
Rapport de Projet

“On prend un âne, ou on fait venir un ingénieur français ?”



Réalisé par
LLUÍS Isaac, KELLNER Maxime et SOURGNES Matéo

Sous la direction de
JOUBERT Alain

Pour l'obtention du DUT informatique

ANNÉE UNIVERSITAIRE 2017-2018



UNIVERSITÉ
DE MONTPELLIER



REMERCIEMENTS

On remercie ici les personnes physiques ou morales qui nous ont aidés dans la réalisation de notre projet.



SOMMAIRE

Table des figures	3
INTRODUCTION	4
1. Cahier des charges	5
1.1. Présentation du sujet et analyse du contexte	5
1.2. Analyse des besoins fonctionnels	5
1.3. Analyse des besoins non fonctionnels	6
1.3.1. Spécifications techniques	6
1.3.2. Contraintes ergonomiques	7
2. Rapport technique	8
2.1 Conception	8
2.2 Réalisation	13
3. Résultats	17
3.1 Installation	17
3.2 Test/Validation	17
3.3 Manuel d'utilisation	18
4. Gestion de projet	20
5. Conclusion	21
6. Bibliographie	22
7. Annexes techniques	23
7.1 Annexe 1: Fonctionnement Algorithme A*	23
7.2 Annexe 2: Code de l'algorithme A*	25

Table des figures

Figure 1: Diagramme des cas d'utilisations	8
Figure 2: Diagramme de classes	9
Figure 3: Diagramme de séquences	11
Figure 4: Fenêtre de l'application	17
Figure 5: Une topologie	18
Figure 6: La topologie après l'exécution du logiciel	18

INTRODUCTION

“On prend un âne, ou on fait venir un ingénieur français ?” est un dicton corse utilisé par les constructeurs de routes qui laisse entendre qu’un âne peut tout aussi bien déterminer le chemin optimal d’une route qu’un ingénieur français.

Pourquoi parler d’un tel dicton ? En 2017, nous avons pu remarquer une organisation complexe mondiale qui a nécessité la mise en relation de nos différentes agglomérations par la création de réseaux routiers.

Dans le cadre de notre projet, nous tenterons donc d’établir un logiciel qui aura pour but de déterminer le meilleur tracé optimal pour une route. C’est-à-dire la route la moins coûteuse et la plus courte.

Il y aura donc un sérieux enjeu sur la prise en compte des contraintes physiques ainsi que des contraintes financières, que l’utilisateur imposera.

Dans un premier temps, nous décrirons l’élaboration du cahier des charges en insistant sur l’analyse des besoins fonctionnels et non fonctionnels.

Ensuite dans un second temps, nous établirons le rapport technique en traitant la conception puis la réalisation du projet. Par ailleurs, dans un autre temps, nous discuterons des résultats obtenus, en détaillant les tests de validité et le manuel d’utilisation. Et enfin, nous expliquerons le déroulement du projet sur sa planification.

1. Cahier des charges

1.1. Présentation du sujet et analyse du contexte

Le but de notre projet est de fournir un logiciel qui permettra de **déterminer le tracé optimal d'une route** ayant pour point de départ un point A et un point d'arrivée B.

À partir d'une topologie donnée ou non (dans ce cas-ci, elle sera générée), nous devons générer le meilleur tracé d'une route.

Par "meilleur tracé", nous sous-entendons qu'il faudra prendre en compte **les contraintes physiques** (pas de pentes trop fortes, pas de virages trop serrés...) et **les contraintes financières** (minimisation du coût de revient de la construction).

Pour cela, nous nous sommes orientés dans une analyse de l'existant afin d'établir diverses solutions et d'adapter la résolution de notre problème à notre contexte. Ainsi, nous avons pu découvrir des logiciels qui remplissent des tâches plus ou moins similaires comme "*Autodesk*", ou des logiciels fondés sur un **algorithme de pathfinding**.

1.2. Analyse des besoins fonctionnels

Le logiciel pourra être utilisé dans un cadre public, utilisé par le gouvernement, ou toutes autres entreprises qui posséderont le droit de construire des routes sur de très grands terrains.

Étant donné que nous donnons une topologie, et qu'un petit terrain ne conviendra pas vraiment à l'utilisation du logiciel, le plus simple serait (probablement) de construire la route sur une ligne droite. En effet, imaginons que nous prenions un terrain si petit, que finalement, les contraintes physiques et financières seraient si infimes qu'il ne serait pas utile d'utiliser le logiciel.

Celui-ci ne sera pas uniquement utile quand il faudra construire une route sur un terrain diversifié (avec des falaises, montagnes et rivières), pas une simple plaine.

L'utilisateur va pouvoir configurer le coût des types de routes (pont, route normale ou tunnel), l'algorithme va se baser sur cela pour déterminer le chemin le plus économique, si pour l'utilisateur, les tunnels sont hors de prix, l'algorithme va naturellement éviter d'en faire.

1.3. Analyse des besoins non fonctionnels

1.3.1. Spécifications techniques

Le problème principal de ce projet est le temps de traitement nécessaire à l'exécution du programme. En effet, tous les algorithmes de pathfinding ont une complexité assez grande et très dépendante de la taille du graphe qui est donné, c'est à dire la taille de l'image.

Ensuite, le fait qu'il faille avoir la topologie dans un format précis (la quantité de vert représente la hauteur, le bleu les rivières et le gris les routes), il faudra que l'utilisateur fournisse une topologie dans cet exact format, sinon l'algorithme ne fonctionnera pas. Entreprise, publique, privée, logiciel adapté à l'ordinateur, utilisation facile ? Topologie format précis

Nous observons ici même, un **problème basé sur la résolution de graphe pondéré**.

1.3.2. Contraintes ergonomiques

Le logiciel devra, bien entendu, s'adapter aux nouvelles technologies, et donc celles qui sont les plus utilisées comme la tablette ou bien l'ordinateur portable. Il ne sera pas très pertinent de le développer sur le smartphone qui n'est pas un outil de travail très adapté dans ce cas (petit écran, pas très pratique, et pas très sérieux).



De plus, chaque technologie, comme le pc ou bien la tablette, possède son propre système d'exploitation (Linux, Windows...) d'où notre utilisation de java dans le but d'optimiser le nombre de plateformes accessibles.

Nous devons également soigner l'interface graphique de notre logiciel qui devra être assez intuitive pour une utilisation simple, et cohérente avec le projet.

Elle devra donc nous permettre d'insérer la topologie si celle-ci est fournie, ou la créer si celle-ci ne l'est pas.

