



TX N°6046

**Calcul de chemins pour le jeu Les
Aventuriers du Rail**

Laura Hénin - IM04
Thomas Lécluse - GI04

Semestre P19



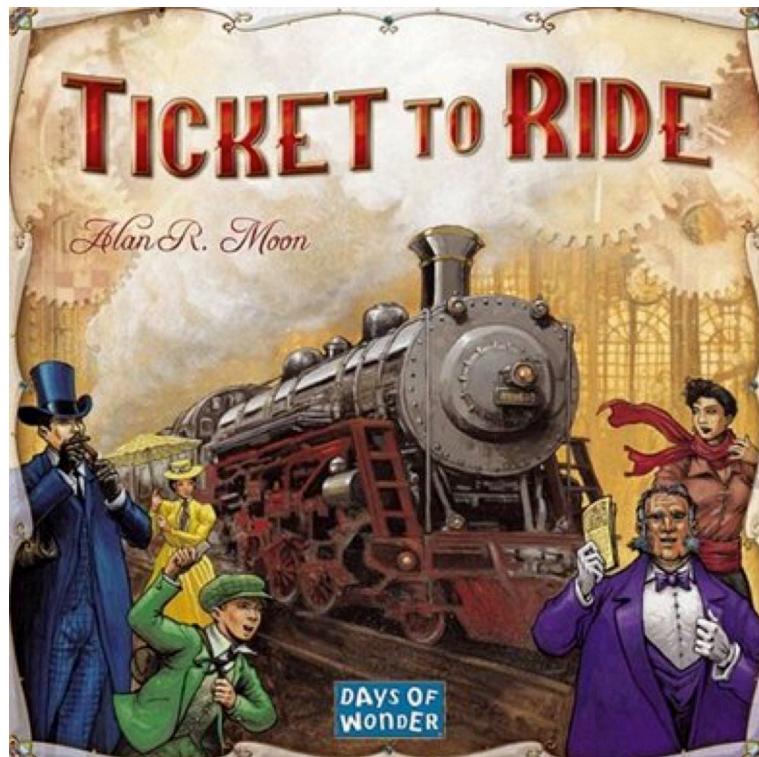
Table des matières

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Présentation du jeu | 2 |
| 1.2 | Contenu du jeu | 3 |
| 1.3 | But du jeu | 5 |
| 2 | Contexte du problème | 6 |
| 2.1 | Problématique | 6 |
| 2.2 | Paramètres du problème | 6 |
| 3 | Calcul de chemins | 7 |
| 3.1 | Déterminer le meilleur chemin entre deux destinations | 7 |
| 3.1.1 | Calcul de la rentabilité des chemins grâce au rapport $\frac{\text{coût}}{\text{gain}}$ | 7 |
| 3.1.2 | Introduire la notion de risque dans le calcul du rapport | 8 |
| 3.1.3 | Interprétation du risque | 9 |
| 3.1.4 | Bilan sur le calcul du meilleur chemin entre deux destinations | 10 |
| 3.2 | Algorithme : meilleur chemin reliant deux destinations | 11 |
| 3.2.1 | Données de l'algorithme | 11 |
| 3.2.2 | Explications | 12 |
| 3.2.3 | Complexité de l'algorithme | 13 |
| 3.2.4 | Exploitation de la liste des chemins | 13 |
| 3.3 | Prise en compte des possessions des joueurs | 14 |
| 3.4 | Prise en compte des tronçons doubles | 14 |
| 3.5 | Prise en compte des couleurs | 15 |
| 3.6 | Meilleur chemin avec étapes | 15 |
| 3.6.1 | Méthode avec combinaisons de chemins intermédiaires | 15 |
| 3.6.2 | Méthode avec la liste de tous les chemins | 16 |
| 4 | Implémentation en Java | 17 |
| 4.1 | Format de donnée | 17 |
| 4.2 | Conception des objets principaux | 18 |
| 4.3 | Choix d'implémentation | 20 |
| 4.4 | Interface graphique | 21 |
| 4.4.1 | Les panneaux latéraux | 22 |
| 4.4.2 | Panneau central | 23 |
| 4.4.3 | Configuration | 25 |
| 5 | Mise en conditions réelles | 26 |
| 6 | Conclusion | 27 |
| 6.1 | Améliorations | 27 |
| 6.2 | Bilan | 27 |

