26 avril 2023

RAPPORT DU PROJET

ANALYSEUR DE LIVRES DONT VOUS ÊTES LE HÉROS

Alexis Pestel
Olivier Burnel
Quentin Dumont

L2 INFORMATIQUE
UE Projet 1 TD2B

UNIVERSITÉ
CAEN
NORMANDIE

http://www.unicaen.fr

Sommaire

1 INTRODUCTION

Les sujets proposés pour le rendu de l'unité d'enseignement "Projet 1" étaient d'une grande diversité et nous semblaient tous plutôt complexes à mettre en place. Nous avons choisi de nous atteler au développement de l'analyseur de "livres dont vous êtes le héros", parce que nous étions inspiré par l'intitulé du sujet. Ce dernier paraissait plus concret : nous devions à partir d'un livre au format `.txt` ou `.json` être capable de découper ses sections et de savoir où ces-dernières pourraient nous mener. En toute honnêteté nous n'imaginions absolument pas que ce projet nous poserait autant de difficultés, nous aurons d'ailleurs le plaisir de vous en faire part au cours de ce rapport. Nous allons diviser notre propos en trois parties, pour reprendre le schéma d'un pattern de conception de logiciel que nous commençons à bien connaître en fin de deuxième année de licence [1]. Nous allons d'abord expliquer comment fonctionne notre modèle, en nous limitant à la partie de l'analyse des différents paragraphes. Nous réservons la grande partie de l'affichage du graphe du livre à la "vue" du logiciel, bien que cet affichage repose sur un algorithme faisant partie du modèle de l'application. Notre ultime partie décrira la forme finale de notre logiciel et les perspectives d'améliorations à développer, afin qu'il se rapproche le plus de ce que l'on voulait en faire il y a maintenant quatre mois. Nous vous souhaitons une bonne lecture.

2 ANALYSE D'UN LIVRE

Le découpage du livre est une partie du projet qui ne nous a pas posé de problèmes. Nous avons su segmenter les blocs de code à développer pour charger un fichier, découper ses paragraphes et enfin les analyser, c'est à dire en extraire des informations importantes comme leurs parents (les paragraphes qui mènent au paragraphe courant) ou leurs enfants (les paragraphes vers lesquels peut nous mener le paragraphe courant).

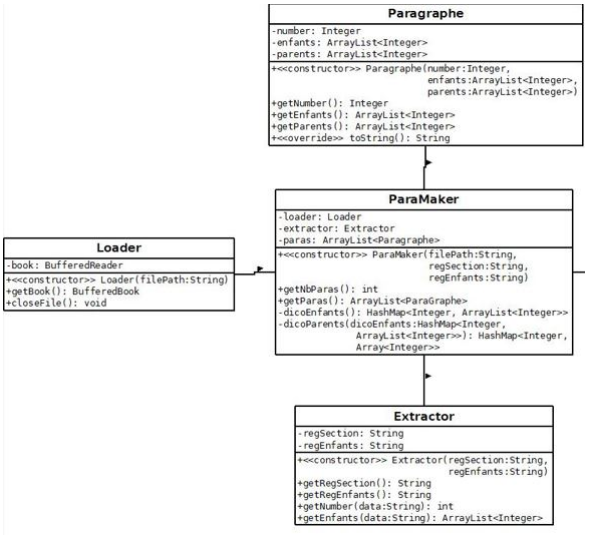

FIGURE 1 – Diagramme de classe, côté modèle

1. Modèle, Vue, Contrôleur

2.1 Chargement d'un fichier (A.P.)

Notre projet commence, que devons-nous faire ? Tâchons déjà de lire le livre, ou du moins le faire lire par un algorithme écrit par nos soins. Afin de nous aider dans notre démarche, nous avons écrit une classe '`Loader`' nous permettant de charger un fichier à partir d'un chemin fourni préalablement. Une fois le fichier correctement trouvé et chargé, nous pouvons ultérieurement l'utiliser dans d'autres classes, expliquées plus tard dans ce rapport.

