrais mettre en pratique cette connaissance dans le futur.

Conclusion

Mon but premier était d’obtenir une autre approche d’un cours polycopié en le rendant plus dynamique et interactif. Je pense maintenant que cet objectif est en grande partie atteint : Mon programme montre en effet qu’il est possible d’obtenir un tel cours. J’ai essayé d’appliquer un maximum de mes idées au programme et je pense que l’on voit bien les changements opérés. Cependant, par manque de temps et de connaissances, certaines idées n’ont pas été concrétisées. Par exemple j’aurais voulu rendre le cours modifiable par l’utilisateur : possibilité de rajouter du texte, de surligner (comme avec un stabilo) ou encore de changer les paramètres d’un graphique. Mais tout cela implique que le cours soit sauvegardé dans un fichier et lu à chaque démarrage, ce qui aurait demandé un effort démesuré par rapport à celui que l’on attend dans le cadre du Travail de Maturité. Je pense donc que mon programme montre le début de ce qui pourrait être fait.

Après la fin du travail, j’ai pu observer que ma maîtrise du langage Java s’est considérablement améliorée. En effet, je suis parti avec seulement quelques bases et me voilà capable de créer un programme complet d’un niveau moyen. J’ai de plus découvert l’EDI Eclipse qui me serra utile pour réaliser mes futurs logiciels. Ce projet m’a aussi permis de réaliser un travail sur une longue période avec ce que ça implique : choix du sujet, planification du déroulement, recherche de documents de référence, etc. Je suis finalement content d’avoir pu concrétiser mon envie de faire un programme complet.

Je termine ce complément de travail avec une idée : serait-il profitable de donner aux élèves un support informatique à leur cours ? La réalisation d’un sondage, voire de tests réalisés en classe à l’aide d’un logiciel dans la lignée du mien mais plus complet, pourrait donner des réponses à cette question et aider les professeurs à rendre leur cours plus « proche » d’une nouvelle génération d’élèves, souvent déjà plongés dans le monde de l’informatique. Je vais montrer visuellement la plus-value présente dans mon programme lors de la soutenance orale, en le comparant avec le cours polycopié de M. Perrin. J’espère donc faire ressortir des idées qui je pense peuvent réellement aider les élèves.

Bibliographie

• Livre de référence sur le langage Java :

H. M. Deitel et P. J. Deitel, *Comment programmer en java*, Goulet, 2002, 1546pp.

• Sites de référence :

http://java.sun.com/javase/reference/index.jsp

http://java.developpez.com/

http://www.javafr.com/

Annexes

Lexique

Ce lexique est uniquement fait pour les personnes ne comprenant pas certains termes propres à la programmation ou à Java en particulier. Les définitions sont donc très simplifiées et leur unique but est de faciliter la compréhension du dossier.

**Classe :** Une classe est l’élément de base du type de langage dit « orienté objet », comme le Java. Il y en a une principale par fichier. Bref pour faire simple, une classe est un *fichier* qui gère une partie du programme : j’ai par exemple une classe *PanGraph* qui s’occupe du graphique dans mon programme.

**JavaDoc :** La JavaDoc est créée automatiquement sous forme de pages HTML et contient les commentaires insérés dans le code (entre autres), cela de manière organisée. Elle aide à avoir un aperçu général du programme (au niveau du fonctionnement).

**Package :** Les packages sont des sortes de « sections ». Par exemple il y a le package standard *text* qui contient toutes les classes qui servent à manipuler du texte. Il est lui-même contenu dans le package *java*.

Log des versions de la partie graphique

Le log des versions qui suit a été écrit au fur et à mesure que les fonctionnalités se rajoutaient au graphique. Je le mets ici pour montrer un exemple de l’avancée d’une partie du programme.

**V0.0 :**

- Dessin d'une courbe sinus avec les GeneralPath.

- Ajout d'un repère.

- Ajout du zoom modifiable.

- Ajout de méthodes qui convertissent les coordonnées de mon repère en celles qui sont pour java (0;0 en haut à gauche)

- Ajout de la graduation (tick).

**V1.0 :**

- Dessin des rectangles, pour l'instant ne marche que si la courbe est positive entre [a;b].

**V1.1 :**

- Les rectangles vont aussi où la courbe est négative.

**V1.2 :**

- L'aire des rectangles est calculée.

- Les coordonnées sont maintenant des doubles, pour avoir plus de précision.

- Rajout de l'anti-aliasing sur la courbe pour enlever l'effet "pixellisé".

- On peut maintenant choisir entre 4 courbes en ouvrant le menu contextuel du graphique.

**V1.3 :**

- On peut changer l'emplacement des repères juste en cliquant là où l'on veut.

- Les premières poignées font leur apparition. Celles-ci permettent de changer les bornes a et b juste en les glissant.

- Les valeurs des bornes s'affichent à proximité des poignées.

**V1.4 :**

- Rajout de la courbe racine carrée.

- Gestion de la molette de la souris, elle contrôle le zoom.

- On peut bouger le repère avec un drag de la souris.

- Création de la classe Courbe. Cela simplifie la gestion des variables qui sont en rapport avec la courbe dessinée. Comme par exemple la borne a qui ne doit pas être plus petite que 0 pour la racine carrée.

- Changements mineurs.

- Le clic gauche met maintenant le lieu du clic au centre du panel.

**V1.5 :**

- Changement des bordures des composants de la Frame

- Le zoom reste maintenant bien centré sur le milieu de la fenêtre et ne se rapproche plus de l'origine du repère. Il reste une petite imprécision (je ne sais pas encore la cause).

- Il est maintenant possible de modifier la taille de la fenêtre.

- Le nom de la courbe dessinée s'affiche en bas à droite de la fenêtre.

- Dessin des "petits ticks", ils découpent le "grand tick" en dix parties.

- Possibilité de dessiner les rectangles de la somme inférieure et supérieure en plus de la somme droite (le changement n'est pas encore possible graphiquement).

**V1.6 :**

- Rajout d'un nouveau panel qui contient toutes les options pour le graphique. On peut choisir en premier le type de cours que l'on veut illustrer (que Somme de Riemann de disponible pour le TM), puis les options qui sont en rapport s'affichent.

- On peut maintenant afficher la somme droite, supérieure et inférieure. Les deux dernières peuvent êtres dessinés en même temps.

- Petits changements d'interface.

- La JavaDoc est complétée.

**V1.7 :**

- Les aires sont maintenant affichées en dessous du panel d'options.

- Les signes grecs sont affichés grâce à l'Unicode.

Code source

Les pages qui suivent contiennent l’intégralité des classes que j’ai écrites et qui composent mon programme. Je n’ai pas imprimé toute la partie de l’interpréteur Latex car ce n’est pas mon code et elle fait plus de 3000 lignes !

La mise en page vient du programme Smultron[[4]](#footnote-2), très bon éditeur de texte pour le code. Je n’ai pas imprimé depuis Eclipse car la police était bien trop grande et je n’ai pas trouvé comment modifier les options d’impression.

1. Site officiel : http://www.geogebra.org/ [↑](#footnote-ref--1)
2. H. M. Deitel et P. J. Deitel, *Comment programmer en java*, Goulet, 2002. [↑](#footnote-ref-0)
3. Site officiel : http://www.eclipse.org/ [↑](#footnote-ref-1)
4. Site officiel : http://smultron.sourceforge.net/ [↑](#footnote-ref-2)