Prototype et Interface - M1 S2
TP2 - Les Petits Débrouillards

Titouan Parcollet
Mathias Quillot

Mars 2021

Contents

1 Introduction

Ce sujet a pour but de vous sensibiliser à la création d'un prototype d'interface dans un contexte réel. Vous auvez une interface à réaliser, en ayant connaissance de l'utilisateur final et de besoins exprimés. Il s'agira dans un premier temps de comprendre le besoin et de trouver des solutions pour proposer une interface fonctionnelle et ergonomique.

Vous aurez à intégrer à votre interface un élément graphique externe à votre IDE : GraphStream. La version 1-3, compatible avec java8, vous est fournie, mais vous êtes libre de télécharger une version plus récente si vous souhaitez.

Vous aurez un rapport ainsi que le code de votre projet à fournir en fin de semestre (date en cours de décision). Petite précision, le rapport devra contenir la maquette que vous aurez réalisée.

2 Contexte

Les petits débrouillards est une organisation à but non lucratif qui sensibilise un public principalement très jeune (généralement 5-15 ans) à la sciences. Un peu comme l'émission "C'est pas sorcier", ils parcourent la France organisant de nombreux évènements de vulgarisation scientifique pour les plus jeunes. Pour plus d'information à leur sujet je vous invite à vous rendre sur leur page internet : cliquez ici

Le pôle d'Avignon de cette organisation, parmi ses nombreuses activités de vulgarisation scientifique, présente une activité traitant du problème du voyageur de commerce (TSP), qui vous sera brièvement expliquée dans la suite de ce document. Leur objectif n'est bien évidemment pas de faire comprendre ce problème complexe à des tout petits mais de leur montrer qu'il existe des problèmes compliqués pour lesquels nous allons être obligés d'optimiser notre démarche pour trouver la meilleure solution. Notamment en utilisant l'outil informatique, mais aussi en réfléchissant dans notre manière de procéder (tester toutes les combinaisons n'est pas toujours possible même pour un ordinateur).

Le **TSP (Travelling Salesman Problem)**, ou encore Problème du voyageur de commerce, est le problème où l'on cherche à connaître le chemin le plus court passant par un ensemble de points donnés. Problème très présent dans la réalité dans bon nombre de domaines et facilement explicable à un enfant : le facteur doit passer par toutes les boites aux lettres, le père-noël à toutes les maisons, le chat doit faire tomber tous les objets qui sont en hauteur dans la maison, etc... Pour présenter ce problème aux enfants, *les Petits Débrouillards d'Avignon* utilise une planche en bois avec plusieurs clous (les points à atteindre). Une ficelle est utilisée pour que les enfants puissent essayer de tâtonner à trouver le chemin le plus court. Après plusieurs essais, on leur explique que

le nombre de combinaisons possibles est très grand et que même l'ordinateur ne pourrait toutes les tester en un temps acceptable (environ 20-25 clous). Ce pourquoi on est obligé de trouver des **algorithmes** plus efficaces pour espérer trouver la solution dans un temps raisonnable. L'interface interviendra donc à ce moment.

3 Objectifs

Cette interface, dont vous aurez un **prototype fonctionnel** à réaliser, a pour but de présenter le déroulement d'un algorithme, explicable en grandes lignes à un enfant (sur le principe au moins). Le code, en Java, de cet algorithme vous est bien évidement intégralement fourni, sous forme d'une librairie (.jar). Votre tâche est la réalisation d'un prototype de cette interface de présentation, pas de résoudre les problèmes NP.

Votre programme sera en **MVC** (ou, disons Modèle-Vue, le contrôleur appartenant ici à la vue de toute manière). Le modèle est la partie qui cherche la meilleure solution au TSP proposé. Il vous est fourni intégralement et vous n'avez pas à y toucher. Si une fonctionnalité supplémentaire et simple vous est nécessaire sur le modèle, faites m'en la demande à l'adresse suivante : titouan.parcollet@univ-avignon.fr. Si cela est réalisable facilement et que l'on reste dans le cadre du sujet (si c'est utile pour améliorer votre interface) je me chargerai de vous intégrer la fonctionnalité. La partie Vue et Contrôleur donc, est votre objectif.