2.2 Expressions régulières (O.C.)

Afin d'analyser que ce soit des fichiers '`.json`' ou '`.txt`' nous devons utiliser des expressions régulières. Notre programme ne prend donc en compte que ces deux extensions de fichiers, suivant l'extensions les expressions régulières y sont adaptées. Le constructeur de la classe '`Paramaker`' prend donc trois arguments dont deux `regexs`, un '`regex de section`' et un '`regex d'enfants`'. Ces regexs nous permettent donc de remplir une 'HashMap' qui collectionne pour chaque paragraphes les paragraphes enfants correspondants. C'est tout ce que nous détectons dans la version finale mais dans les versions nous arrivons à y détecter de nombreux types différents ('Victoire', 'Défaite', 'Spécial', 'Loot', 'Combat', 'Talent' et pour finir 'Aléatoire'. Chaque classe '`Paragraphe`' peut posséder plusieurs types. Il y a donc une hiérarchie dans les types, et suivant le type le plus important du tableau une couleur y était définit.

2.3 Structures de données (Q.D.)

Pour que notre extraction de données soit efficace, il a fallu utiliser des structures de données adaptées à nos besoins. Notre objectif était de pouvoir accéder au plus de valeurs possibles de façon constante, à l'aide d'une clé ou d'un index. Nous voulions à tout prix éviter le parcours de grandes listes ou la recherche dans des tableaux, parce que la complexité de ce type d'opération est linéaire, c'est-à-dire qu'elle augmente de façon proportionnelle par rapport à la taille des données que nous devons traiter. Pour pouvoir tracer un graphe des possibilités du livre, nous nous étions mis en tête de sauvegarder les paragraphes parents et les paragraphes enfants de chaque paragraphe. Il nous fallait donc un objet `Paragraphe` que l'on puisse instancier à chaque nouvelle section découpée. Nous stockons ces objets `Paragraphe` dans un dictionnaire (une `HashMap` en java), en les associant à une clé unique qui n'est autre que leur numéro (donc un entier). Ainsi, pour qu'un paragraphe connaisse ses enfants et ses parents, il n'a besoin de conserver qu'une liste d'entiers qui sont en fait des clés à utiliser dans la `HashMap` d'objets `Paragraphe`. C'est en manipulant ces structures que notre objet `Paramaker` parvient à créer autant d'objets `Paragraphe` que l'extracteur a trouvé de sections, et à lui renseigner les liens de parentés qui le relient à d'autres sections.

3 AFFICHAGE VIA KAMADA-KAWAI (Q.D.)

Une fois que nous avions ce dictionnaire de `Paragraphe`, nous pensions que nous avions fait le plus difficile. Que nenni ! Le déroulé du projet a pris une toute autre tournure lorsque nous sommes arrivés devant ce problème finalement très complexe : comment sommes-nous censés tracer un graphe qui soit visible et esthétique dans la mesure du possible ? Après avoir essayé de développer une solution qui plaçait les sommets au fur et à mesure sur un plan, nous nous sommes rapidement rendus compte que le résultat d'un algorithme aussi procédurier ne donnerait qu'une masse informe de sommets reliés dans tous les sens par des arêtes partant dans toutes les directions, ce qui serait tout bonnement incompréhensible. Alors nous avons relu l'intitulé de notre sujet, et une piste nous était donnée : une meilleure disposition du graphe