1 Introduction

1.1 Présentation du jeu

"Les aventuriers du rail" est un jeu de plateau de stratégie publié pour la première fois par Days of Wonder en 2004. Centré sur le thème du chemin de fer, ce jeu permet aux joueurs d'exploiter un réseau ferroviaire d'une région du monde afin de relier différentes villes et de marquer le plus de points possible.



Les joueurs devront utiliser à bon escient des wagons pour, d'une part, réaliser le plus de destinations possibles, et d'autre part, empêcher leurs concurrents de réaliser les leurs.

1.2 Contenu du jeu

Le jeu se compose de :

- **1 carte du réseau ferroviaire** de la région où figure le nom des grandes villes ainsi que les chemins sur lesquels les joueurs vont devoir placer leurs wagons et construire leur réseau.



FIGURE 1 – Plateau du jeu (région Europe)

- **45 wagons de chaque couleur** correspondant aux pions de chaque joueur et que ceux-ci vont placer sur le plateau pour indiquer la possession d'un chemin.



FIGURE 2 – Wagons de couleur



- **110 cartes wagon** nécessaires pour l'achat des chemins par les joueurs :

- 12 cartes wagon de chaque couleur (jaune, bleu, rouge, vert, blanc, rose, noir, orange)
- 14 locomotives pouvant substituer à n'importe quelle couleur (joker)



FIGURE 3 – Cartes wagon

- **46 cartes Destination** qui sont les objectifs que les joueurs doivent réaliser au cours de la partie.



FIGURE 4 – Carte Destination



1.3 But du jeu

Le but du jeu est d'être le joueur totalisant un maximum de points. Un joueur peut gagner des points de différentes manières :

- En plaçant ses wagons sur des tronçons reliant des villes (plus le tronçon est long, plus le nombre de points est élevé)
 - chemin de longueur 1 : 1 point
 - chemin de longueur 2 : 2 points
 - chemin de longueur 3 : 4 points
 - chemin de longueur 4 : 7 points
 - chemin de longueur 6 : 15 points
 - chemin de longueur 8 : 21 points
- En réalisant le contrat d'une carte destination en sa possession
- En construisant le chemin continu le plus long (10 points supplémentaires)

A chaque tour, plusieurs possibilités s'offrent à un joueur. Il peut :

- Choisir deux cartes Wagon ou une carte Locomotive parmi 5 cartes visibles ou encore piocher deux cartes de la pioche (face cachée).
- Prendre possession d'une route en défaussant des cartes wagons :
 - Certains chemins imposent un nombre de wagons d'une couleur spécifique. Dans ce cas il faut défausser autant de cartes wagons de la couleur donnée.
 - D'autres chemins laissent le choix de la couleur aux joueurs (ils sont représentés par la couleur grise). Le joueur cependant utiliser une seule couleur pour toutes les cartes wagons.
- Piocher de nouvelles cartes Destination

NB : Les contrats de cartes Destination non réalisés pénalisent le joueur d'autant de points qu'indiqués sur la carte.

Lorsqu'un joueur ne possède plus que deux wagons, un dernier tour est joué (incluant le dit joueur) puis le jeu s'arrête ; les points de chaque joueur sont alors comptabilisés.



2 Contexte du problème

2.1 Problématique

La principale difficulté pour les joueurs est d'effectuer les meilleurs choix stratégiques en fonctions des possibilités s'offrant à eux et des divers paramètres lors de leur tour de jeu. Le but de cette TX est de fournir un outil d'aide à la décision basé sur un calcul de chemins qui donnera au joueur la meilleure option se présentant à lui à chaque tour.