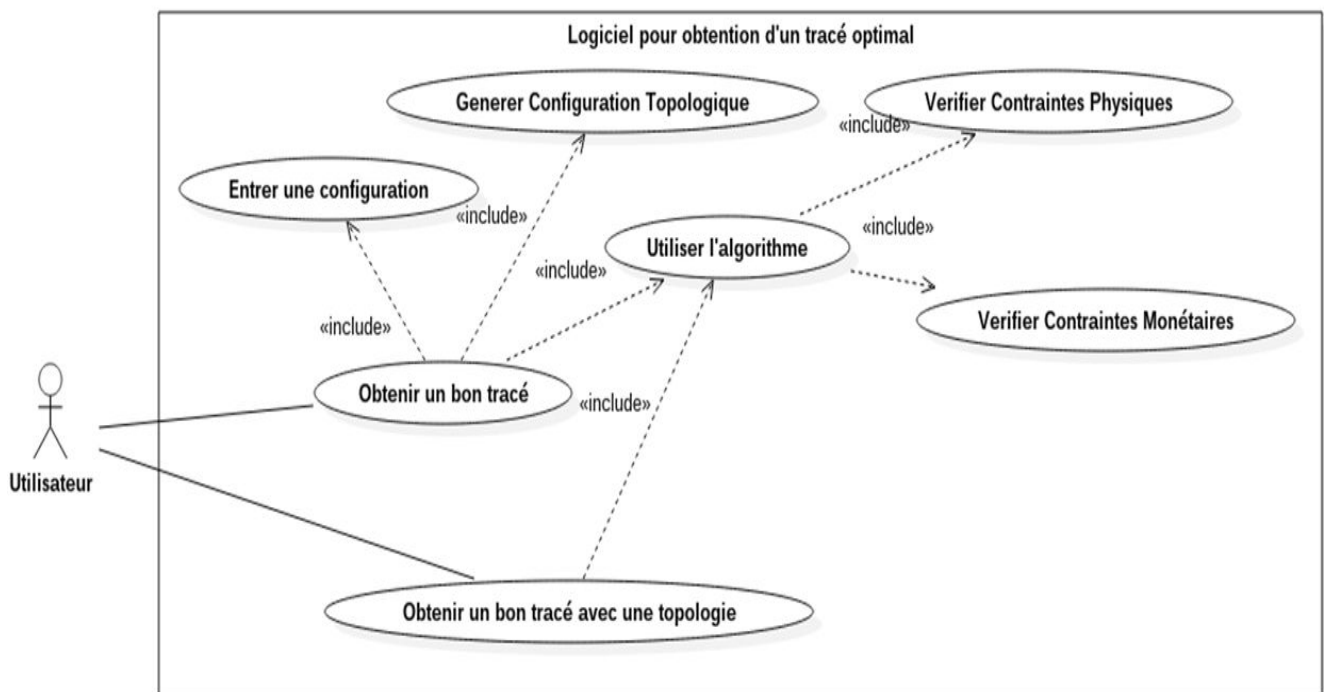
2. Rapport technique

2.1 Conception

Dans un premier temps, nous avons après étude de la problématique mûri les intentions réelles de l'utilisateur vis à vis de notre futur logiciel et pensé de manière à répondre à ses besoins.

Pour cela, nous avons mis à profit nos compétences sur la conception d'application dans le but d'élaborer un premier diagramme des cas d'utilisations (**Figure 1**) qui nous a permis d'avoir une vision globale du comportement fonctionnel du futur système et des acteurs qui rentreront en jeux.

Figure 1: Diagramme des cas d'utilisation



Nous pouvons donc observer que nous n'aurons qu'un acteur: **l'utilisateur**

Celui-ci, lors de l'usage de notre logiciel, pourra décider d'obtenir soit un tracé optimal avec une topologie qu'il aura fourni, ou bien, sans celle-ci.

Dans les deux cas, il y aura une utilisation de **l'algorithme A*** (*détaillé dans la partie réalisation*) qui s'exécutera en prenant en compte la topologie (*générée si non fournie*) et de la configuration (*contraintes physiques et financières*) renseignée par l'utilisateur .

Après cette analyse, nous avons donc une idée plus précise sur ce que nous allons devoir créer. Cependant, nous sommes arrivés dans la phase de codage du logiciel. Une phase qui sera dispatcher sur une partie de recherche de l'algorithme qui nous permettrait de résoudre notre problème, de documentation.

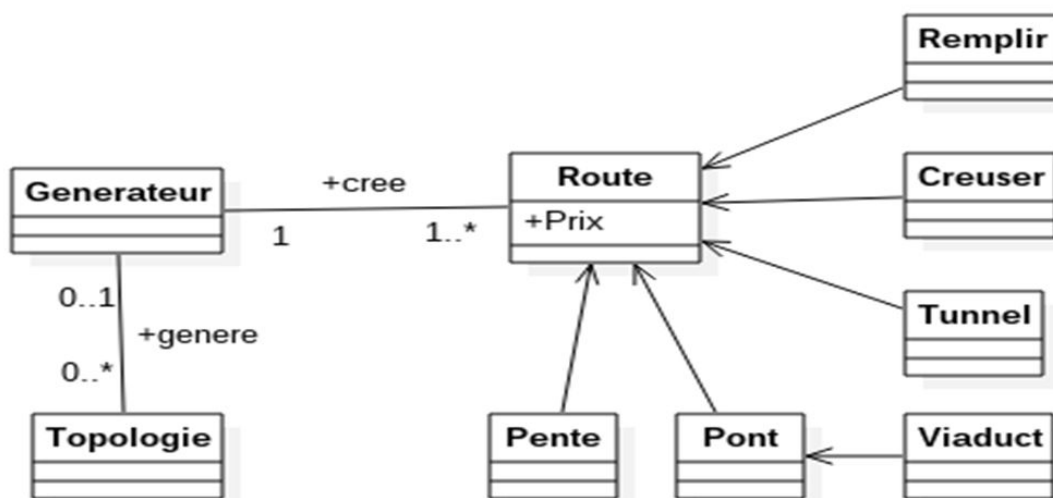
Ainsi, nous avons élaborer un diagramme des classes (**Figure 2**), pour savoir ce que nous aurions à coder.

Il faut bien comprendre que notre logiciel à la fin retournera la topologie avec le tracé optimal trouvé par le générateur (*sous-entendu l'algorithme A**).

Nous avons alors commencé à nous poser des questions sur les différentes manières d'obtenir un tracé optimal, car il est possible de creuser, ou bien de remplir (un creux si faisable), ou de faire un tunnel (si cela revient moins chère que de contourner "une montagne"), ou bien encore de faire un pont, tout en prenant en compte des contraintes physiques et financières imposées par l'utilisateur, tellement de possibilités à étudier.

Et nous avons pu en déduire le diagramme suivant:

Figure 2: Diagramme de classes





Celui-ci présente bien les classes dont nous aurons alors besoin si nous nous fions à notre logique. Lors de l'exécution de notre logiciel, nous avons décidé que celui-ci pourra alors générer des routes, faire des creux, remplir des creux, faire des ponts ou bien tunnels s'il jugeait cela moins coûteux et faisable (***mais toujours en respectant la configuration utilisateur***).

