tp1.cpp File Reference

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <math.h>
#include <iterator>
#include <list>
```

Classes

class	**noeud**
class	**Matrice**
class	**ASTAR**

Macros

#define	**interrupteurON** 1
#define	**interrupteurOFF** 0
#define	**myPair** pair<char, pair<double, double>>
#define	**maisonVisitee** 'M'
#define	**interrupteurBriseSignale** 'B'
#define	**condRepare** 'R'

Enumerations

enum	**codeRetour** { **echoue**, **reussi** }
enum	**reparation** { **necessaire**, **pasNecessaire** }

Functions

string	**lectureNomFichier** (int argc, char *argv[])
string	**lectureContenuFichier** (string nomFichier)
double	**getNbrLigne** (string contenuFichier)
double	**getNbrColonne** (string contenuFichier)
double	**posToIndex** (**Matrice** *m, **noeud** c)
void	**trouveCible** (**Matrice** *m, char carac, vector< **noeud** > *liste)
bool	**deplacementPossibleCourant** (**Matrice** *m, **noeud** c, **noeud** dest)
bool	**deplacementPossibleEquipe** (**Matrice** *m, **noeud** c)

noeud	**deplacementNord** (**Matrice** *m, **noeud** c, **noeud** dest, int flag)
noeud	**deplacementSud** (**Matrice** *m, **noeud** c, **noeud** dest, int flag)
noeud	**deplacementOuest** (**Matrice** *m, **noeud** c, **noeud** dest, int flag)
noeud	**deplacementEst** (**Matrice** *m, **noeud** c, **noeud** dest, int flag)
void	**reparerConducteur** (**ASTAR** *a)
void	**allumerInterrupteur** (**ASTAR** *a)
void	**eteindreInterrupteur** (**ASTAR** *a)
double	**fheuristique** (**noeud** c, **noeud** dest)
bool	**sontIdentiques** (**noeud** a, **noeud** b)
bool	**estDansListe** (**noeud** c, vector< **noeud** > vect)
void	**ajusteQualite** (**ASTAR** *a, **noeud** c, double nouveauCout)
noeud	**trouveMeilleurQualite** (**ASTAR** *a)
double	**coutParent** (**ASTAR** *a, **noeud** c)
void	**trouveVoisins** (**Matrice** *m, **ASTAR** *a, **noeud** c, **noeud** dest, int flag)
void	**ajouteListeFermee** (**ASTAR** *a, **noeud** c)
noeud	**trouveNoeud** (**ASTAR** *a, pair< double, double > precedant)
void	**trouverChemin** (**ASTAR** *a, **Matrice** *m, **noeud** depart, **noeud** destination, int flag)
void	**deplacementCourantOUequipe** (**Matrice** *m, **ASTAR** *a, **noeud** depart, **noeud** destination, int flag)
bool	**tierVect** (pair< int, **noeud** > a, pair< int, **noeud** > b)
vector< int >	**trierSources** (**Matrice** *m, **ASTAR** *a)
string	**cheminOptimal** (**Matrice** *m, **ASTAR** *a, vector< string > vect)
bool	**contientInterrupteurOFF** (string chemin)
void	**reparationNecessaire** (string chemin)
vector< myPair >	**filtreActions** (**Matrice** *m, **ASTAR** *a)
vector< myPair >	**trouveActions** (vector< myPair > vect)
vector< myPair >	**condBrisesInterCommuns** (vector< myPair > vect)
void	**uneOuPlusieursSource** (vector< myPair > vect, **Matrice** *m, **ASTAR** *a)
void	**cheminPossible** (**ASTAR** *a)
void	**validation** ()
void	**imprimeResultat1** (**Matrice** *m, **ASTAR** *a)
void	**imprimeResultat2** (**Matrice** *m, **ASTAR** *a)
vector< **noeud** >	**trierSelonDistance** (**Matrice** *m, **ASTAR** *a)
int	**main** (int argc, char *argv[])

Variables

vector< codeRetour >	**vectCodeRetour**

vector< reparation > **vectReparation**

Detailed Description

Travail pratique #1 : Recherche dans un espace d'états / Rétablir l'électricité Auteur : Fadi Feghali - FEGF07069109

Function Documentation

◆ ajouteListeFermee()