Voici une liste des fonctionnalités **minimales** de l'interface que vous devez implémenter :

- **Le déroulement de l'algorithme doit être affiché à l'écran** : Le graphe "courant" doit être visible et mis à jour à chaque action de l'algorithme; un deuxième graphe montrera aussi quel est, en chaque instant, la meilleure solution trouvée (dès l'instant où une première solution de chemin est proposée). Il faudra donc deux graphes sur votre interface. Ces graphes sont à coder à l'aide de la librairie GraphStream : cliquez ici. L'utilisation de GraphStream est **obligatoire**.
- **L'information dois être comprise par un humain** : l'algorithme se déroule très rapidement, il est obligatoire de le faire ralentir pour pouvoir le visualiser convenablement. Ceci étant, différentes vitesses d'exécution pourraient être utiles. La vitesse doit donc être contrôlable.
- **L'utilisateur doit pouvoir facilement proposer un TSP qu'il souhaite résoudre**. C'est à dire **un ensemble de points** pour lequel il souhaite trouver le plus court chemin. On se limite UNIQUEMENT à des points sur un repère orthonormé. La distance entre ces points est la distance Euclidienne (gérée par le modèle, de toute manière, vous n'avez pas à calculer les

distances). Au moins deux méthodes pour cela doivent être intégrer : par lecture d'un fichier CSV, et par une entrée de coordonnées brutes depuis l'interface (type tableau). D'autre méthodes peuvent/devraient être imaginées, elles seront considérées comme des fonctionnalités supplémentaires.

- L'utilisateur doit pouvoir **lancer** l'algorithme, le **mettre en pause**, le faire **repartir**, l'**arrêter complètement**, le **redémarrer du début**. Par contre, l'algorithme ne peut pas revenir en arrière. Si vous souhaitez absolument pouvoir repartir en marche arrière vous pouvez enregistrer les différentes étapes du graphe au niveau de la vue pour le faire mais cela n'est pas demandé et me semble peu utile (cela serait néanmoins considéré comme une fonctionnalité supplémentaire si vous justifiez de son utilité).

Ce travail est à réaliser en binôme. Vous utiliserez pour votre interface du JavaFX ou Swing (à vous de choisir). Les groupes devront être définis à la première séance de TP. Un rapport sera à rendre.

4 Etapes de développement de votre interface

4.1 Première partie - Réflexion

Cette partie, obligatoire, est une phase où vous effectuez vos choix quant à l'interface proprement dite. Prenez en considération le contexte et les fonctionnalités demandées pour établir premièrement une maquette, un plan provisoire du prototype d'interface que vous allez réaliser. Essayez de penser comment optimiser, en terme d'ergonomie, chaque contrôleur (outil de l'interface qui permet à l'utilisateur de lancer une action sur le modèle) ainsi que la vue (l'information qui est transmise à l'utilisateur). Adaptez vous au contexte dans vos choix d'éléments, de textes, d'images. Visualisez chaque fonctionnalité de l'interface et choisissez, de manière temporaire au moins, comment vous souhaitez les intégrer.

La maquette ainsi réalisée (graphiquement, sans code, sans NetBeans, ect...) sera à intégrer à votre rapport, même si celle-ci ne correspond plus par la suite à vos choix finaux. Il peut simplement s'agir d'une photo ou scan d'un dessin réalisé sur papier (un minimum propre et lisible tout de même). Ajoutez-y de brèves informations quant au fonctionnement complet choisi durant cette première étape. Ce rapport doit seulement être clair et lisible. Des points pourront être retirés s'il est absent ou vraiment trop négligé (phrases incompréhensibles, langage SMS...). Plus d'informations sur son contenu complet sont détaillés dans la quatrième partie.

4.2 Deuxième partie - Intégration de GraphStream et TSP-Model_PtiDeb et lancement de l'algorithme.

Avant de se lancer dans l'interface complète vous allez commencer par une simple interface contenant seulement un graphe et un bouton.

En première étape, vous allez intégrer un graphe (librairie GraphStream) à votre interface et lui faire afficher une liste de points.