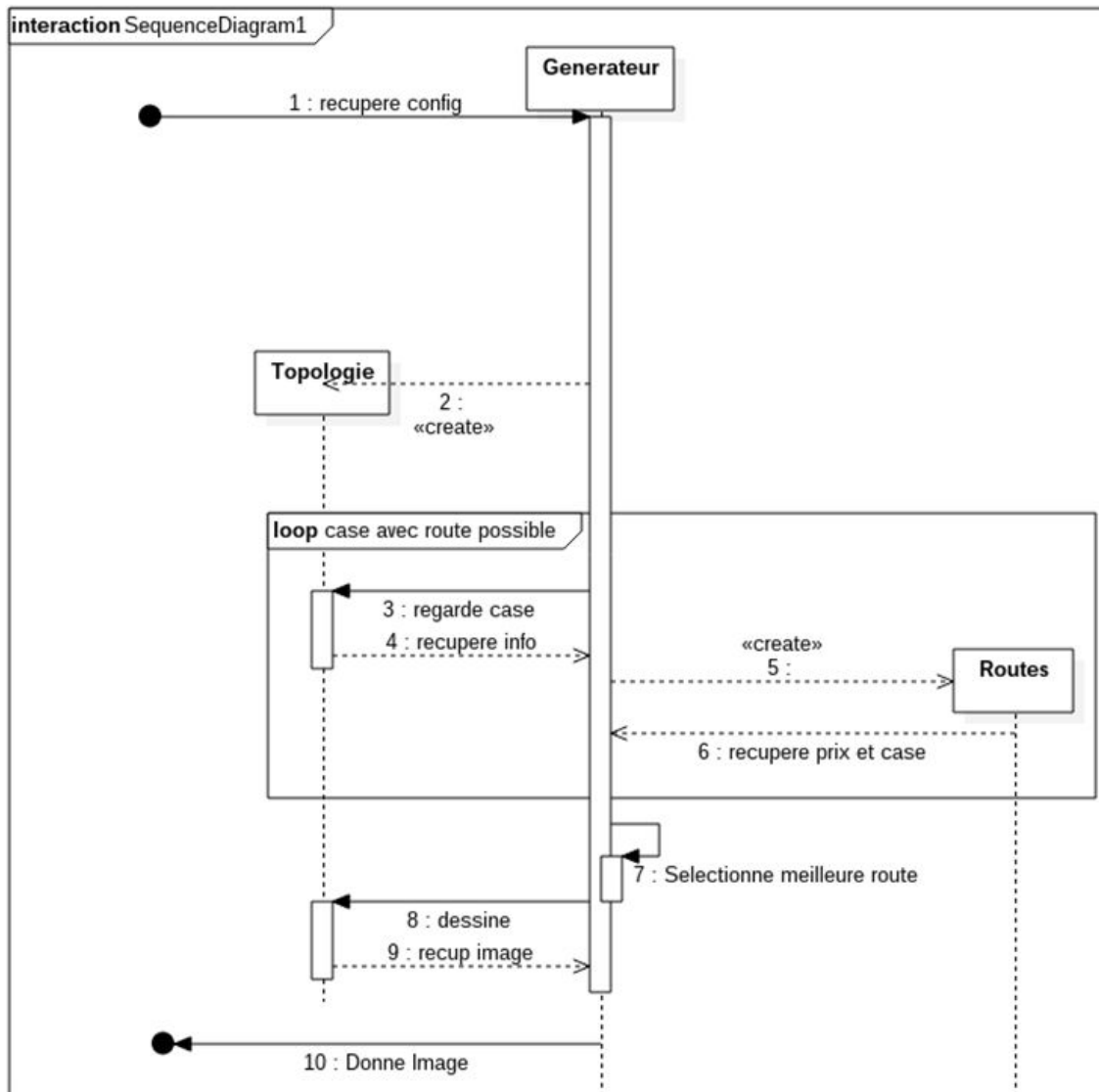
Après avoir compris ce qu'il y aurait à coder nous avons alors modéliser le fonctionnement de notre logiciel, et montrer les interactions internes au système auquel

nous pensions. Pour cela, nous nous sommes appuyés sur une schématisation par le diagramme de séquences.

Dans ce diagramme de séquence (***Figure 3***), on commence avec la récupération de la configuration par l'utilisateur, le prix de la route, des ponts, etc... Une fois cela récupéré, le programme va créer une topologie, en la générant aléatoirement, ou en la récupérant à l'aide d'un fichier.

Ensuite, l'algorithme est lancé sur cette topologie, il va regarder les cases une par une pour en vérifier l'information et trouver le chemin le plus optimal. Une fois cela fait, il va tracer la route finie sur l'image et va donner l'image à l'utilisateur.

Figure 3: Diagramme de séquences



2.2 Réalisation

2.2.1. Introduction

Dans la réalisation de ce projet, nous avons assez vite compris qu'il s'agissait de la résolution de graphe avec des paramètres divers qui entraient en jeux.

Et nous nous sommes donc directement tournés vers les algorithmes de pathfinding, or, de ces algorithmes, nous devons en choisir un qui correspondait à notre problématique, qui serait également compréhensible par l'équipe mais aussi modifiable.

Après diverses recherches, nous avons aboutis sur la découverte de l'algorithme A*, qui permet la résolution d'un graphe tout en prenant en compte certains paramètres.

Dans notre cas, nous voulions un algorithme qui nous permet d'établir un tracé optimal et donc le moins coûteux possible. Un tracé représente pour nous, le chemin établit entre un point A et B de la topologie donnée (ou non).

Cependant, nous avons pensé à une autre solution envisageable en cas de temps supplémentaire, c'est à dire utiliser un algorithme génétique avec des réseaux neuronaux afin d'obtenir un réseau neuronal optimisé pour le tracé de route.

Cela n'a rien à voir avec le pathfinding, et est beaucoup plus complexe à mettre en place, mais dans le cas ou ça aurait été fait, il y aurait beaucoup moins de temps de calcul, et la lenteur du programme serait amoindrie.

Pour comprendre le choix du réseau neuronal, il faut voir le problème non pas comme un graphe, mais comme une image qu'il faut résoudre. Avec une image, qui est la topologie, le réseau neuronal serait entraîné pour savoir ou mettre des routes dans une zone déterminée qui se déplacerait en fonction de la route.

2.2.2. Principe de L'algorithme A*

En sachant que pour trouver un chemin d'un point (point A) à un autre, il faut commencer par se diriger vers la destination (point B).



Avec l'algorithme A^* , à chaque itération (algorithme itératif), on va tenter de se rapprocher de la destination, nous allons donc avantager les possibilités directement plus proches de la destination, en écartant les autres.

Elles sont mises de cotées mais pas supprimées, nous les mettons dans une liste de possibilités à explorer si jamais la solution explorée actuellement est mauvaise. En effet, on ne peut pas savoir à l'avance si un chemin sera correcte ou sera le plus court. Il suffit que ce chemin amène à une impasse pour que cette solution devienne inexploitable.

L'algorithme choisit d'abord les chemins les plus directs et si ces chemins s'avèrent mauvais, il examinera alors les solutions mises dans la liste de possibilités.

C'est ce retour en arrière pour examiner les solutions dans la liste de probabilité qui nous garantit que l'algorithme nous trouvera toujours une solution.

On peut donc lui donner une topologie avec autant d'obstacles (pente, creux, rivières, ..., etc) qu'on veut, s'il existe une solution, A^* la trouvera.

2.2.3. Déroulement de L'algorithme A^*

On détermine donc assez facilement que A^* utilise deux listes (liste des cases déjà étudiées et celles pas encore étudiées). On peut également dire que ces listes contiennent des noeuds d'un graphe, ce graphe qui ,lui, représente notre topologie.

La liste ouverte, va contenir tous les noeuds étudiés. Dès que l'algorithme va vouloir étudier un noeud du graphe, il passera dans la liste ouverte (sauf si déjà présent).

L'autre liste, (liste fermée), contiendra tous les noeuds qui, à un moment où à un autre, ont été considérés comme faisant partie du chemin solution. Avant de passer dans la liste fermée, un noeud doit d'abord passer dans la liste ouverte, en effet, il doit d'abord être étudié avant d'être jugé comme étant correcte.

Pour déterminer si un noeud peut faire partie du chemin solution, nous devons pouvoir le mesurer (ici en fonction des paramètres voulues).

D'après nos recherches il existe une méthode souvent utilisée et qui donne de bons résultats, il s'agit de mesurer l'écartement entre ce noeud et le chemin à vol d'oiseau.

On calcule donc la distance entre la case étudiée et la dernière case jugée comme étant correcte.

Et on calcule aussi la distance entre la case étudiée et la case de destination (où se trouve le point B). La somme de ces deux distances nous permet de "mesurer" le noeud étudié.

Pour calculer ces distances on utilise soit la distance euclidienne, distance de Manhattan ou autre.

On ne connaît pas le chemin solution si ce n'est que la case qui pourra nous rapprocher de la solution c'est la case voisine de la case que nous étudions. On va donc étudier chacun des noeuds voisins du noeud courant pour connaître celui qui fera partie du chemin solution.

La recherche du chemin commence par la première case, en étudiant toutes ses voisines, en calculant leur qualité, et en choisissant la meilleure pour continuer.