2.2 Paramètres du problème

De multiples paramètres sont à prendre en compte lors du choix de l'action du joueur. Les paramètres concernant le joueur lui-même :

- Nombre et couleur des cartes en main
- Nombre de wagons restants
- Destinations à réaliser
- Chemins déjà acquis

Les paramètres concernant les autres joueurs :

- Nombre de cartes en main
- Nombre de wagons restants
- Chemins déjà acquis

Les paramètres de la partie :

- Nombre de cartes de chaque couleur ayant déjà été jouées
- Cartes présentes dans la pioche

Ainsi, pour estimer le meilleur chemin possible entre deux villes, nous devons non seulement prendre en compte les cartes wagons que le joueur possède en main, les chemins dont il a la maîtrise sur le plateau de jeu et le nombre de pions wagons qu'il lui reste (il ne peut en jouer plus qu'il en a). Mais aussi les possessions de ses adversaires (notamment les chemins possédés sur le plateau de jeu), ainsi que les cartes susceptibles d'être piochées (en se basant sur les 5 cartes disponibles visibles). On pourra également prendre en compte les cartes wagons ayant déjà été utilisées pour construire des chemins sur le plateau de jeu.



3 Problème 1 : Définir le meilleur chemin

3.1 Déterminer le meilleur chemin entre deux destinations.

L'une des premières questions à se poser est celle de la comparaison des différents chemins : comment discriminer les chemins entre eux. En effet, un trajet (ou chemin) possède un coût, exprimé en cartes wagons, mais aussi en pions, et nécessite un nombre conséquent de tours pour le réaliser. Pour compenser cela, il rapporte des points (appelé gain), en plus des points que rapporte le fait d'avoir accompli la destination. Ce gain dépend directement de la longueur de chaque tronçon composant le trajet complet (voir détail des points au début de la section [1.3 But du Jeu](#)).

On ne peut pas simplement choisir le chemin qui rapporte le plus de points ou qui coûte le moins de wagons à construire, car cela n'est peut être pas stratégiquement optimal. Il faut essayer de trouver un compromis entre le coût et le gain associé. C'est donc ce que nous avons commencé par rechercher.

3.1.1 Calcul de la rentabilité des chemins grâce au rapport $\frac{\text{coût}}{\text{gain}}$

Pour calculer le coût d'un chemin il faut simplement additionner la longueur de chacun de ses tronçons. Le calcul du gain nécessite d'additionner le gain associé à chaque tronçon.

En effet, prenons deux chemins de longueur 4. Le premier est composé de deux tronçons de longueur 2. Le second est formé d'un tronçon de longueur 3 et d'un autre de longueur 1. Les deux chemins ont donc le même coût : 4. En revanche, le premier rapporte $2+2=4$, tandis que l'autre rapporte $4+1=5$ (*un tronçon de longueur 1 rapporte 1, un tronçon de longueur 2 rapporte 2 et un tronçon de longueur 3 rapporte 4*). Avec ces deux paramètres, nous pouvons calculer un ratio $\frac{\text{coût}}{\text{gain}}$ permettant de qualifier la rentabilité des chemins. Le chemin ayant le rapport le plus faible sera considéré comme le plus rentable et donc le meilleur choix pour un joueur.

Par exemple, imaginons plusieurs chemins pour une même destination quelconque :

| N° | Longueur des tronçons | coût du chemin | gain du chemin | rapport $\frac{\text{coût}}{\text{gain}}$ |
|----|-----------------------|----------------|--------------------|---|
| 1 | 4+3+2 | 9 | $7+4+2 = 13$ | 0.69 |
| 2 | 3+3+2+2 | 10 | $4*2+2*2 = 12$ | 0.83 |
| 3 | 4+3+2+1 | 10 | $7+4+2+1 = 14$ | 0.71 |
| 4 | 4+4+4+3+3+3+2+2 | 25 | $7*3+4*3+2*2 = 37$ | 0.68 |

On note donc que le chemin le plus rentable ici pour relier nos deux destinations est le N°4 car il possède le rapport $\frac{\text{coût}}{\text{gain}}$ le plus faible (0.68).

