RAPPORT TP1

Intelligence artificielle

RÉALISÉ PAR :

HANAFI Hamza & TABASSAMET Hamza

ENCADRÉ PAR

M'Hamed AIT KBIR

Plan

Introduction

But :

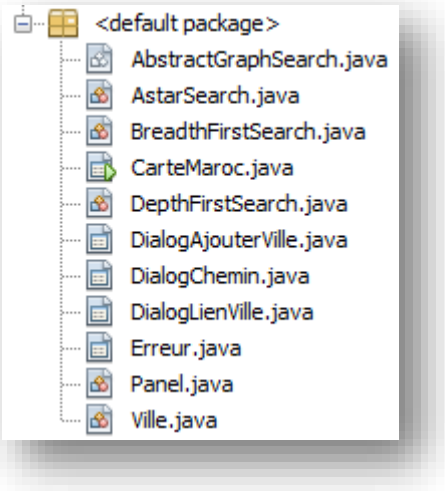
On veut créer une application en java permettant de trouver le chemin entre deux villes du Maroc.

Le problème est représenté sous forme de graphe d'états et met en application les stratégies de recherche : recherche en profondeur, recherche en largeur, et la recherche informée (A*).

1. Déclaration

a) Les classes :

Voici un schéma qui représente l'ensemble des classes utilisées pour la création de l'application :



L'ensemble des classes est regroupés dans le package par défaut de notre application et nommé : <default package>

1. *La classe AbstractGraphSearch :*

C'est une classes abstraite et implémente l'interface Serializable, elle contient les attributs permettant la gestion d'un graphe, les nœuds sont représentés sous forme d'un ArrayList de type Ville *(static ArrayList<Ville> villes)*, et les liens entre les villes sont des tableaux de type int *(protected static int[] link_1 = new int[MAX], protected static int[] link_2 = new int[MAX]).*

Les villes portent le mot clé *(static)* car ils seront communs entre toutes les instances des classes qui dérivent de la classe AbstractGraphSearch.

2. *La classe ville :*

Cette classe représente les nœuds du graphe, et permet l'accès aux informations d'une ville grâce à des méthodes *(setters & getters)*, une ville est caractérisée par un nom, et sa position dans la carte.

3. *La classe BreadthFirstSearch :*

C'est une classe enfant de la classe AbstractGraphSearch et permet d'appliquer l'algorithme de la recherche en largeur elle contient en paramètre 3 listes :

```
ArrayList<Ville> L = new ArrayList<Ville>(), T = new ArrayList<Ville>();
LinkedList<Ville> file = new LinkedList<Ville>();
```

L : une liste qui contient les nœuds déjà visité.

T : ensemble des états buts.

File : une fille des nœuds à explorer.

La méthode *connected_nodes(int node) :*

Cette méthode retourne un tableau de type *Ville* contenant toutes les villes connectées à l'indice de la ville passé en paramètre.

La méthode *doBFS() :*

Applique l'algorithme de la recherche en largeur.

La méthode *findPath(Ville start_node, Ville goal_node) :*

Retourne sous forme d'un tableau de villes le chemin entre les villes passées en paramètre.

4. *La classe DepthFirstSearch :*

C'est une classe enfant de la classe AbstractGraphSearch et permet d'appliquer l'algorithme de la recherche en largeur elle contient en paramètre 3 listes :

```
ArrayList<Ville> L = new ArrayList<Ville>(), T = new ArrayList<Ville>();
LinkedList<Ville> pile = new LinkedList<Ville>();
```

L : une liste qui contient les nœuds déjà visité.

T : ensemble des états buts.

Pile : une fille des nœuds à explorer.

La méthode *doDFS() :*

Applique l'algorithme de la recherche en profondeur.

5. *La classe AstarSearch :*

C'est aussi une classe enfant de la classe AbstractGraphSearch et permet d'appliquer l'algorithme de la recherche informée, elle contient en paramètre deux villes : ville de départ et ville d'arrivé *(public static Ville start, goal)* et une liste qui sera remplie par les villes par les quelles passe la route entre les deux villes (start, goal)

La méthode *findPath(Ville start_node, Ville goal_node)* :

Permet de trouver le chemin entre la ville *start_node* et Ville *goal_node,* et retourne ce chemin sous forme de tableau de villes.

6. *La classe CarteMaroc :*

C'est la classe principale de notre application, elle permet de mettre en pratique toutes les fonctionnalités de l'application, elle hérite de la classe *javax.swing.JFrame* pour la gestion de l'interface graphique et implémente les interfaces : *MouseListener, KeyListener, Serializable* pour la gestion des écouteurs (Listener) de l'application, et aussi pour l'enregistrement et lecture des villes et liens depuis les fichiers.

7. *La classe Panel :*

C'est une classe enfant de la classe *JPanel,* elle permet de dessiner les villes et les liens dans la carte ainsi que le chemin entre deux villes à l'aide de la méthode *paintComponent(Graphics g).*

8. *La classe Erreur :*

C'est une classe qui hérite de *javax.swing.JFrame* et permettant la gestion de quelques erreurs, elle prend en paramètre le message d'erreur initialisé dans le constructeur de la classe.

b) Interface graphique :

A l'exécution du programme la carte du Maroc apparait au milieu de l'écran, à la suite on peut ajouter de nouvelles villes et de nouveaux liens, et aussi on peut avoir le chemin entre deux villes.