Chaque case étudiée est mise dans la liste ouverte et la meilleure de cette liste passe dans la liste fermée, elle va servir de base pour la recherche suivante.

A chaque itération on regarde parmi les noeuds étudiés et on choisit, celui qui a la meilleure qualité. Si le meilleur n'est pas un voisin direct de la case courante. Cela signifie que la case courante nous éloigne de la solution.

L'algorithme s'arrête quand on atteint le point B ou bien lorsque toutes les solutions mises de côté ont été étudiées et qu'il n'y en a aucune de bonnes, soit le cas où la solution est inexistante.

Pour déterminer quel est le noeud voisin à suivre, on suit les étapes suivantes:

- Est-ce un obstacle ? si oui, nous oublions le noeud.
- Est-il dans la liste fermée ? si oui, ce noeud a déjà été étudié ou l'est encore
- Est-il dans la liste ouverte ? si oui, on juge le noeud, et s'il est meilleur que son homologue dans la liste ouverte, on modifie le noeud présent dans la liste ouverte.

Sinon, on l'ajoute dans la liste ouverte et on le juge à nouveau.



Afin de vous rendre la compréhension de **l'algorithme A*** plus accessible, vous pouvez vous référer au document fourni dans l'Annexe (***annexe 1 page***)

De plus, nous vous fournissons le code de l'algorithme commenté (***annexe 2 page***)

3. Résultats

3.1 Installation

Notre logiciel ne nécessitera pas d'installation particulière, nous avons cependant réellement fait attention à la portabilité du projet.

En effet, nous devons prendre au sérieux la réelle évolution technologique, de plus en plus d'entreprises se sont informatisées et il existe une multitude de système d'exploitation (**majoritairement Windows et Linux**).

Ainsi, il sera possible de faire usage du logiciel sur ces deux systèmes d'exploitations. Nous n'avons besoin d'installer ce logiciel, il n'y aura qu'à l'exécuter.

3.2 Test/Validation

Afin de rendre un projet qui répond à tous les critères du cahier des charges, nous avons mené une série de test. Nous avons dû tester si nos paramètres étaient bien pris en compte par notre algorithme.

Pour cela, nous avons dans un premier temps créé notre propre topologie puis déterminé le/les tracés optimaux.

Dans un second temps, nous avons fourni à notre logiciel notre topologie et avons observé si celui-ci nous fournissait à nouveau la même topologie avec un des tracés optimaux que nous avons pu trouver.

D'ailleurs plusieurs simulations ont été nécessaires car nous n'avions pas forcément tous les tracés possibles. Cependant, dans ces diverses simulations plusieurs de nos tracés sont apparus. Cela nous à permis de confirmer la prise en compte de nos paramètres par le logiciel que nous lui avons entré en configuration.

De plus, une fois le projet terminé (au niveau du code), nous avons poussé ces tests en simulant un certains nombre de fois l'utilisation du logiciel sur une topologie générée à chaque test par le logiciel.

Et avons constaté, si oui ou non, le tracé obtenu était solution (dans le cas où il en existait) et avons pu valider le projet.

3.3 Manuel d'utilisation

L'utilisation de notre logiciel est simple et nous l'avons penser de sorte à ce qu'elle soit assez intuitive ainsi nous avons optez pour une interface basique (les jeunes utilisateurs ne seront pas perdu lors de l'usage du logiciel).

Sur le premier écran, nous voyons deux boutons, des champs a remplir et un bouton valider. Pour configurer la route, c'est a dire les prix des routes, ponts, etc... il suffit de rentrer les valeurs voulues puis d'appuyer sur valider.

Ensuite, si la topologie est fournie, il suffit de cliquer sur "Donner une Topologie", puis de rentrer le lien et valider.

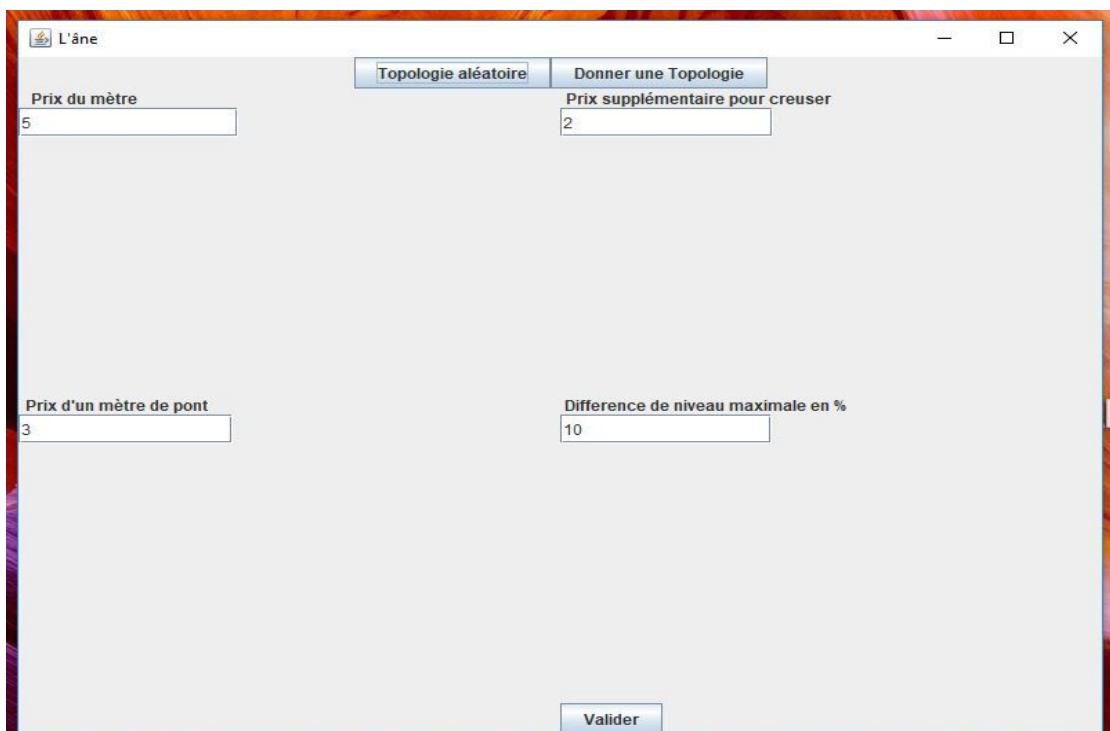
Si la topologie n'est pas fournie, il faut cliquer sur "Topologie aléatoire".

Dans les deux cas, la topologie s'affiche avec le lien de l'image, si la topologie est aléatoire, il y a possibilité de la générer à nouveau.

Après avoir cliqué sur le bouton "Exécuter", la topologie est régénérée avec le tracé optimal de la route déterminé par l'algorithme A*.

Le lien de l'image est aussi visible, afin de retrouver l'image.