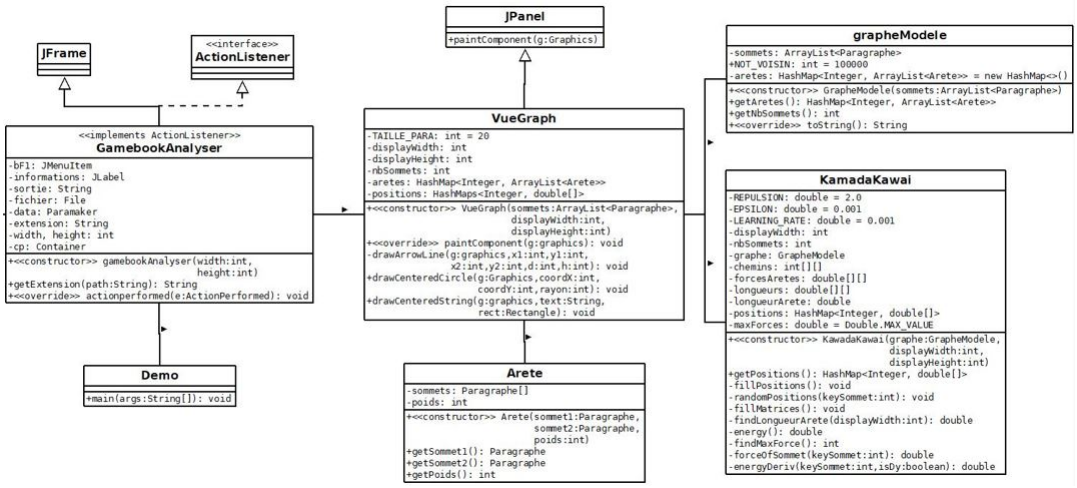

FIGURE 2 – Diagramme de classe, côté vue

pouvait s'obtenir en plaçant les sommets avec un algorithme de forces ; c'est à ce moment là que nous sommes partis pour implémenter l'algorithme de Kamada-Kawai.

3.1 Matrice des plus courts chemins

Qu'est-ce qu'un algorithme de forces ? En se renseignant sur le domaine visiblement très fourni et complexe de la théorie des graphes, nous avons pu nous faire une idée. Un algorithme de forces tel que Kamada-Kawai déplace des sommets initialisés à des positions aléatoires, de sorte à minimiser l'énergie physique entre tous les sommets. Cette énergie est calculée à partir de différents facteurs, notamment la distance optimale entre un sommet et ses voisins. Cette distance optimale et théorique peut être retrouvée à partir d'une matrice répertoriant, pour chaque sommet, le nombre minimal de sommets à parcourir pour atteindre chaque autre sommet. Ce problème est bien connu des scientifiques et programmeurs travaillant sur des graphes, et porte le nom de "All Pair Shortest Path Matrix", c'est-à-dire la matrice des plus courts chemins entre paires de sommets. Le calcul de cette matrice peut se faire de différentes façons, avec des algorithmes différents. Nous retiendrons les deux principaux, celui de Floyd-Warshall et celui de Johnson. Ces deux algorithmes ont une complexité en temps assez différente.

- Floyd-Warshall : $O(n^3)$
- Johnson : $O(n^2 * log(n) + E * n)$, avec E égal au nombre d'arêtes du graphe (*edges*)

Nous avons fait le choix de nous inspirer de l'algorithme de Johnson, pour des raisons qui vont sembler évidentes en comparant l'évolution en fonction de n des deux courbes des fonctions précédentes. Ci-dessous en rouge, la complexité de l'algorithme de Floyd-Warshall, en bleu, celle de l'algorithme de Johnson :