Problème : Des chemins extrêmement longs peuvent obtenir un ratio plus avantageux que des chemins courts. Le chemin le plus avantageux coûte ici 25 wagons, ce qui est considérable. Ce chemin n'est pourtant pas si éloigné du chemin N°1, qui lui possède un rapport de 0.69. Même si le rapport gain par rapport au coût est parfois meilleur sur un chemin très long, le joueur ne souhaitera sans doute pas faire le tour du plateau pour rejoindre ses destinations. En effet, sans compter le temps perdu, il risquerait de se faire barrer la route par un adversaire. Ce risque augmente avec le nombre de tronçons composant le chemin, dépendant de la longueur des tronçons le composant. De plus, plus un tronçon est court plus il est facile à construire. Si jamais un tronçon est "capturé", le joueur devra faire un détour, et aura donc dévié du chemin jugé originaleme meilleur.



3.1.2 Introduire la notion de risque dans le calcul du rapport

Compte tenu du problème énoncé ci-dessus, nous nous proposons d'introduire la notion de risque pour discriminer les chemins selon la longueur de leurs tronçons. Cela permettra de prendre en compte le risque qu'un tronçon soit capturé par d'autres joueurs et que le chemin soit ainsi bloqué.

Nous avons effectué plusieurs essais pour introduire la notion de risque dans le calcul du rapport et avons finalement décidé de le représenter par un coefficient multiplicateur. Ainsi, le ratio R_c d'un chemin c donné, sera calculé de la façon suivante :

$$R_c = \frac{\text{coût}}{\text{gain}} * \text{risque}$$

Dans tous les cas nous calculons le risque en fonction de la longueur n de chaque tronçon qui compose le chemin. C'est à dire que nous associons à une longueur de tronçon donnée, un facteur de risque qui en dépend. Le risque total est donc la somme des risques associés à chacun des tronçons.

* * *

Dans un premier temps, nous avons choisi pour valeur de risque $r = \frac{1}{n}$. A chaque tronçon nous associons l'inverse de son coût comme valeur de risque.

Nous nous sommes vite rendu compte que les tronçons courts étaient trop désavantagés par rapport aux longs. Reprenons les chemins N°2 et N°3 présentés précédemment.

| N° | Longueur des tronçons | coût du chemin | gain du chemin | rapport $\frac{\text{coût}}{\text{gain}}$ |
|----------|-----------------------|----------------|----------------|---|
| 2 | 3+3+2+2 | 10 | $4*2+2*2 = 12$ | 0.83 |
| 3 | 4+3+2+1 | 10 | $7+4+2+1 = 14$ | 0.71 |

A priori le chemin N°3 est le plus intéressant des deux car il offre un meilleur gain que le N°2, pour le même coût total. Calculons le risque associé à chaque chemins, en utilisant $r = \frac{1}{n}$ pour le risque associé à chaque tronçon des chemins.

| Longueur du tronçon n | risque associé $r = \frac{1}{n}$ |
|-------------------------|----------------------------------|
| 1 | $\frac{1}{1} = 1$ |
| 2 | $\frac{1}{2} = 0.5$ |
| 3 | $\frac{1}{3} \approx 0.33$ |
| 4 | $\frac{1}{4} = 0.25$ |

L'écart entre ces différentes valeurs selon la longueur des chemins semble importante, mais calculons tout de même les ratios R_c des chemins N°2 et N°3.

| N° | tronçons | coût | gain | risque | $R_c = \frac{\text{coût}}{\text{gain}} * \text{risque}$ |
|----------|----------|------|------|------------------------|---|
| 2 | 3+3+2+2 | 10 | 12 | $0.33*2+0.5*2=1.66$ | $0.83*1.66=1.38$ |
| 3 | 4+3+2+1 | 10 | 14 | $0.25+0.33+0.5+1=2.08$ | $0.71*2.08=1.48$ |

Avec le critère de risque $r = \frac{1}{n}$, le chemin N°2 est donc désormais considéré comme meilleur que le chemin N°3. De manière générale, si le chemin est composé de tronçons de longueur 1 ou 2, il est beaucoup trop discriminé par rapport aux chemins composés de tronçons un peu plus longs. Le but est bien de discriminer les chemins composés de tronçons courts, ainsi que les longs chemins, toutefois ici le facteur de discrimination est beaucoup trop important.

