Compte rendu des Parties B : deuxième partie du semestre

Fait par Remy XU, numéro etudiant: 22123345.

Preambule:

Dans le Fichier CompteRenduXU.zip, il y a les fichiers du dossier TP7PartieB, ainsi que ce compte rendu. Dans le fichier App.java, l'ensemble des TODO ont été remplis sauf les deux de la méthode main.
Il n'y a aucun ajout supplémentaire sur ces consignes la.

J'ajoute également en supplément un dossier appelé TP7. Il s'agit d'un code que j'ai implémenté pour la partie B du TP7 que j'ai mal compris: Je pensais qu'il fallait implémenter l'exercice 2.9 en java. Donc j'ai fait ma propre version d'un graphe pondéré et une tentative du code de l'algorithme de Dijkstra.

Algorithme A*:

```
//TODO: mettre tous les noeuds du graphe dans la liste des noeuds a
visiter:
```

Ici, j'ai simplement copié les valeurs de graphe.vertexlist dans to_visit.

```
//TODO: Remplir l'attribut graph.vertexlist.get(v).heuristic pour tous
les noeuds v du graphe:
```

Pour obtenir l'heuristique, on peut soit choisir une heuristique distance euclidienne, distance de Manhattan ou bien une heuristique avec Dijkstra. Cependant, ce dernier rendrait la complexité du code très lourde.

Ici, on privilégie l'utilisation d'une distance euclidienne.
Pour cela, on veut accéder à aux coordonnées en x et en y de chaque nœud et de celui du nœud d'arrivée end. Ensuite cela nous permettra d'obtenir le vecteur puis de calculer la distance.

//numCase est la case où est situé le nœud dans le graphe de l'UI.
//On suppose ici qu'il est situé à l'indice i, par manque d'information et d'impossibilité de modifier la structure du code.

```
//TODO: trouver le noeud min_v parmis tous les noeuds v ayant la
distance temporaire
//      (graph.vertexlist.get(v).timeFromSource + heuristic) minimale.
```

Rien à signaler en plus du code.

```
//TODO: pour tous ses voisins, on v�rifie si on est plus rapide en
passant par ce noeud.
```

Rien à signaler en plus du code.

```
//TODO: remplir la liste path avec le chemin
```

On ajoute les nœuds au chemin en suivant les pères successifs en partant du père de end.
On aurait pu raccourcir le code et supprimer l'attribut prev de la classe, si on ajoutait les nœuds de path pendant l'algorithme.
Cependant, on construit path après l'algorithme A*.

Algorithme Dijkstra:

Remarques identiques à celles de A*

main:

```
//TODO: obtenir le fichier qui décrit la carte
```

Le fichier étant dans le même dossier que le programme, on ajoute simplement de nom de graphe.txt
Si cela ne fonctionne pas, on peut toujours mettre l'arborescence dans l'ordinateur.

```
//TODO: ajouter les aretes
```

Non fait.

```
//TODO: laisser le choix entre Dijkstra et A*
```

On laissera le choix en entrée utilisateur: A pour A* et D pour Dijsktra.