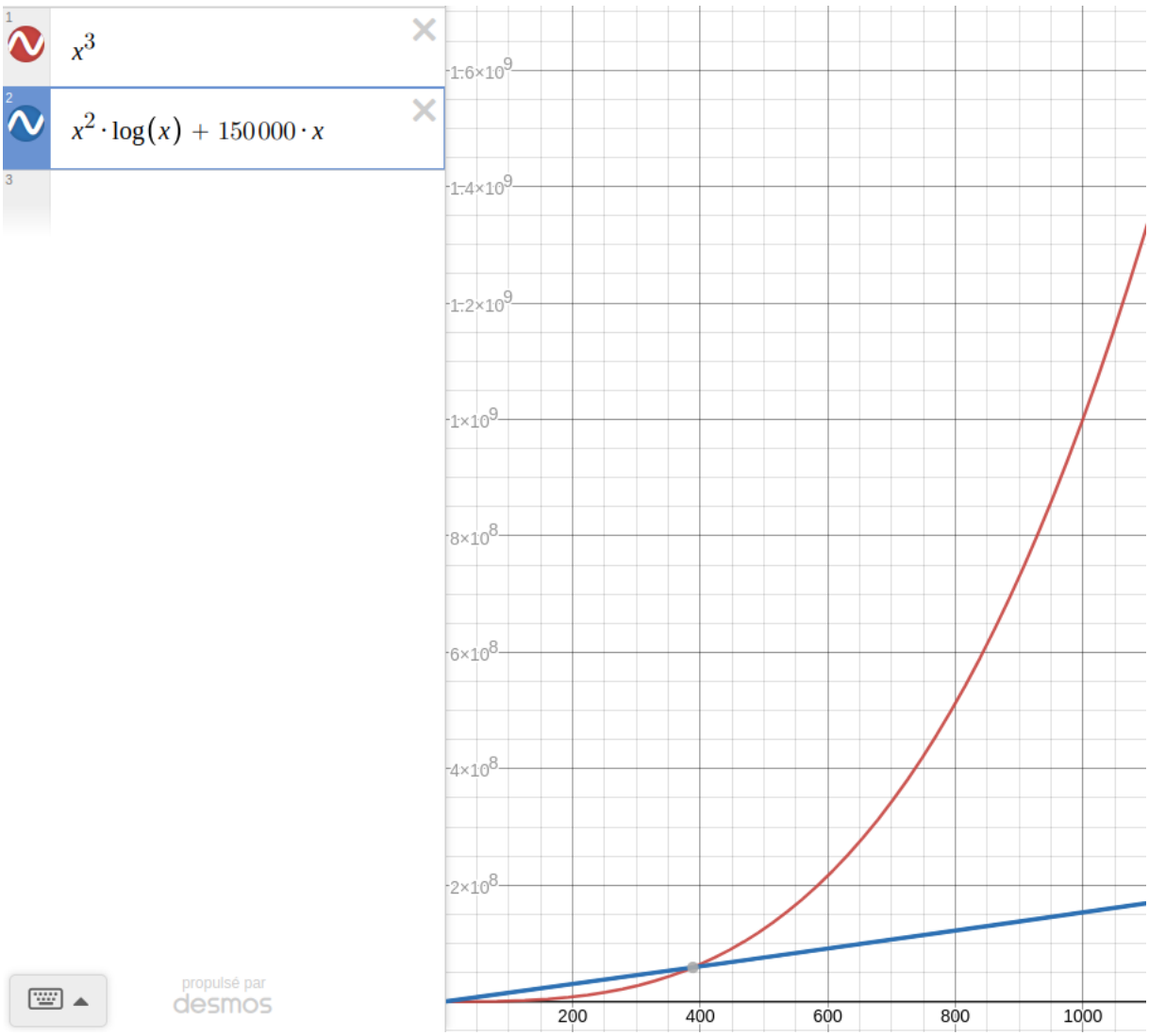

FIGURE 3 – Johnson (bleu) VS Floyd-Warshall (rouge)

On remarque très vite que même avec un nombre d'arêtes volontairement colossal, l'algorithme de Johnson garde une complexité en temps très raisonnable par rapport à celle de l'algorithme de Floyd-Warshall qui prend des proportions démesurées passé les 600 sommets. Le pire, c'est que ce calcul de matrice n'est pas la chose la plus coûteuse dans l'algorithme de Kamada-Kawai, alors autant aller au plus efficace dès qu'on le peut. Nous avons donc repris le fonctionnement de l'algorithme de Johnson, en le simplifiant davantage et en partant du principe que notre graphe ne pourrait pas être pondéré négativement. En fait, utiliser l'algorithme de Johnson dans notre situation revenait simplement à utiliser l'algorithme de Dijkstra. Le gain d'efficacité entre les deux algorithmes s'explique par le fait qu'à chaque tour de boucle, l'algorithme de Dijkstra i.e. celui de Johnson simplifié ne parcourt que les arêtes du sommet le plus proche, alors que celui de Floyd-Warshall parcourt tous les sommets à chaque fois. L'algorithme de Floyd-Warshall est donc plus facile à implémenter, mais aussi beaucoup moins efficace.