Figure 4: Fenêtre de l'application



L'âne

Topologie aléatoire Donner une Topologie

Prix du mètre 5

Prix supplémentaire pour creuser 2

Prix d'un mètre de pont 3

Différence de niveau maximale en % 10

Valider

Figure 5: Une topologie

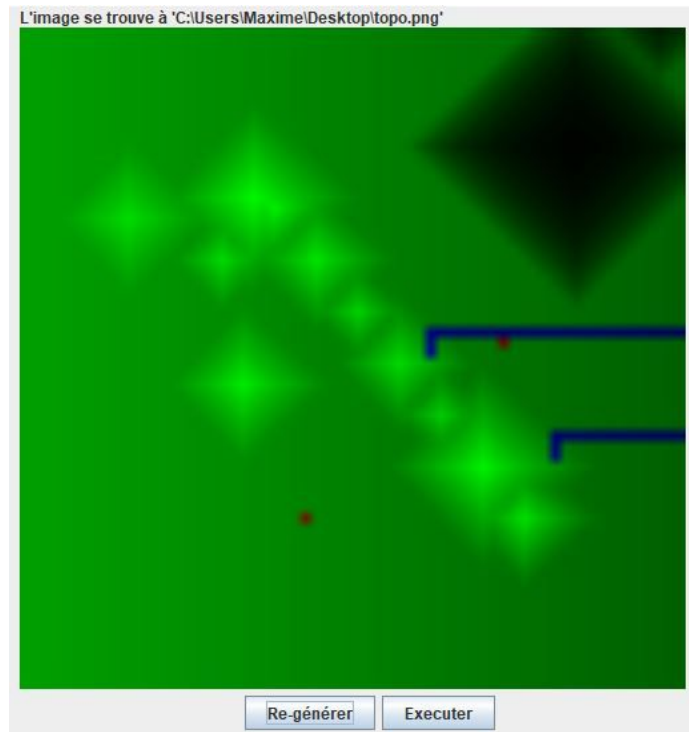
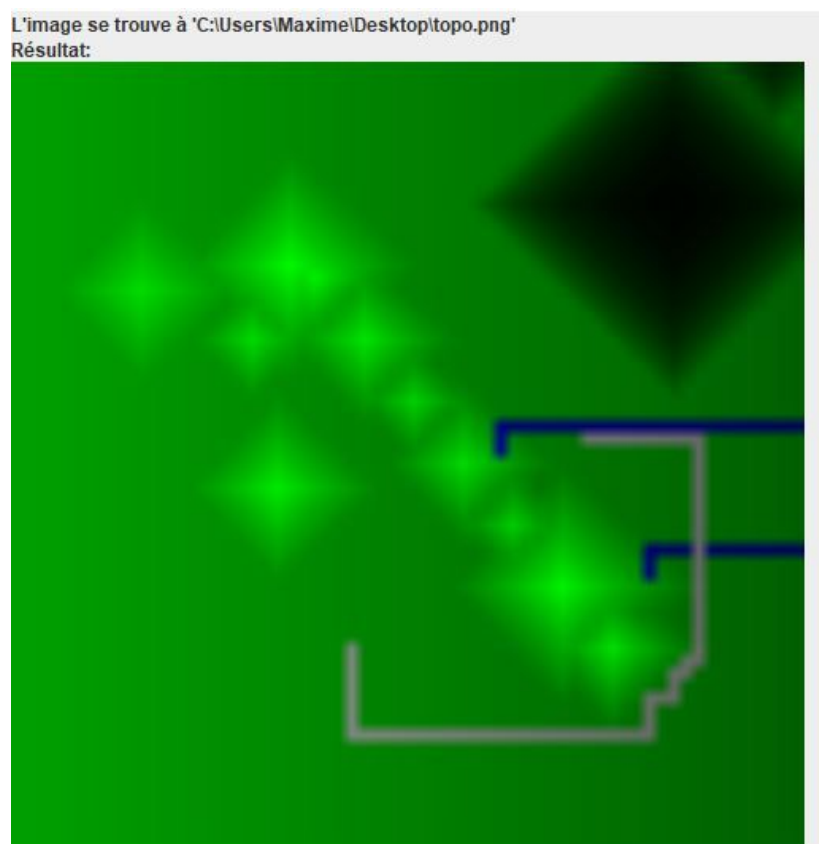


Figure 6: La topologie après l'exécution du logiciel



4. Gestion de projet

Pour la gestion de projet, nous n'avons pas jugé utile d'utiliser des outils comme Trello ou des diagrammes de Gantt.

Les raisons étant que nous sommes trois membres pour le projet. Tout ce qui est gestion de projet en agile nous semble difficile car chaque membre devrait savoir tout faire et non pas se spécialiser sur une partie. On ne peut pas se permettre d'avoir un scrum-master et un product-owner alors que nous ne sommes que trois.

Pour ce qui est du cahier des charges, nous le définissons au fur et à mesure, car nous connaissons depuis le début la direction générale du projet, qui est d'ailleurs très linéaire, il nous faut une partie pour passer à la suivante, et chaque étape est très floue quant à sa difficulté, donc nous ne pouvons pas prévoir le temps que ça prendra.

C'est un projet qui nécessite beaucoup de recherche — de l'algorithme de pathfinding notamment — et d'expérimentation, donc sans essayer, il est difficile de voir combien de temps ça prendra.

Cependant, nous nous sommes attribué des tâches de façon générale et nous avons beaucoup discuté de l'avancement du projet malgré le fait que nous avançons un peu séparément.

Afin de mener au mieux le projet, nous avons pour habitude d'aller à la rencontre de notre tuteur (***environ une fois par semaine***), afin de définir au mieux les tâches effectuées qui répondent à ce qui était demandé ainsi que celles qui ne l'étaient pas tout en définissant les nouvelles tâches.

Nous avons aussi utilisé Github pour le partage du code source ainsi que le Drive pour la partage des documents sources.

5. Conclusion

Pour conclure, nous sommes parvenu à atteindre les objectifs fixés, soit rendre un logiciel permettant d'obtenir le tracé optimal d'une route en prenant en compte des contraintes physiques et financières. Nous pouvons également renvoyer après exécution du logiciel la topologie fournie (qui a été générée ou non) avec le tracé de dessiné sur celle-ci. Il est alors acceptable de dire que le projet est finie.

La réalisation de ce projet a été très bénéfique pour nous , car il s'agit de la mise en communs de nos compétences et de leur concrétisation.

A cela s'ajoute, le fait que nous avons dû apprendre chacun à travailler avec des personnes que nous ne connaissions pas, ce qui nous permet d'avoir un avantage pour nos futurs projets (tels que ceux de notre stage futur), qui seront souvent un travail d'équipe avec du personnel qui ne se connaîtra pas forcément.

D'un point de vue plus technique, certains d'entre nous ont pu en apprendre plus sur la réalisation d'une interface graphique, ou bien d'un algorithme complexe (le principal problème du projet) sur la résolution de graphes avec des paramètres à prendre en compte.

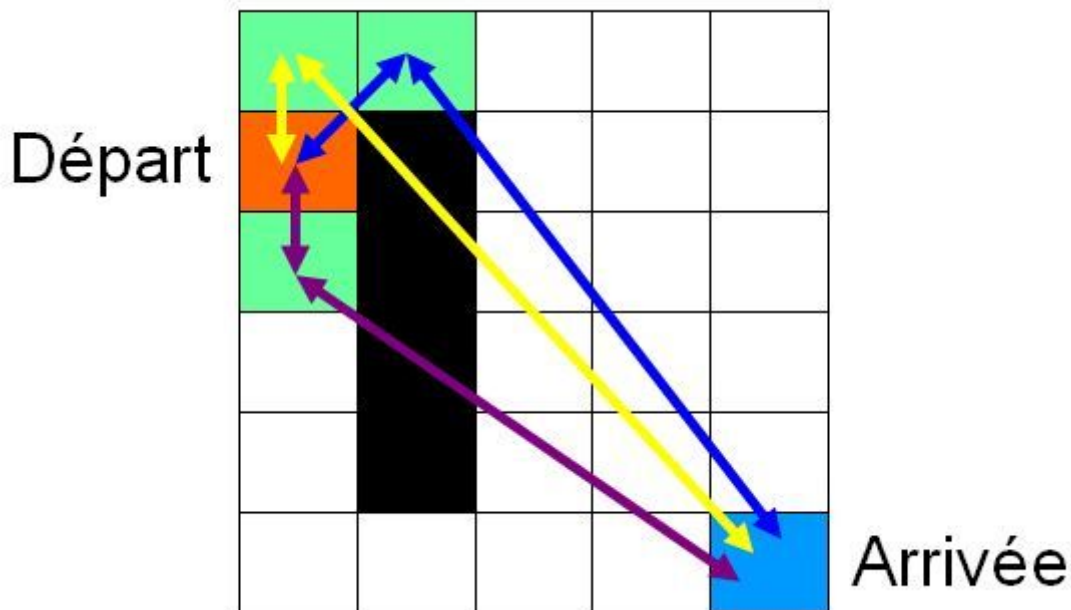


6. Bibliographie

- 1- [Explication de l'algorithme A* - Wikipédia](#)
- 2- [Explication plus détaillée de l'algorithme A* - aetoile.pdf](#)
- 3- [Réalisation interface graphique](#)