* * *



Nous avons donc ensuite essayé avec $r = \frac{1}{\sqrt{n}}$. Les valeurs de risque associées aux longueurs de tronçons sont donc les suivantes :

| Longueur du tronçon n | risque associé $r = \frac{1}{\sqrt{n}}$ |
|------------------------------|---|
| 1 | $\frac{1}{\sqrt{1}} = 1$ |
| 2 | $\frac{1}{\sqrt{2}} \approx 0.71$ |
| 3 | $\frac{1}{\sqrt{3}} \approx 0.58$ |
| 4 | $\frac{1}{\sqrt{4}} = 0.5$ |
| 6 | $\frac{1}{\sqrt{6}} \approx 0.41$ |
| 8 | $\frac{1}{\sqrt{8}} \approx 0.35$ |

La racine carrée permet de réduire les écarts trop importants entre les coefficients de risque pour les différentes longueurs de tronçons. Ainsi, cela limite les valeurs abhérentes obtenues lorsque le chemin contient un tronçon très petit. Recalculons les ratios R_c des chemins N°2 et N°3.

| N° | tronçons | coût | gain | risque | $R_c = \frac{\text{coût}}{\text{gain}} * \text{risque}$ |
|-----------|----------|------|------|------------------------|---|
| 2 | 3+3+2+2 | 10 | 12 | $0.58*2+0.71*2=2.58$ | $0.83*2.58=2.14$ |
| 3 | 4+3+2+1 | 10 | 14 | $0.5+0.58+0.71+1=2.79$ | $0.71*2.79=1.98$ |

On note que le risque associé au chemin N°3 est supérieur que celui associé aux chemin N°2. En effet, cela est logique puisqu'il contient un tronçon de longueur 1, qui est susceptible d'être capturé très facilement par un autre joueur. Cependant, étant donné que le chemin N°3 possède un meilleur rapport $\frac{\text{coût}}{\text{gain}}$ que le chemin N°2, son R_c reste le plus intéressant.

Nous adoptons donc $r = \frac{1}{\sqrt{n}}$ comme étant le risque associé à chaque tronçon, et le risque d'un chemin comme étant la somme des risques associés à chacun de ses tronçons.

3.1.3 Interprétation du risque

Un nombre n'étant pas très parlant pour un utilisateur, l'idée est de placer le risque d'un chemin sur une échelle pour donner une interprétation plus concrète de ce risque.

Nous avons choisi d'utiliser 5 plages différentes de risque, comme valeurs pour ces plages nous prenons les risques associés à chaque longueur de tronçons (arrondies à quelques unités près), que nous multiplions par 100. Voici l'échelle de niveau de risque :

- Minime : [0, 36[
- Faible : [36, 50[
- Modéré : [50, 60[
- Elevé : [60, 72[
- Critique : [72, $+\infty$ [

Pour étailler le risque total d'un chemin sur cette échelle, nous divisons le risque du chemin par le nombre de tronçons qu'il utilise, et nous multiplions le tout par 100 pour positioner le risque sur notre échelle.



3.1.4 Bilan sur le calcul du meilleur chemin entre deux destinations

Afin de déterminer le chemin le plus optimal pour le joueur, nous associons une valeur à chaque chemin permettant de relier deux destinations. Cette valeur, notée R_c , est le rapport entre le coût du chemin et le gain qui lui est associé, auquel nous multiplions le risque associé à la prise du chemin dans son intégralité.

$$R_c = \frac{\text{coût}}{\text{gain}} * \text{risque}$$

Nous avons choisi ce rapport en faisant en sorte qu'il mette en valeur les chemins relativement courts (composés de peu de tronçons) et qui possèdent un gain important par rapport à leur coût.

Nous définissons comme meilleur chemin le chemin reliant deux destinations avec le ratio R_c minimal.



3.2 Algorithme : meilleur chemin reliant deux destinations

Nous avons commencé par rédiger un algorithme permettant de lister tous les chemins pour relier deux destinations données.