3.2 Résolution de systèmes

Une fois cette matrice des plus courts chemins entre paires de sommets calculée, nous pouvons calculer la distance réelle optimale entre chaque paire de sommet. Pour ce faire nous utilisons la valeur du plus court chemin théorique, et une unité de longueur en pixels calculée à partir de la longueur d'un côté de la fenêtre. A partir de là, si nous suivons le plan de l'algorithme de Kamada-Kawai, nous devons résoudre un système plutôt effrayant ; enfin, personnellement, il me fait peur :

$$\frac{\partial^2 E}{\partial x_m^2}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial x_m}(x_m^{(t)}, y_m^{(t)});$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial y_m^2}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial y_m}(x_m^{(t)}, y_m^{(t)})$$

$$x_m := x_m + \delta x;$$
$$y_m := y_m + \delta y;$$

FIGURE 4 – Système à résoudre pour minimiser l'énergie physique du graphe en un sommet

Où le grand E est l'énergie, caractérisée par la formule ci-dessous :

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{i,j} \left((x_i - x_j)^2 + (y_i - y_j)^2 + l_{i,j}^2 - 2l_{i,j}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right).$$

FIGURE 5 – Formule de l'énergie de Kamada-Kawai

Il s'agit là de résoudre un système d'équations non-linéaires, impliquant des dérivées partielles et des dérivées partielles secondes de la fonction d'énergie en fonction des coordonnées x et y des sommets. Le tout en utilisant la distance réelle optimale et la force de répulsion du sommet et de ses voisins. Très honnêtement je ne disposais pas d'un bagage mathématique suffisant pour calculer seul des dérivées partielles. Je me suis donc aidé de l'application web symbolab (https://fr.symbolab.com/) pour calculer ces dérivées. J'ai ensuite développer une classe `Gauss` qui me permettait de résoudre le système d'équations en utilisant la méthode du pivot de Gauss. Le résultat de ce cheminement quelque peu hasardeux ne fut pas au rendez-vous, probablement à cause de mauvais calculs de dérivées partielles. En effet, il était question de dériver une double sommation (donc une double boucle) dans laquelle le compteur de la première boucle servait dans le calcul de l'énergie sur le sommet courant, mais ce genre de détail ne pouvait pas être renseigné sur symbolab.

3.3 Descente de gradients

Malgré cette première tentative ayant demandé beaucoup de temps et de recherche, qui ne produit qu'un échec, j'ai voulu essayer autre chose. Ce système d'équations servait à minimiser la fonction d'énergie. Hors pour minimiser une fonction, on peut utiliser des outils mathématiques puissants : les gradients. Pour peu que la fonction soit suffisamment convexe, c'est à dire qu'elle n'ondule pas trop, on peut grâce aux gradients atteindre un minimum de la fonction. En fait, les gradients permettent de déterminer un coefficient de pente de la fonction. On peut donc savoir si la courbe de la fonction "monte" ou "descend". Cela nous permet de savoir si l'on doit augmenter ou diminuer x et/ou y pour s'approcher d'un coefficient de pente proche de zéro. Je suis donc parti du principe que nous n'étions pas obligés de résoudre un système pour déterminer la valeur exacte de x et de y à ajouter aux coordonnées du sommet, mais qu'on pouvait additionner aux coordonnées les gradients de la fonction d'énergie. Cependant les gradients sont en général de trop grandes valeurs.