```
void ajouteListeFermee ( ASTAR *  a,
                         noeud    c
                       )
```

Fonction pour ajouter un noeud à la liste fermée et le supprimer de la liste ouverte

Parameters

a **ASTAR** de la matrice

c un noeud

◆ ajusteQualite()

```
void ajusteQualite ( ASTAR *  a,
                     noeud    c,
                     double   nouveauCout
                   )
```

Fonction qui ajuste la qualité d'un noeud dans la liste ouverte s'il a une moins bonne qualité

Parameters

a **ASTAR** de la matrice

c un noeud

nouveauCout le nouveau coût (la distance)

◆ allumerInterrupteur()

void allumerInterrupteur (**ASTAR** * a)

imprime 1 quand un interrupteur est ouvert

Parameters

a **ASTAR** de la matrice

◆ cheminOptimal()

string cheminOptimal (**Matrice** * m,
ASTAR * a,
vector< string > vect
)

Fonction qui compare le nombre d'actions de plusieurs chemins menant à la même destination, et et qui choisit le meilleur = moins d'actions

Parameters

m la matrice

a **ASTAR** de la matrice

vect le chemin en forme de string

Returns

le chemin qui contient le moins d'actions

◆ cheminPossible()

void cheminPossible (**ASTAR** * a)

Fonction qui sert à imprimer le résultat final (la série d'actions à faire)

Parameters

a **ASTAR** de la matrice

◆ condBrisesInterCommuns()

vector<myPair> condBrisesInterCommuns (vector< myPair > vect)

Fonction qui sert à detecter les interrupteurs communs pour un seul bris

Parameters

vect vector qui contient des paires : caratère du noeud + ses coordonnées dans la grille

◆ contientInterrupteurOFF()

bool contientInterrupteurOFF (string chemin)

Fonction qui cherche la présence d'un interrupteur OFF dans le chemin donné

Parameters

chemin le chemin donné

Returns

vrai ou faux

◆ coutParent()

double coutParent (**ASTAR** * a,
noeud c
)

Fonction qui retourne le coût G du parrent d'un noeud

Parameters

a **ASTAR** de la matrice

c un noeud

Returns

le coût du parent (h + f)

◆ deplacementCourantOUequipe()

```
void deplacementCourantOUequipe ( Matrice *  m,
                                  ASTAR *    a,
                                  noeud      depart,
                                  noeud      destination,
                                  int        flag
                                )
```

Fonction qui fait appel à la fonction trouverChemin pour trouver le chemin optimal de la source vers la destination

Parameters

m	la matrice
a	**ASTAR** de la matrice
depart	le noeud de depart
destination	le noeud destination
flag	utile pour distinguer entre deplacementPossibleCourant ou deplacementPossibleEquipe

◆ deplacementEst()

```
noeud deplacementEst ( Matrice *  m,
                       noeud      c,
                       noeud      dest,
                       int        flag
                     )
```

deplacement possible vers l'est?

Parameters

m	la matrice
c	un noeud
dest	la destination
flag	1 pour courant et 0 pour équipe d'entretien

Returns

noeud voisins

◆ deplacementNord()

```
noeud deplacementNord ( Matrice *  m,
                        noeud      c,
                        noeud      dest,
                        int        flag
                      )
```

deplacement possible vers le nord?

Parameters

m	la matrice
c	un noeud
dest	la destination
flag	1 pour courant et 0 pour équipe d'entretien

Returns

noeud voisins

◆ deplacementOuest()

```
noeud deplacementOuest ( Matrice *  m,
                         noeud      c,
                         noeud      dest,
                         int        flag
                       )
```

deplacement possible vers l'ouest?

Parameters

m	la matrice
c	un noeud
dest	la destination
flag	1 pour courant et 0 pour équipe d'entretien

Returns

noeud voisins

◆ deplacementPossibleCourant()

```
bool deplacementPossibleCourant ( Matrice *  m,
                                  noeud      c,
                                  noeud      dest
                                )
```

Fonction pour trouver le chemin autorisé du courant (peut uniquement traverser des objets conducteurs)

Parameters

m la matrice

c un noeud

dest noeud destionation = maison

Returns

vrai ou faux

◆ deplacementPossibleEquipe()