3.2.1 Données de l'algorithme

Nous considérons les éléments suivants comme connus :

- Villes voisines (qui possèdent un tronçon les reliant entre elles).
- Coût (longueur) des tronçons reliant deux villes voisines.
- Nombre de points associés à chaque tronçon (en fonction de sa longueur).
- Risque associé à un tronçon.

Cela est représenté dans l'algorithme de la façon suivante :

Soit **A** une ville.

A.voisins renvoie la liste des voisins **v** de **A**.

Soit **A** et **B** deux villes voisines.

coût(A, B) renvoie le coût du tronçon entre **A** et **B**.

gain(A, B) renvoie le gain du tronçon entre **A** et **B**.

risque(A, B) renvoie le risque du tronçon entre **A** et **B**.

FIGURE 5 – Données de l'algorithme 1

Nous définissons ensuite une structure chemin qui permettra de contenir les différentes informations relatives aux chemins. Ces informations sont les suivantes :

- La liste des villes contenues dans le chemin
- Le coût global du chemin
- Le gain global associé au chemin
- Le risque global associé au chemin

Cela est représenté dans l'algorithme de la façon suivante :

Soit **c** un chemin.

c.villesVisitées donne la liste des villes composant le chemin.

c.coût donne le coût associé au chemin

c.gain donne le gain associé au chemin

c.risque donne le risque associé au chemin

FIGURE 6 – Structure de Chemin dans l'algorithme 1



La procédure suivante permet donc de remplir une liste de tous les chemins possibles reliant deux destinations A et B.

```
Chemin[] listeChemins ;  
  
trouverChemins(Ville A, Ville B, Ville[] visitées, int coût, int gain, float risque) :  
    ajouter A à visitées ;  
    Si B = A :  
        Chemin c ;  
        c.villesVisitées = visitées ;  
        c.coût = coût ;  
        c.gain = gain ;  
        c.risque = risque ;  
        ajouter c à listeChemins ;  
    Sinon :  
        Pour chaque Ville v de A.voisins :  
            Si visitées ne contient pas v :  
                trouverChemins(v, B, visitées, coût + coût(A, v), gain + gain(A, v), risque + risque(A, v)) ;
```

FIGURE 7 – Algorithme 1

3.2.2 Explications

La procédure *trouverChemins* est une procédure récursive. Cet algorithme s'appuie sur un algorithme d'exploration de tous les noeuds d'un graphe non-orienté.

En effet, nous pouvons aisément comparer le plateau de jeu à un graphe non-orienté, dont villes en représentent les noeuds, et les tronçons sont les arrêtes du graphe.

Les arguments de cette procédure sont donc les deux destinations (notées ville A et ville B), la liste des villes déjà explorées (ou visitées) ainsi que le coût, le gain et le risque actuel du chemin parcouru.

À noter : pour le premier appel de la procédure, les paramètres coût, gain et risque sont à 0, et la liste des villes visitées est vide. Seules les villes A et B sont donc renseignées.

Première partie : bloc "Si"

Nous souhaitons aller de la ville A à la ville B. La condition d'arrêt de la récursion est donc relativement évidente : A = B. Dans ce cas, nous avons trouvé un nouveau chemin entre les deux villes. Il faut donc l'ajouter à la liste des chemins de A à B (ici appellée listeChemins). Pour chaque chemin, nous spécifions les informations décrites ci-dessus, à savoir : la liste des villes du chemin, son coût, son gain et son risque associé.

Seconde partie : bloc "Sinon"

Si A est différent de B, nous n'avons pas encore trouvé de chemin entre les deux villes : nous continuons donc l'exploration. Pour chaque voisin v de A qui n'est pas déjà dans la liste des villes visitées (on ne veut pas de boucles) on appelle de nouveau la procédure *trouverChemins*, non plus entre A et B, mais entre v et B. On incrémente les valeurs du coût, du gain et du risque selon la valeur correspondante entre A et v dans les paramètres de la fonction.



3.2.3 Complexité de l'algorithme

Posons M le nombre d'arrêtes du graphe (nombre de tronçons) et N le nombre de noeuds (les villes). L'algorithme cherche à lister toutes les possibilités pour relier un noeud A à un noeud B.