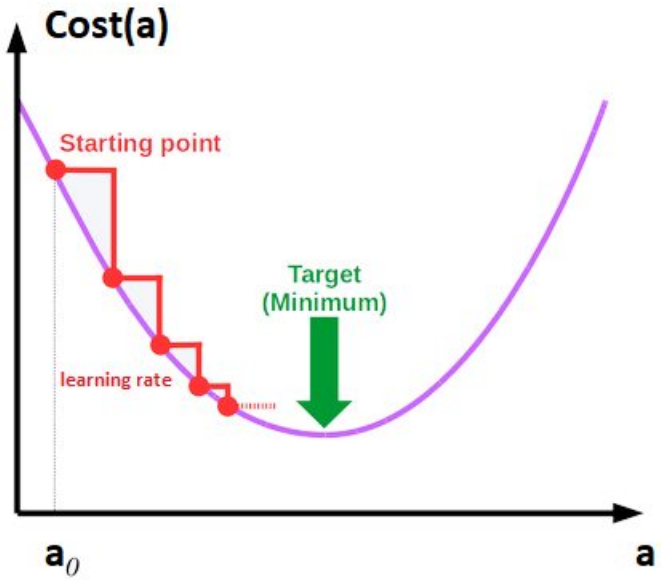

FIGURE 6 – Principe d'une descente de gradients

Cela nous empêche de se rapprocher d'un coefficient de pente nul, car on ne fait que sauter de part et d'autre de la courbe. Il faut donc choisir un taux d'apprentissage (ce dernier terme est utilisé en machine learning) pour se rapprocher suffisamment du minimum. J'ai pu essayer cette méthode avec des touts petits graphes de 5 sommets sur de petites surfaces, et l'algorithme produisait un résultat très lisible. Cependant lorsque j'ai tenté de l'appliquer à notre graphe de plus de 300 sommets, j'ai eu le même désordre qu'avec la version précédente de l'algorithme.

3.4 Recours à JUNG

N'ayant pas réussi à venir à bout de cet algorithme et voulant quand même obtenir un résultat visible de notre travail, j'ai décidé d'utiliser une bibliothèque implémentant des algorithmes de visualisation de graphes, y compris un algorithme de Kamada-Kawai. La librairie JUNG nous a permis de produire le résultat suivant :

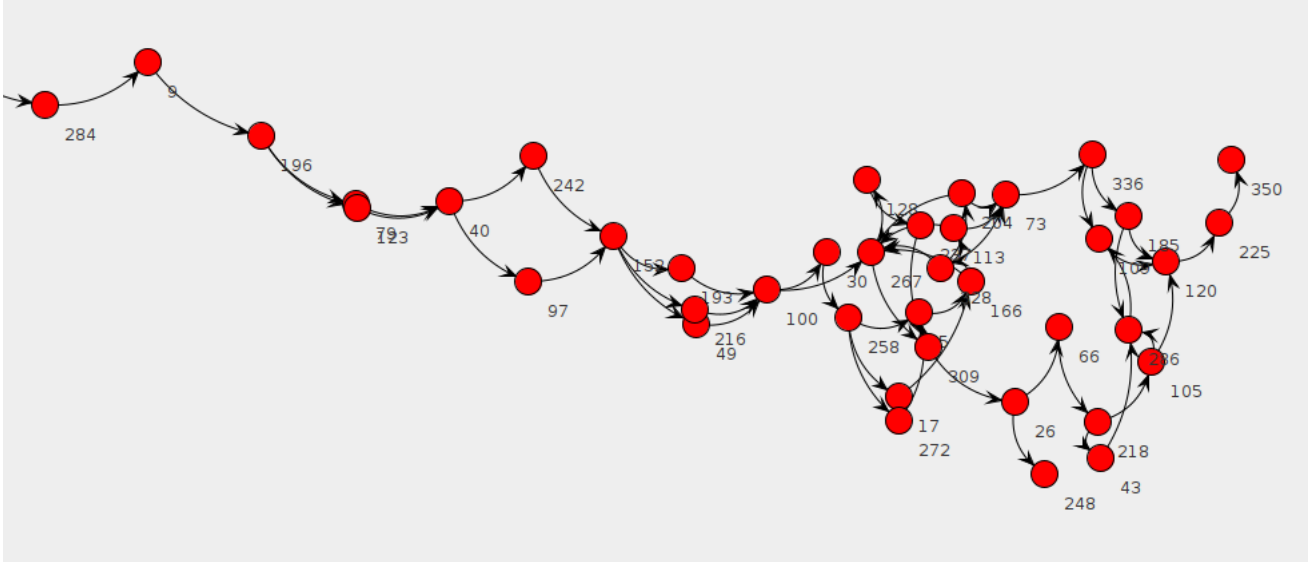

FIGURE 7 – Fin du graphe du livre "LoneWolf : Fire on the Water"

Ce résultat est meilleur que ce que nous avons pu produire avec des graphes de cette envergure, mais on constate qu'il y a encore beaucoup d'enchevêtrements de sommets et de croisements d'arêtes. Les méthodes et les objets fournis par JUNG sont paramétrables à souhait et je pense que nous pourrions trouver des réglages qui soient optimaux pour afficher nos graphes correctement. Toutefois cela prouve qu'il ne peut y avoir de solution miracle à un problème aussi complexe que celui de l'affichage automatisé de graphes.