```
bool deplacementPossibleEquipe ( Matrice *  m,
                                 noeud      c
                               )
```

Fonction pour trouver le chemin autorisé de l'équipe d'entretien (en évitant les obstacles)

Parameters

m la matrice

c un noeud

Returns

vrai ou faux

◆ deplacementSud()

```
noeud deplacementSud ( Matrice *  m,
                       noeud      c,
                       noeud      dest,
                       int        flag
                     )
```

deplacement possible vers le sud?

Parameters

m la matrice

c un noeud

dest la destination

flag 1 pour courant et 0 pour équipe d'entretien

Returns

noeud voisins

◆ estDansListe()

```
bool estDansListe ( noeud              c,
                    vector< noeud >    vect
                  )
```

Fonction qui détermine si un noeud est présent dans une liste donnée

Parameters

c un noeud

vect la liste

Returns

vrai ou faux

◆ eteindreInterrupteur()

void eteindreInterrupteur (**ASTAR** * a)

imprime 0 quand un interrupteur est fermé

Parameters

a **ASTAR** de la matrice

◆ fheuristique()

double fheuristique (**noeud** c,
noeud dest
)

Fonction heuristique pour quantifier la qualité des noeuds : On calcule la distane entre le point étudié et le dernier point qu'on a jugé comme bon et on calcule aussi la distance entre le point étudié et le point de destination. La somme des ces deux distances donne la qualité du noeud. Heurisitique utilisée : la distance euclidienne

Parameters

c un noeud

dest la destination

Returns

la distance

◆ filtreActions()

vector<myPair> filtreActions (**Matrice** * m,
ASTAR * a
)

Fonction qui trouve le chemin optimal des sources vers les maisons et décide si une réparation de courant est nécessaire ou pas

Parameters

m la matrice

a **ASTAR** de la matrice

◆ getNbrColonne()

double getNbrColonne (string contenuFichier)

fonction qui trouve le nombre de colonne de la grille

Parameters

contenuFichier

Returns

le nombre de colonnes

◆ getNbrLigne()

double getNbrLigne (string contenuFichier)

Fonction qui trouve le nombre de ligne de la grille

Parameters

contenuFichier

Returns

le nombre de lignes

◆ imprimeResultat1()

void imprimeResultat1 (**Matrice** * m,
ASTAR * a
)

Fonction qui traite les actions et imprime la séquence d'actions permettant de rétablir l'électricité pour toutes les maisons

Parameters

m la matrice

a **ASTAR** de la matrice

imprimeResultat2()

```
void imprimeResultat2 ( Matrice *  m,
                        ASTAR *  a
                      )
```

Fonction qui traite les actions et imprime la séquence d'actions permettant de rétablir l'électricité pour toutes les maisons

Parameters

m la matrice

a **ASTAR** de la matrice

lectureContenuFichier()

```
string lectureContenuFichier ( string  nomFichier )
```

Fonction pour lire le contenu du fichier fourni par l'utilisateur

Parameters

nomFichier

Returns

le contenu du fichier

lectureNomFichier()

```
string lectureNomFichier ( int       argc,
                           char *    argv[]
                         )
```

Fonction pour lire le nom du fichier de la ligne de commande

Parameters

argc nombre d'arguments

argv les arguments

Returns

le nom du fichier à taiter

◆ main()

```
int main ( int       argc,
           char *    argv[]
         )
```

La fonction principale

Parameters

argc nombre d'arguments

argv les arguments

Returns

0

◆ posToIndex()

double posToIndex (**Matrice** * m,
noeud c
)

fonction pour convertir une position (x,y) en indice

Parameters

m la matrice

c un noeud

Returns

l'indice du noeud

◆ reparationNecessaire()

void reparationNecessaire (string chemin)

Fonction qui decide si une réparation est nécessaire dans un chemin donné, si oui ajoute necessaire dans le vector vectReparation, sinon ajoute pasNecessaire

Parameters

chemin le chemin donné

◆ reparerConducteur()

void reparerConducteur (**ASTAR** * a)

imprime R quand un conducteur est réparé

Parameters

a **ASTAR** de la matrice

◆ sontIdentiques()

bool sontIdentiques (**noeud** a,
noeud b
)

Heuristique #2 -> carré de la distance euclidienne (pour tester seulement) car l'heuristique #1 est meilleure pour minimiser le nombre d'actions double **fheuristique(noeud c, noeud dest)** {