- La première partie de l'algorithme (jusqu'au bloc Sinon) se réalise en temps constant $O(1)$, en effet il ne s'agit ici uniquement que d'attribution de valeurs, définitions de variables et ajout en fin de liste (on suppose que l'ajout d'un élément en fin de liste se fait en temps constant et ne dépend pas du nombre d'éléments dans la liste).
- La partie nous intéressant le plus est le bloc Sinon. La boucle **Pour** est réalisée dans le pire des cas N fois. En effet, un noeud possèdera au pire tous les autres noeuds comme voisins. C'est donc une complexité en $O(N)$.
L'appel récursif lui dépend du nombre d'arrêtes. Dans le pire des cas, on parcourra toutes les arrêtes avant de relier le noeud A au noeud B. On appellera donc récursivement la procédure M fois avant de rejoindre le noeud B. On a donc une complexité en $O(M)$.

En conclusion, la complexité globale de la procédure dépend donc de celle du bloc Sinon, qui est donc en $O(N^M)$.

Cette complexité est considérable : exécuter cet algorithme prendrait un temps considérable. En effet, si nous prenons l'exemple du plateau de la version Europe du jeu, il y a ($M =$) 90 tronçons et ($N =$) 48 villes.

Dans la pratique, une ville possède au plus 7 voisins (sur la version Europe, on peut concevoir qu'il n'y aura pas plus d'une dizaine de voisins d'une ville dans une version quelconque du jeu). De plus, le graphe n'est pas représentable sous forme d'arbre dégénéré (arbre composé d'une seule branche) : il n'est donc jamais nécessaire de parcourir tous les tronçons avant de trouver la ville de destination : **cela limite donc grandement la complexité dans la pratique**.

3.2.4 Exploitation de la liste des chemins

Une fois la procédure terminée, la liste des chemins entre A et B (`listeChemins`) est désormais remplie. Il ne reste plus qu'à calculer les \mathbf{R}_c de chaque chemin dans cette liste ; le chemin qui aura le plus petit ratio \mathbf{R}_c pourra être considéré comme étant le meilleur chemin.

```
float Rc_mini = +∞ ;
Chemin c_mini ;

Pour chaque Chemin c de listeChemins :
  float Rc = c.coût * c.risque / c.gain ;
  Si Rc < Rc_mini :
    | Rc_mini = Rc ;
    | c_mini = c ;
```

FIGURE 8 – Exploitation de la liste des chemins pour trouver le meilleur.

La boucle ci-dessus effectue le calcul du \mathbf{R}_c pour chaque chemin trouvé à l'aide de la procédure `trouverChemins`, et permet de retenir le chemin qui possède le plus petit ratio.

La complexité ici dépend de la taille (N) de la liste des chemins. En effet nous parcourons l'ensemble de la liste pour trouver l'élément possédant le meilleur ratio. Toutes les opérations



sont effectuée en temps constant : $O(1)$, et la boucle répétée N fois. La complexité de cette partie est donc en $O(N)$.

3.3 Prise en compte des possessions des joueurs

Dans cette partie, nous souhaitons relier deux destinations en utilisant des tronçons déjà acquis par le joueur ou en excluant les chemins des joueurs adverses. Il est donc nécessaire de pouvoir connaître l'état de chaque tronçon à tout moment de la partie.

Pour indiquer qu'un chemin est bel et bien en possession de ce joueur, nous proposons d'attribuer à un tronçon des valeurs de gain, de coût et de risque égales à 0 dès lors que le joueur en prend possession. A contrario, pour signifier qu'un joueur adverse possède un tronçon, le tronçon est supprimé afin qu'il ne soit plus considéré du tout. Le calcul du meilleur chemin s'effectue donc exactement de la même manière que précédemment.