7. Annexes techniques

7.1 Annexe 1: Fonctionnement Algorithme A*



Nous pouvons voir cette image comme une topologie

- On commence par le noeud de départ, c'est le noeud courant
- On regarde tous ses noeuds voisins
- si un noeud voisin est un obstacle, on l'oublie
- si un noeud voisin est déjà dans la liste fermée, on l'oublie
- si un noeud voisin est déjà dans la liste ouverte, on met à jour la liste ouverte si le noeud dans la liste ouverte a une moins bonne qualité (et on n'oublie pas de mettre à jour son parent)
- sinon, on ajoute le noeud voisin dans la liste ouverte avec comme parent le noeud courant
- On cherche le meilleur noeud de toute la liste ouverte. Si la liste ouverte est vide, il n'y a pas de solution, fin de l'algorithme



-
- On le met dans la liste fermée et on le retire de la liste ouverte
 - On réitère avec ce noeud comme noeud courant jusqu'à ce que le noeud courant soit le noeud de destination.

Voici une illustration d'une itération de l'algorithme.

On souhaite aller du point orange au point bleu. Les noeuds voisins de la case orange sont les cases marquées en vert, ils passent en liste ouverte. Et de chacune d'elles on calcule les coûts G et H pour aller à la case orange et pour aller à la destination. J'ai choisi la distance suivant la configuration.

Et la case qui aura le coût le plus faible sera la case verte du dessous, elle va donc passer en liste fermée. L'algorithme va réitérer à partir de cette case.

7.2 Annexe 2: Code de l'algorithme A*

```
import java.util.ArrayList;
import java.util.List;

public class Algo {
    Topo topo;
    int[] d; //depart
    int[] a; //arrivée;
    Case c; //current
    List<Case> open = new ArrayList<Case>(); // cases à traiter
    List<Case> close = new ArrayList<Case>(); // cases traitées
    //variables d'informations
    int tourDeTest;

    public Algo(Topo topo,int[] d,int[] a) {
        this.topo = topo;
        this.d = d;
        this.a = a;
    };

    public void executerAlgo(int prixP, int prixT) {
        if (this.Path(prixP, prixT)) {
            this.affichageInformatif(this.faireRoute());
        } else {
            System.out.println("L'algo n'a pas pu trouver de solution");
        }
    }
}
```

```
public boolean Path(int prixP, int prixT){

    this.topo.find(d[0], d[1]).x = d[0];
    this.topo.find(d[0], d[1]).y = d[1];
    this.topo.find(d[0], d[1]).F = Math.abs(d[0] - a[0]) + Math.abs(d[1] - a[1]);
    this.open.add(this.topo.find(d[0], d[1]));
    this.c = this.topo.find(d[0], d[1]);

    //variable pour la selection de la case avec le F minimum
    int minF = 0;
    int IDminF = 0;
    //variables d'informations
    boolean cheminTrouve = false;
    int tourDeTest = 0;
    //variable pour pont/tunnel
    int longueurPont;
    int prixPont = prixP;
    int prixTunnel = prixT;
    int prix = 0;
    // (variable qui simplifie une modification future de l'algorithme)
    int multiplieurH = 1;

    while(!this.open.isEmpty() && !cheminTrouve){ // tant qu'il reste des cases à
traiter (open list) et que le chemin n'a pas déjà été trouvé

        tourDeTest++; // pour compter le nombre de tour de l'algo (à titre
informatif)

        for(int i = 0; i < this.open.size() ; i++){ // on prend la case avec le F
minimum

            this.c = this.open.get(i);

            if(minF == 0 || this.open.get(i).F < minF){
                minF = this.topo.find(this.c.x, this.c.y).F;
                IDminF = i;
            }

        }

    }

}
```

```
}
this.c = this.open.get(IDminF); // et on la stock dans c
//System.out.println(this.c.F + " " + this.c.x + this.c.y); // affichage
informatif

minF = 0;
IDminF = 0;

//~~~~~ case à gauche
~~~~~
~~~~~

if(this.c.x != 0 &&
!this.close.contains(this.topo.find(this.c.x-1,this.c.y)) && Math.abs(this.c.niv -
this.topo.find(this.c.x-1,this.c.y).niv)<10 ){ //si la case ne dépasse pas de la
matrice et n'est pas dans la close list et est accessible

//calcule case
=====|

if( this.open.contains(this.topo.find(this.c.x-1,this.c.y)) &&
this.topo.find(this.c.x-1,this.c.y).G < this.c.G + 1){ // si elle est déjà dans
l'open list et que son G est plus intéressant que c.G +1 alors il existe déjà un
chemin pour cette case plus optimisé
}else{
    this.open.remove(this.topo.find(this.c.x-1,this.c.y)); // on
enlève la case de l'open list car on recalcule ses variables
    this.topo.find(this.c.x-1,this.c.y).x = this.c.x-1;
    this.topo.find(this.c.x-1,this.c.y).y = this.c.y;
    this.topo.find(this.c.x-1,this.c.y).parent = this.c;
    this.topo.find(this.c.x-1,this.c.y).G = this.c.G + 1;
    this.topo.find(this.c.x-1,this.c.y).H = (Math.abs(this.c.x-1
- a[0]) + Math.abs(this.c.y - a[1])) * multiplieurH;
    this.topo.find(this.c.x-1,this.c.y).F =
this.topo.find(this.c.x-1,this.c.y).G + this.topo.find(this.c.x-1,this.c.y).H;
    this.open.add(this.topo.find(this.c.x-1,this.c.y));
} // fin calcule case
=====|

}else if (this.c.x != 0 &&
!this.close.contains(this.topo.find(this.c.x-1,this.c.y))){ //pont/tunel
```

```
longueurPont = 2; //remise à 2 LongueurPont (variable aussi
utilisé pour le tunnel)

if((this.c.niv - this.topo.find(this.c.x-1,this.c.y).niv) > 10){
// pont
    prix = prixPont;
    //System.out.println("pont gauche:" );
    while(this.c.niv -
this.topo.find(this.c.x-longueurPont,this.c.y).niv > 10 && this.c.x-longueurPont !=
0){ //calcule de la longueur du pont dans LongueurPont
        longueurPont++;
    }
}else{ //tunnel
    prix = prixTunnel;
    //System.out.println("tunnel gauche:");
    while((this.c.niv -
this.topo.find(this.c.x-longueurPont,this.c.y).niv < -10) && this.c.x-longueurPont !=
0){ //calcule de la longueur du tunnel dans LongueurPont
        longueurPont++;
        //System.out.println(this.c.x + "/" + this.c.y + "/" +
this.topo.find(this.c.x-longueurPont,this.c.y).niv);
    }
}
//calcule case (pont/tunnel) eventuel
=====|

if(this.open.contains(this.topo.find(this.c.x-longueurPont,this.c.y)) &&
this.topo.find(this.c.x-longueurPont,this.c.y).G < this.c.G + (prix*longueurPont)){
// si elle est deja dans l'open list et que son G est plus interessant que c.G + prix
du pont alors il existe deja un chemin pour cette case plus optimisé
    }else
if(!this.close.contains(this.topo.find(this.c.x-longueurPont,this.c.y))){

this.open.remove(this.topo.find(this.c.x-longueurPont,this.c.y)); // on enlève la
case de l'open list car on recalcule ses variables
    this.topo.find(this.c.x-longueurPont,this.c.y).x =
this.c.x-longueurPont;
    this.topo.find(this.c.x-longueurPont,this.c.y).y = this.c.y;
    this.topo.find(this.c.x-longueurPont,this.c.y).parent =
```

```
this.c;
                                this.topo.find(this.c.x-longueurPont,this.c.y).G = this.c.G
+ (prix*longueurPont);
                                this.topo.find(this.c.x-longueurPont,this.c.y).H =
(Math.abs(this.c.x-longueurPont - a[0]) + Math.abs(this.c.y - a[1])) * multiplieurH;
                                this.topo.find(this.c.x-longueurPont,this.c.y).F =
this.topo.find(this.c.x-longueurPont,this.c.y).G +
this.topo.find(this.c.x-longueurPont,this.c.y).H;

this.open.add(this.topo.find(this.c.x-longueurPont,this.c.y));
                                //System.out.println("Lg : "+LongueurPont +" / F :
"+this.topo.find(this.c.x-longueurPont,this.c.y).F);
                                }// fin calcule case
=====|
}

//~~~~~ case en haut
~~~~~