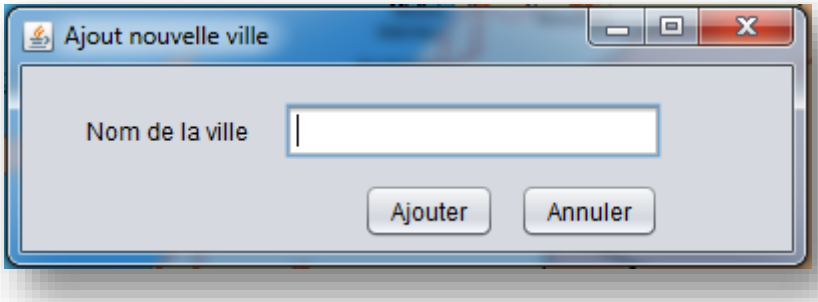
Clic droit : ajout d'une nouvelle ville.

Clic gauche : ajout d'un lien entre deux villes.

Bouton espace : choix de l'algorithme de recherche, et fixation des villes (départ et arrivée).

2. Ajout d'une ville :

La classe *DialogAjouterVille* hérite de *javax.swing.JFrame* permet d'ajouter une nouvelle ville et l'insérer dans la liste des villes *(villes.add(v))* lors d'un clic gauche sur la carte



Une petite fenêtre apparaît pour entrer le nom de la ville.

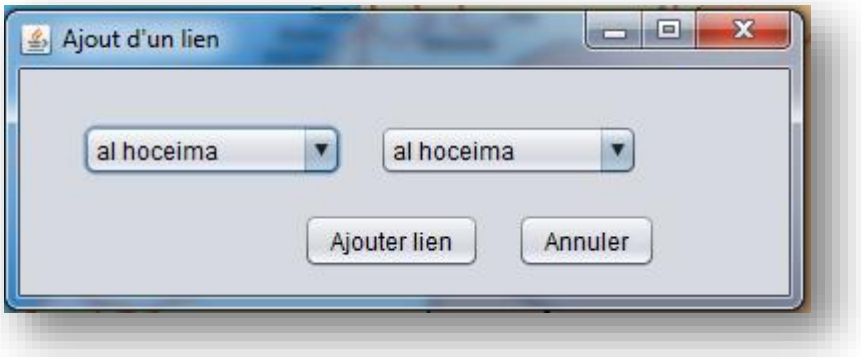
A la fin de l'ajout de la ville, il y'a une actualisation de la carte avec l'instruction *(carteMaroc.panel.repaint())* pour dessiner un point rouge à la position du clic.

Si l'utilisateur n'a pas saisie le nom de la ville un message d'erreur s'affiche !

3. Ajout d'un lien entre deux villes :

La classe *DialogLienVille* hérite de *javax.swing.JFrame* permet d'ajouter une liaison entre deux villes voisines lors d'un clic droit sur la carte

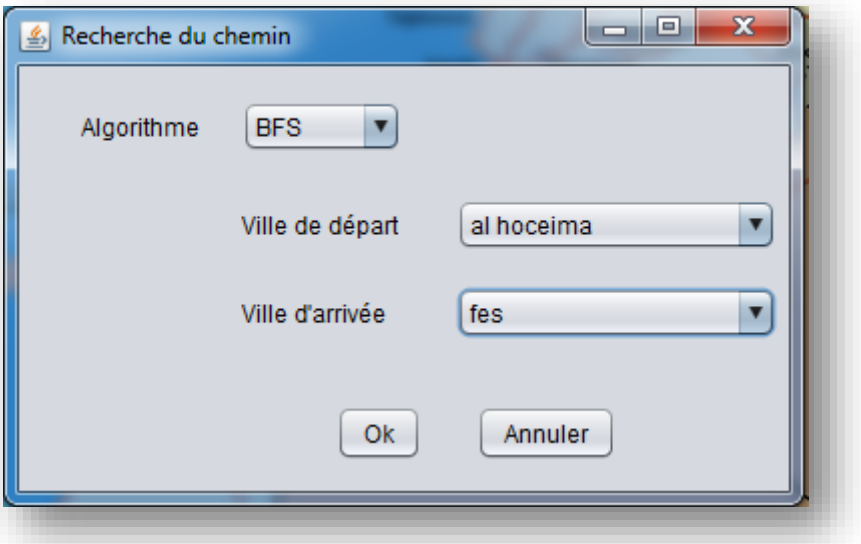
Une petite fenêtre apparaît pour indiquer le nom des deux villes.



A la fin il y'a une actualisation de la carte avec l'instruction *(carteMaroc.panel.repaint())* pour dessiner une liaison en noire entre les deux villes.

4. Recherche du chemin :

La classe *DialogChemin* hérite de la classe *javax.swing.JFrame*, elle donne à l'utilisateur la main pour indiquer la ville de départ et la ville d'arrivée quand il tape le bouton *espace*, et aussi le choix de l'algorithme à utiliser lors de la recherche



Conclusion

Ce TP nous a permet de mettre en pratique nos connaissances en JAVA, et aussi la mise en œuvre des stratégies de recherche vus en cours.