```
return ((c.pos.first - dest.pos.first) * (c.pos.first - dest.pos.first) +
    (c.pos.second - dest.pos.second) * (c.pos.second - dest.pos.second));
```

}

Fonction qui détermine si deux noeuds sont identiques en comparant leur positions

Parameters

a premier noeud

b deuxième noeud

Returns

vrai ou faux

◆ tierVect()

bool tierVect (pair< int, **noeud** > a,
pair< int, **noeud** > b
)

Fonction utile pour la fonction "sort" qui sert à trier un vector de pair

Parameters

a première paire

b deuxième paire

Returns

vrai ou faux

◆ trierSelonDistance()

vector<**noeud**> trierSelonDistance (**Matrice** * m,
ASTAR * a
)

Fonction qui sert à trier les maisons et les sources d'électricité en fonction de la distance de l'équipe d'entretien; du plus proche au plus loin

Parameters

m la matrice

a **ASTAR** de la matrice

◆ trierSources()

vector<int> trierSources (**Matrice** * m,
ASTAR * a
)

Fonction qui sert à trier les sources par rapport à leur distance de l'équipe d'entretien

Parameters

m la matrice m

a **ASTAR** de la matrice

Returns

les sources triées dans un vector

◆ trouveActions()

vector<myPair> trouveActions (vector< myPair > vect)

Fonction qui filtre les chemins, en enlevant les noeuds inutiles et en gardant les noeuds qui feront parti de la solution finale

Parameters

vect vector qui contient des paires : caratère du noeud + ses coordonnées dans la grille

◆ trouveCible()

```
void trouveCible ( Matrice *          m,
                   char               carac,
                   vector< noeud > *  liste
                 )
```

fonction qui trouve les positions des noeuds cibles : équipe d'entretien, interrupteurs, source d'électricité

Parameters

m	la matrice
carac	le caractère qui représente le noeud cible
liste	vector initialement vide

◆ trouveMeilleurQualite()

```
noeud trouveMeilleurQualite ( ASTAR *  a )
```

Fonction qui trouve le noeud qui a la meilleure qualité dans toute la liste ouverte.

Parameters

a ASTAR	de la matrice

Returns

le noeud avec la meilleur qualité

◆ trouveNoeud()

```
noeud trouveNoeud ( ASTAR *                 a,
                    pair< double, double >  precedant
                  )
```

Fonction qui sert à trouver un noeud dans la liste fermée en utilisant ses coordonnées

Parameters

a	**ASTAR** de la matrice
precedant	les coordonées

Returns

le noeud recherché

◆ trouverChemin()

```
void trouverChemin ( ASTAR *  a,
                     Matrice *  m,
                     noeud      depart,
                     noeud      destination,
                     int        flag
                   )
```

Fonction qui sert à trouver le chemin une fois la destination a été atteinte. On remonte les noeuds de parent en parent -> chaque noeud est ajouté en tête de la liste en utilisant push_front

Parameters

a	**ASTAR** de la matrice
m	la matrice
depart	le noeud de départ
destination	le noeud destination
flag	1 trouver le chemin de la position de l'équipe d'entretien vers la destination, 2 trouver le chemin de la/les source(s) vers la/les maison(s)

◆ trouveVoisins()

```
void trouveVoisins ( Matrice *  m,
                     ASTAR *    a,
                     noeud      c,
                     noeud      dest,
                     int        flag
                   )
```

Fonction qui sert à repérer les noeuds voisins et les ajoute à la liste ouverte

Parameters

- **m** la matrice
- **a** **ASTAR** de la matrice
- **c** un noeud
- **dest** la destination
- **flag** utile pour la fonction deplacementPossibleCourant et deplacementPossibleEquipe

◆ uneOuPlusieursSource()

```
void uneOuPlusieursSource ( vector< myPair >  vect,
                            Matrice *         m,
                            ASTAR *           a
                          )
```

Fonction qui sert à detecter la présence de plusieurs sources dans le monde et appelle différentes fonctions en conséquence

Parameters

- **vect** vector qui contient des paires : caratère du noeud + ses coordonnées dans la grille
- **m** la matrice
- **a** **ASTAR** de la matrice

◆ validation()

void validation ( )

Fonction qui imprime IMPOSSIBLE en cas d'absence de solution