Une fois cela réalisé, importez la librairie TSPModel_PtiDeb.jar à votre projet, pour créer une instance de la classe TSPModel_PtiDeb. Le MVC ici sera géré par le jeu de classe abstraite Observable/Observer de java.util.*. La classe TSPModel_PtiDeb qui vous est fournie dans la librairie TSPModel_PtiDeb.jar est une classe qui extends java.util.Observable (héritage), votre interface doit "implements" l'interface java.util.Observer.

Votre interface doit être renseignée en paramètre du construction du model. Le constructeur du modèle enregistrera votre vue en temps qu'observateur. Lorsque les données du modèle changeront et devront être mises à jour sur la/les vues, l'Observable (le modèle) réveille tous ses Observateurs (votre vue) pour prévenir que les données ont changées. Cela aura pour conséquence de lancer la fonction "update()" des "Observer" (votre vue), que vous devrez @Override et implémenter.

Lors de cette update() vous devez tout d'abord déterminer quel type de changement a été effectué sur le modèle. On se limitera dans cette partie à seulement traiter les ajouts ou suppression de segment. À chacune de ces deux actions, votre graphe dois permettre de visualiser le nouveau graphe complet actuel. Les fonctions de récupération des données du modèle qui vous sont utiles pour update() dans votre vue sont dans le fichier TSPModel_PtiDeb.java (bien commentées,... si si je vous promets) ET sont aussi décrites en ici en page 7. Il faudra les utiliser correctement dans la fonction update() afin de pouvoir visualiser le déroulement de l'algorithme après l'avoir lancer en appuyant sur votre bouton. Le code source de toute la librairie TSPModel_PtiDeb.jar vous est fourni dans TSPModel_PtiDeb.zip afin que vous puissiez, par curiosité, en connaître son fonctionnement.

Après avoir créé le modèle, envoyez lui vos points via la fonctions dédiée et lancer le dans un Thread. La classe TSPModel_PtiDeb est une "Runnable" et sa fonction run() est la recherche de TSP entre les points fournis. Vous pouvez directement l'intégrer dans un Thread et lancer ce thread avec start().

Pour clôturer cette partie, ralentissez l'algorithme afin de pouvoir vous même visualiser son déroulement. Bon à savoir : après avoir effectuer un notifyObserver, l'Observable (le modèle) attend que son observateur ait fini d'exécuter l'update() qu'il a provoqué avant de continuer ses processus. Vous n'avez donc

aucun risque de "manquer" une mise à jour, ou qu'elles ne s'effectuent pas dans l'ordre.

Renseignements complémentaires au sujet des identifiants des points et des segments : les identifiant des points, nombres entiers, seront choisis depuis la vue (par vous), qui lance la création des points au modèle (en les nommant par un identifiant qui doit être unique). Le modèle génère automatiquement les identifiants des segments qu'il crée. Il vous sera nécessaire de bien récupérer l'identifiant du segment donné par le modèle lors d'un ajout de segment, au niveau de l'identifiant que vous attribuerez à l'Edge correspondant. Vous vous resservirez de cet identifiant lors de la suppression de ce même Edge sur le graphe lorsque nécessaire.

4.3 Troisième partie - Autres fonctionnalités principales

Intégration des autres fonctionnalités principales.

Commencez par le deuxième graphe qui permet la visualisation du meilleur chemin trouvé. Le message envoyé par le modèle à ce moment là est simplement le type d'action, qui est un NewBest. Vous devez donc, dans votre fonction d'update, correctement gérer les actions NewBest. Vous n'avez pas besoin que le modèle vous renvoie plus d'informations.

Pour le reste, vous pouvez implémenter les autres fonctionnalités dans l'ordre que vous souhaitez.

Pour la gestion de la vitesse, vous avez normalement géré le ralentissement lors de la deuxième partie, il reste simplement à faire en sorte que ce ralentissement soit paramétrable depuis l'interface.

La récupération des points depuis un fichier ou même depuis l'interface ne devrait pas poser de soucis particuliers.