if(this.c.y != 0 &&
!this.close.contains(this.topo.find(this.c.x,this.c.y-1)) && Math.abs(this.c.niv -
this.topo.find(this.c.x,this.c.y-1).niv)<10 ){

                                //calcule case
=====|
                                if( this.open.contains(this.topo.find(this.c.x,this.c.y-1)) &&
this.topo.find(this.c.x,this.c.y-1).G < this.c.G + 1){
                                }else{
                                    this.open.remove(this.topo.find(this.c.x,this.c.y-1));
                                    this.topo.find(this.c.x,this.c.y-1).x = this.c.x;
                                    this.topo.find(this.c.x,this.c.y-1).y = this.c.y-1;
                                    this.topo.find(this.c.x,this.c.y-1).parent = this.c;
                                    this.topo.find(this.c.x,this.c.y-1).G = this.c.G + 1;
                                    this.topo.find(this.c.x,this.c.y-1).H = (Math.abs(this.c.x -
a[0]) + Math.abs(this.c.y-1 - a[1])) * multiplieurH;
                                    this.topo.find(this.c.x,this.c.y-1).F =
this.topo.find(this.c.x,this.c.y-1).G + this.topo.find(this.c.x,this.c.y-1).H;
                                    this.open.add(this.topo.find(this.c.x,this.c.y-1));
                                }// fin calcule case
```

```
=====|

    }else if (this.c.y != 0 &&
!this.close.contains(this.topo.find(this.c.x,this.c.y-1))) { //pont/tunel

        longueurPont = 2; //remise à 2 LongueurPont (variable aussi
utilisé pour le tunel)

        if((this.c.niv - this.topo.find(this.c.x,this.c.y-1).niv) > 10){
// pont
            prix = prixPont;
            //System.out.println("pont haut:");
            while(this.c.niv -
this.topo.find(this.c.x,this.c.y-longueurPont).niv > 10 && this.c.y-longueurPont !=
0){ //calcule de la longueur du pont dans LongueurPont
                longueurPont++;
            }
        }else{ //tunel
            prix = prixTunel;
            //System.out.println("tunel haut:");
            while(this.c.niv -
this.topo.find(this.c.x,this.c.y-longueurPont).niv < -10 && this.c.y-longueurPont !=
0){ //calcule de la longueur du tunnel dans LongueurPont
                longueurPont++;
            }
        }
        //calcule case (pont/tunel) eventuel
=====|

        if(
this.open.contains(this.topo.find(this.c.x,this.c.y-longueurPont)) &&
this.topo.find(this.c.x,this.c.y-longueurPont).G < this.c.G + (prix*longueurPont)){
// si elle est deja dans l'open list et que son G est plus interessant que c.G + prix
du pont alors il existe deja un chemin pour cette case plus optimisé
        }else
        if(!this.close.contains(this.topo.find(this.c.x,this.c.y-longueurPont))){

this.open.remove(this.topo.find(this.c.x,this.c.y-longueurPont)); // on enlève la
case de l'open list car on recalcule ses variables
            this.topo.find(this.c.x,this.c.y-longueurPont).x = this.c.x;
```

```

        this.topo.find(this.c.x, this.c.y-longueurPont).y =
this.c.y-longueurPont;
        this.topo.find(this.c.x, this.c.y-longueurPont).G = this.c.G
+ (prix*longueurPont);
        this.topo.find(this.c.x, this.c.y-longueurPont).parent =
this.c;
        this.topo.find(this.c.x, this.c.y-longueurPont).H =
(Math.abs(this.c.x - a[0]) + Math.abs(this.c.y-longueurPont - a[1])) * multiplieurH;
        this.topo.find(this.c.x, this.c.y-longueurPont).F =
this.topo.find(this.c.x, this.c.y-longueurPont).G +
this.topo.find(this.c.x, this.c.y-longueurPont).H;

this.open.add(this.topo.find(this.c.x, this.c.y-longueurPont));
        //System.out.println("lg "+longueurPont +" /
"+this.topo.find(this.c.x, this.c.y-longueurPont).F);
    } // fin calcule case
=====|

}

//~~~~~ case à droite
~~~~~
~~~~~

    if(this.c.x != this.topo.largeur-1 &&
!this.close.contains(this.topo.find(this.c.x+1, this.c.y)) && Math.abs(this.c.niv -
this.topo.find(this.c.x+1, this.c.y).niv)<10){

        //calcule case
=====|
        if( this.open.contains(this.topo.find(this.c.x+1, this.c.y)) &&
this.topo.find(this.c.x+1, this.c.y).G < this.c.G + 1){
        }else{
            this.open.remove(this.topo.find(this.c.x+1, this.c.y));
            this.topo.find(this.c.x+1, this.c.y).x = this.c.x+1;
            this.topo.find(this.c.x+1, this.c.y).y = this.c.y;
            this.topo.find(this.c.x+1, this.c.y).parent = this.c;
            this.topo.find(this.c.x+1, this.c.y).G = this.c.G + 1;
            this.topo.find(this.c.x+1, this.c.y).H = (Math.abs(this.c.x+1
- a[0]) + Math.abs(this.c.y - a[1])) * multiplieurH;
            this.topo.find(this.c.x+1, this.c.y).F =

```

```
this.topo.find(this.c.x+1,this.c.y).G + this.topo.find(this.c.x+1,this.c.y).H;
    this.open.add(this.topo.find(this.c.x+1,this.c.y));
} // fin calcule case
=====|