4 CONTRÔLE DE L'INTERFACE (A.P.)

4.1 Composition

Afin de faciliter l'utilisation de notre travail, nous avons divisé notre affichage en trois parties distinctes, telles que :

- **Une barre de navigation JMenuBar** : Située en haut de la fenêtre, cette barre de navigation permet à l'utilisateur de sélectionner un fichier directement à partir d'une boîte de dialogue, rendant ainsi très fluide l'implémentation de nos algorithmes ;
- **Une zone d'affichage de graphe VueGraph** : Implémentant la classe '`JPanel`', cette zone d'affichage permet au graphe de correctement s'afficher avec la taille dont il a besoin. On utilise ici un '`ScrollPanel`' afin d'entièrement parcourir la zone du graphe si celui-ci dépasse la taille de la fenêtre ;
- **Un champ textuel JLabel** : Petite zone de texte nous permettant d'afficher certaines informations à propos du graphe, telles que l'extension du fichier importé, le nombre total de paragraphe et la transmission d'erreurs si le fichier n'est pas correctement pris en compte par l'affichage.

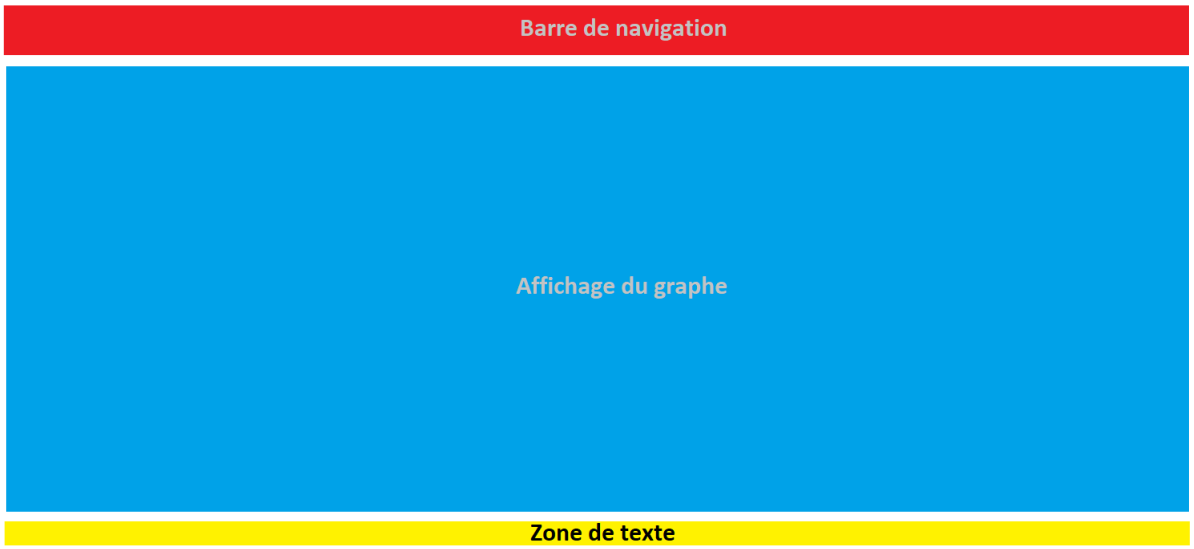

FIGURE 8 – Schéma de la composition de l'affichage

4.2 Utilisation

Nous avons tout fait pour que l'utilisation de notre vue soit la plus facile et ergonomique possible. Notre but ici est que l'utilisateur retrouve rapidement ses habitudes, comme par exemple avec la barre de navigation et son fameux menu déroulant '`Fichier`'.

4.2.1 Idéal

Une fois que l'utilisateur aura cliqué sur le bouton '`Ouvrir...`', celui-ci rencontrera une boîte de dialogue lui permettant de sélectionner le fichier de son choix.