Les fonctionnalités liées à la mise en pause se feront à l'aide des fonctions setPause() / getPause. La fonction setPause permet modifier un booleen "pause" du modèle. Si ce dernier est sur "true", l'algorithme s'arrêtera directement à la prochaine itération (c'est un algorithme récursif). La pause sera provoquée par un "wait()" synchronisé sur le modèle. Pour la réactivation il faudra, premièrement remettre pause sur "false", puis ensuite réveiller le modèle qui dord (wait()) par un notify().

4.4 Quatrième partie - Fonctionnalités supplémentaires, ergonomie/esthétique et rapport

Intégration de fonctionnalités supplémentaires (cela peut être aussi de nouveaux renseignements visuels pour l'utilisateur).

Bien évidement je n'attends pas de vous que vous fassiez le strict minimum nécessaire, c'est à dire les fonctionnalités demandées de la section précédente, qui sont réalisables en 3 ou 4 heures. Trouvez des idées intéressantes, d'après le contexte, pour obtenir l'interface la plus en adéquation possible avec le contexte, et ergonomique. Décrivez-les dans votre rapport.

Vous devez noter aussi dans votre rapport les choix que vous avez effectués, tant au niveau ergonomique qu'esthétique. Ce rapport se construira en 3 parties :

- Maquette initiale et fonctionnement (durant la première partie, avant de commencer l'implémentation)
- Fonctionnalités supplémentaires
- Choix effectués (esthétique et ergonomie)

Dans la troisième partie du rapport faites simplement une liste des choix effectués (qui ont été un minimum réfléchis) avec un petit paragraphe explicatif.

Par exemple :

"**Bouton Start :**

Nous avons choisi de représenter le bouton de démarrage de l'algorithme par un triangle vert car il s'agit d'une icône directement reconnue pour démarrer (une musique, une vidéo), même par les enfants les plus jeunes. Ce bouton est désactivé (grisé) dès lors que l'algorithme se lance. Cela permet à l'utilisateur de comprendre que ce bouton n'est pas fonctionnel lorsque l'algorithme est lancé."

5 Fonctionnalités de communication avec le modèle

Ici sont répertoriés les fonctionnalités du modèle qui vous est fourni ainsi que la manière de les utiliser.

Pour les accesseurs de la classe TSPModel_PtiDeb :

- ActionType getAction() : retourne le type de la dernière action effectuée (Add, Remove, NewBest, Finish)

- int getSegmentID() : retourne l'identifiant du segment concerné par la dernière action, quand nécessaire.
- int getSegmentP1() : retourne un point du segment concerné par la dernière action, quand nécessaire.
- int getSegmentP2() : retourne l'autre point du segment concerné par la dernière action, quand nécessaire.
- boolean getPause() : pour savoir si l'algo est actuellement en pause.

Pour les fonctionnalités de la classe TSPModel_PtiDeb :

- TSPModel_PtiDeb(Observer v) : Constructeur, prenant un Observer (votre vue) en paramètre pour l'enregistrer dans sa liste d'Observer directement à sa construction.
- run() : lance l'algo. La classe TSPModel_PtiDeb implements Runnable.
- addPoint(int ID, int x, int y) : ajoute un point d'identifiant ID et de coordonnée (x,y).
- setPause(boolean b) : met (b=true), ou enlève (b=false) la "pause".
- removePoint(int ID) : supprime le point d'identifiant ID
- clear() : supprime tout, points et segments.

  Fonctionnalités rajoutées le 17/02 sur demande étudiante (demandées le 11/02) :

- premierPoint(int ID) : définit le point d'identifiant ID (s'il existe) comme point d'origine (ou d'arrivée c'est pareil, le graphe n'est pas orienté). Un seul premier point peut être défini, un second lancement de "premierPoint(ID)" annule le précédent point choisi.
- deletePremierPoint() : annule l'existance d'un "premier point". Le chemin le plus court ne cherchera donc plus à avoir d'origine ou de destination précise.

6 Notation

6.1 Fonctionnel - 5 points

- Mon prototype est-il fonctionnel?
- Toutes les fonctionnalités de base demandée par le sujet du TP ont-elles étaient intégrées à l'interface?