    }else if (this.c.x != this.topo.largueur-1 &&
!this.close.contains(this.topo.find(this.c.x+1,this.c.y))) { //pont/tunel

        longueurPont = 2; //remise à 2 LongueurPont (variable aussi
utilisé pour le tunel)

        if((this.c.niv - this.topo.find(this.c.x+1,this.c.y).niv) > 10){
// pont

            prix = prixPont;
            //System.out.println("pont droite:");
            while(this.c.niv -
this.topo.find(this.c.x+longueurPont,this.c.y).niv > 10 && this.c.x+longueurPont !=
this.topo.largueur-1){ //calcule de la longueur du pont dans LongueurPont
                longueurPont++;
            }
        }else { //tunel
            prix = prixTunel;
            //System.out.println("tunel droite:");
            while((this.c.niv -
this.topo.find(this.c.x+longueurPont,this.c.y).niv < -10) && this.c.x+longueurPont !=
this.topo.largueur-1){ //calcule de la longueur du tunnel dans LongueurPont
                longueurPont++;
                //System.out.println(this.c.x + "/" + this.c.y + "/" +
this.topo.find(this.c.x+longueurPont,this.c.y).niv);
            }
        }
        //calcule case (pont/tunel) eventuel
=====|

        if(
this.open.contains(this.topo.find(this.c.x+longueurPont,this.c.y)) &&
this.topo.find(this.c.x+longueurPont,this.c.y).G < this.c.G + (prix*longueurPont)){
// si elle est déjà dans l'open list et que son G est plus intéressant que c.G + prix
du pont alors il existe déjà un chemin pour cette case plus optimisé
        }else
if(!this.close.contains(this.topo.find(this.c.x+longueurPont,this.c.y))) {
```



```
this.open.remove(this.topo.find(this.c.x+longueurPont,this.c.y)); // on enlève la
case de l'open list car on recalcule ses variables
    this.topo.find(this.c.x+longueurPont,this.c.y).x =
this.c.x+longueurPont;
    this.topo.find(this.c.x+longueurPont,this.c.y).y = this.c.y;
    this.topo.find(this.c.x+longueurPont,this.c.y).parent =
this.c;
    this.topo.find(this.c.x+longueurPont,this.c.y).G = this.c.G
+ (prix*longueurPont);
    this.topo.find(this.c.x+longueurPont,this.c.y).H =
(Math.abs(this.c.x+longueurPont - a[0]) + Math.abs(this.c.y - a[1])) * multiplieurH;
    this.topo.find(this.c.x+longueurPont,this.c.y).F =
this.topo.find(this.c.x+longueurPont,this.c.y).G +
this.topo.find(this.c.x+longueurPont,this.c.y).H;

this.open.add(this.topo.find(this.c.x+longueurPont,this.c.y));
    //System.out.println("Lg : "+LongueurPont +" / F :
"+this.topo.find(this.c.x+LongueurPont,this.c.y).F);
    }// fin calcule case
=====|
    }

    //~~~~~ case en bas
~~~~~
~~~~~

    if(this.c.y != this.topo.hauteur-1 &&
!this.close.contains(this.topo.find(this.c.x,this.c.y+1))&& Math.abs(this.c.niv -
this.topo.find(this.c.x,this.c.y+1).niv)<10){

        if( this.open.contains(this.topo.find(this.c.x,this.c.y+1)) &&
this.topo.find(this.c.x,this.c.y+1).G < this.c.G + 1){
            }else{
                this.open.remove(this.topo.find(this.c.x,this.c.y+1));
                this.topo.find(this.c.x,this.c.y+1).x = this.c.x;
                this.topo.find(this.c.x,this.c.y+1).y = this.c.y+1;
                this.topo.find(this.c.x,this.c.y+1).parent = this.c;
                this.topo.find(this.c.x,this.c.y+1).G = this.c.G + 1;
```

```

        this.topo.find(this.c.x, this.c.y+1).H = (Math.abs(this.c.x -
a[0]) + Math.abs(this.c.y+1 - a[1])) * multiplieurH;
        this.topo.find(this.c.x, this.c.y+1).F =
this.topo.find(this.c.x, this.c.y+1).G + this.topo.find(this.c.x, this.c.y+1).H;
        this.open.add(this.topo.find(this.c.x, this.c.y+1));
    }
    }else if (this.c.y != this.topo.hauteur-1 &&
!this.close.contains(this.topo.find(this.c.x, this.c.y+1))) { //pont/tunel

        longueurPont = 2; //remise à 2 LongueurPont (variable aussi
utilisé pour le tunnel)

        if((this.c.niv - this.topo.find(this.c.x, this.c.y+1).niv) > 10){
// pont

            prix = prixPont;
            //System.out.println("pont bas:");
            while(this.c.niv -
this.topo.find(this.c.x, this.c.y+longueurPont).niv > 10 && this.c.y+longueurPont !=
this.topo.hauteur-1){ //calcul de la longueur du pont dans LongueurPont
                longueurPont++;
            }
        }else{ //tunel
            prix = prixTunel;
            //System.out.println("tunel bas:");
            while(this.c.niv -
this.topo.find(this.c.x, this.c.y+longueurPont ).niv < -10 && this.c.y+longueurPont !=
this.topo.hauteur-1){ //calcul de la longueur du tunnel dans LongueurPont
                longueurPont++;
            }
        }
        //calcul case (pont/tunel) eventuel
        =====/
        if(
this.open.contains(this.topo.find(this.c.x, this.c.y+longueurPont )) &&
this.topo.find(this.c.x, this.c.y+longueurPont ).G < this.c.G + (prix*longueurPont)){
// si elle est déjà dans l'open list et que son G est plus intéressant que c.G + prix
du pont alors il existe déjà un chemin pour cette case plus optimisé
        }else
if(!this.close.contains(this.topo.find(this.c.x, this.c.y+longueurPont))){

```

```
this.open.remove(this.topo.find(this.c.x, this.c.y+longueurPont )); // on enlève la
case de l'open list car on recalcule ses variables
        this.topo.find(this.c.x, this.c.y+longueurPont ).x =
this.c.x;
        this.topo.find(this.c.x, this.c.y+longueurPont ).y =
this.c.y+longueurPont ;
        this.topo.find(this.c.x, this.c.y+longueurPont ).G =
this.c.G + (prix*longueurPont);
        this.topo.find(this.c.x, this.c.y+longueurPont ).parent =
this.c;
        this.topo.find(this.c.x, this.c.y+longueurPont ).H =
(Math.abs(this.c.x - a[0]) + Math.abs(this.c.y+longueurPont - a[1])) * multiplieurH;
        this.topo.find(this.c.x, this.c.y+longueurPont ).F =
this.topo.find(this.c.x, this.c.y+longueurPont ).G +
this.topo.find(this.c.x, this.c.y+longueurPont ).H;
        this.open.add(this.topo.find(this.c.x, this.c.y+longueurPont
));
        //System.out.println("Lg "+longueurPont +" /
"+this.topo.find(this.c.x, this.c.y+longueurPont ).F);
    } // fin calcule case
=====|
    }

    if(this.c.x == a[0] && this.c.y == a[1]){ // cas d'arrêt
        this.tourDeTest = tourDeTest;
        cheminTrouve = true;
    }else{
        this.open.remove(this.c);
        this.close.add(this.c);
    }
}
return cheminTrouve;
}
```

```
public List<Case> faireRoute() {
    List<Case> optimal = new ArrayList<Case>();

    optimal.add(0, this.topo.find(d[0], d[1]));
    while(this.c.x != d[0] || this.c.y != d[1]){ // retourner et stocker
Le chemin dans la liste "optimal"
        optimal.add(0, this.c);
        this.c = this.c.parent;
    }

    return optimal;
}

public void affichageInformatif(List<Case> optimal) {
    // affichage informatif
    System.out.println("Chemin optimal : ");
    System.out.println "[" + d[0] + d[1] + "]" + " -> ";
    for (Case n : optimal) {
        System.out.println "[" + n.x + n.y + "]" + " -> ";
    }
    System.out.println("arrivée ! (longueur du chemin : " + optimal.size() +
" coût du chemin "+this.topo.find(a[0],a[1]).F + " )");
    System.out.println("Pour ce résultat l'algo à fait " + this.tourDeTest +
" tests");
}
}
```



UNIVERSITÉ
DE MONTPELLIER