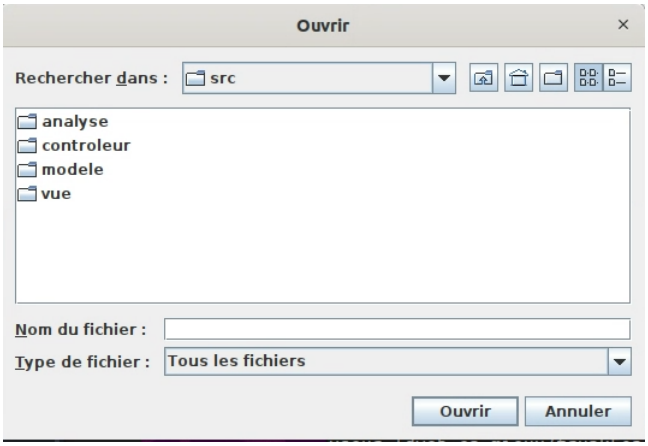

FIGURE 9 – Exemple de la boîte de navigation

Une fois que l'utilisateur aura sélectionné son fichier, ce dernier sera testé avant d'être soumis à l'algorithme permettant d'afficher le graphe : si celui-ci n'a pas d'extension en `.json` ou `.txt`, un message s'affichera dans la zone de texte en bas de la fenêtre lui indiquant que le fichier n'est pas pris en charge. Si le fichier a la bonne extension, alors le graphe commencera à s'afficher dans la zone prévue à cet effet. Dans le code actuel, il n'est pour l'instant pas possible de procéder ainsi à cause d'une présumée erreur de threads. Nous avons donc procédé autrement.

4.2.2 Réalité

Pour le moment, l'utilisation de notre algorithme de force se fait directement via le fichier '`GamebookAnalyser.java`', en changeant la valeur de la variable `FILE_PATH`. Il suffit d'aller soi-même chercher le chemin du fichier que l'on souhaite analyser et de modifier la valeur de cette dernière, par exemple avec l'aide de la commande Linux `realpath`. Ce n'est pas forcément la meilleure configuration (*contrairement à celle montrée ci-dessus*), mais c'est tout de même fonctionnel. Nous verrons dans le futur comment régler le problème que nous avons rencontré, déjà en re-vérifiant son origine, afin d'améliorer notre interface.

4.3 Ambitions

Beaucoup de choses sont encore à faire afin d'améliorer cette interface. Plusieurs exemples nous viennent à l'esprit, en voici quelques-uns :

- L'ajout de différentes informations, telles que le nombre de paragraphes ayant pour issue une victoire, un combat, un talent... ;
- La coloration de ces mêmes informations directement sur le graphe, avec des points de couleur différente (*impliquant donc l'ajout d'une légende*) ;
- La possibilité d'exporter son graphe sous forme de fichier `.xml`, afin d'avoir la possibilité de l'utiliser à n'importe quel moment, à n'importe quel endroit.

5 CONCLUSION

Pour conclure c'était un projet très intéressant, nous y avons vu de nombreux concept et avons pu bien expérimenter le côté interface. Au debut du projet, nous n'imaginions pas qu'il était aussi complex d'obtenir un rendu graphique propre à partir de données... C'était donc un challenge qui en valait la peine. Ce projet nous a demandé une grande implication et une bonne entente dans le groupe que ne pas se marcher dessus en essayant un maximum de partager les tâches. nous avons trouvé que la partie la plus difficile était de très loin la partie interface en Swing. Il est assez drole de voir la differences entre nos premiers graphes et nos plus récents que ce soit en esthétique ou en efficacité. Nous sommes donc arrivé à court de temps mais nous avions pourtant l'ambitions de rajouter encore plus de fonctionnalités comme la détection de tous les types de paragraphes comme nous en avons parlé precedement. Nous avions aussi prevu d'ameliorer nos analyse de fichiers en détectant les jeux à victoires-multiples. Et pout finir le dernier ajout auquel on avait pensé aurait été de trouver tout un tas de statistiques comme la défaite/victoire la plus rapide ainsi que la plus lente, le chemin qui possède le plus de combats ou encore celui qui en possède le moins.