- L'interface est-elle toujours active même lorsque les fonctions du modèle sont lancées ?

À titre indicatif mais non-strict, pour vous donner une idée du barème :
0: rien ne marche / 1: presque rien ne marche / 2: de sérieux problèmes persistent, par exemple l'interface bloque pendant l'exécution / 3: ça marche à peu près mais quelques soucis sont présents / 4: ça marche en utilisation "normale" mais on peut obtenir une erreur si on utilise les fonctionnalités principales accessibles sur l'interface d'une manière inattendue. / 5: ça marche parfaitement.

6.2 Fonctionnalités supplémentaires - 5 points

- Le prototype d'interface dispose t-il d'autres fonctionnalités qui n'ont pas clairement été spécifiées sur le sujet du TP ? Qui auront pu vous sembler utiles, que ce soit par le contexte, par les questions posées au "client" ou même lors de votre réalisation du prototype. Ce ne sont pas forcément que de nouvelles fonctions au sens stricte du terme, simplement de nouvelles valeurs ajoutées à l'interface qui n'ont pas été décrites par le sujet. En d'autres termes, il s'agit de valeurs ajoutées au niveau du Controller (nouvelles possibilités d'action de l'utilisateur) ou de la Vue (nouveaux renseignements utiles apportés à l'utilisateur).

- On parle ici de fonctionnalités liées à l'interface et non au modèle lui-même. Vous n'avez pas à modifier le modèle.

- Même si cela n'est pas d'usage en MVC, vous serez autorisés à effectuer, si vous en avez besoin pour une nouvelle fonctionnalité, de petits traitement dans les fonctions de votre vue.

- Certaines fonctionnalités évidentes et simples ont volontairement été omises dans le sujet.

Pour l'attribution des points, chaque fonctionnalité rajoutée rapporte en fonction de sa complexité et de son intérêt entre 0,5 et 2 points. Il faudra pour obtenir la totalité des points réaliser au moins une fonctionnalité demandant un réel effort de conception, sans quoi cette partie ne sera notée que sur 3 points. Par exemple, permettre la suppression de points depuis l'interface ne demande pas vraiment de sérieux efforts, même s'il faudra faire attention à certaines conséquences. Cela comptera dans les fonctionnalités supplémentaires pour 0,5.

6.3 Ergonomie d'utilisation - 3 points

- Des efforts ont-ils été fait dans le choix des composants (à l'exception des éléments purement graphiques, qui comptent pour la section suivante) et des méthodes utilisées pour faciliter l'utilisation de l'interface ?

- L'utilisation de votre interface est-il simple et intuitif, même pour un enfant ?

- Les fonctionnalités supplémentaires ajoutées peuvent venir impacter ces points, si elles améliorent l'ergonomie de l'interface.

À titre indicatif mais non-strict, pour vous donner une idée du barème :
0: aucun effort ressenti sur ces points / 1: efforts plutôt minimalistes ou peu pertinents / 2: de sérieux efforts pertinents ont été constatés / 3: la perfection (ou pas très loin)

6.4 Ergonomie visuelle - 3 points

- Incitation : les choix des différents éléments graphiques est-il pertinent pour les différentes fonctionnalités auxquelles ils sont rattachés ?
- Esthétique : l'interface est-elle adaptée visuellement pour l'utilisateur, le contexte et les spectateurs ?

À titre indicatif mais non-strict, pour vous donner une idée du barème :
0: aucun effort ressenti sur ces points / 1: efforts plutôt minimalistes ou peu pertinents / 2: de sérieux efforts pertinents ont été constatés / 3: "2"+Woaaw !

6.5 Satisfaction de l'utilisateur - 4 points

Vous présenterez, lors d'une des séances du TP suivant (je vous donnerai la date dès que possible), votre prototype d'interface à l'utilisateur.

- L'utilisateur final est-il satisfait du prototype que vous présentez ?

À titre indicatif mais non-strict, pour vous donner une idée du barème :
0: pas satisfait du tout / 1: moyennement satisfait / 2: satisfait / 3: très satisfait / 4 : très satisfait et impressionné