Nous supposons qu'en général, la stratégie la plus intéressante serait d'utiliser un maximum de tronçons déjà construits afin de relier les destinations avec un coût minimal car cela représente un gain de temps et de wagons qui pourront être utiles à l'achèvement de nouvelles destinations. Or, si nous affichons seulement le chemin ayant le R_c minimal, l'algorithme nous donne parfois des chemins rentables en points mais peut-être moins optimaux dans certains cas (si un joueur adverse ne possède plus beaucoup de wagons par exemple, alors finir ses destinations le plus rapidement possible devient une priorité, au risque de perdre des points). Il serait peut être alors pertinent d'afficher à la fois le chemin le plus rentable, ainsi que le chemin le plus court pour permettre au joueur de s'adapter à chaque situation.

3.4 Prise en compte des tronçons doubles

Comme nous en avons déjà parlé plus haut, il existe des tronçons dédoublés constitués de deux passages de couleur différente. Lorsque le nombre de joueurs est supérieur ou égal à 4, ces tronçons peuvent être possédés par deux joueurs différents et permettent ainsi d'éviter que les joueurs soient bloqués trop rapidement.

Lorsque les tronçons doubles ne sont pas actifs, le joueur voulant prendre possession de l'un d'eux choisit la couleur qu'il préfère pour construire ses wagons. Puis une fois que la route est occupée, elle est considérée comme supprimée pour tous les autres joueurs. A l'inverse, lorsque les chemins doubles sont actifs, lorsqu'un premier joueur a fait l'acquisition d'un des deux tronçons, la route est encore active pour les autres joueurs mais ceux-ci n'ont plus le choix de la couleur à utiliser.

Nous avons souhaité pouvoir les prendre en compte dans notre algorithme. Pour cela, il faudra d'abord indiquer le nombre de joueur. Pendant la partie, si un joueur prend possession de l'un des deux passages constituant un tronçon double, ce passage sera supprimé pour les autres joueurs mais la seconde possibilité apparaîtra toujours.

De plus, dans le cas où les deux passages sur un tronçon sont actifs, le risque associé à ce tronçon sera divisé par 2. Si l'un des deux passages est occupé, le risque toutefois redouble.



3.5 Prise en compte des couleurs

Une majeure partie du jeu se base sur la gestion des couleurs que le joueur a en main. Quelle couleur doit il garder ou utiliser ? Quelle couleur doit il piocher ? Quelles couleurs les autres joueurs ont en main ou bien encore quelles couleurs ont déjà été utilisées ?

Autant de questions qu'un joueur peut se poser lors d'une partie. Nous avons donc réfléchi à un moyen d'intégrer ces couleurs à notre algorithme. Pour cela, il faudrait se baser sur un calcul de probabilités, connaissant la proportion de chaque couleur et des locomotives, les cartes ayant été utilisées, l'état de la pioche,etc...

L'estimation de ces paramètres paraît compliquée. En effet, cela requerra des joueurs qu'ils entrent systématiquement un nombre conséquent d'informations qui serait très lourd. De plus en considérant que la pioche est renouvelable, les probabilités deviennent très complexes à calculer.

Nous avons donc décidé de ne pas prendre en compte les couleurs dans notre calcul mais de seulement indiquer aux joueurs de quelles couleurs ils auront besoin pour construire leurs routes.

3.6 Meilleur chemin avec étapes

Rarement dans le jeu, un joueur réalise des chemins les uns après les autres indépendamment. Nous souhaitons donc ici proposer une solution pour relier deux villes en passant par une ou plusieurs autres, afin de compléter plusieurs trajets en même temps.

3.6.1 Méthode avec combinaisons de chemins intermédiaires

Nous avons imaginé une première méthode, qui va chercher à combiner des chemins afin d'en obtenir un qui passe par toutes nos villes.

Prenons un exemple :

Soient 3 villes que nous nommerons A, B et C. L'objectif est de relier A à C en passant par la ville B.

Pour connaître la meilleure combinaison de tronçons, nous pouvons réutiliser le même algorithme que précédemment en l'appliquant aux 3 chemins possibles (A-B, A-C et B-C). Nous pourrons déterminer à l'aide des listes de chemins ainsi générées le chemin le plus optimisé et le chemin le plus court pour chacun de ces tronçons. Afin de déterminer la meilleure combinaison pour réaliser notre objectif, nous allons devoir parcourir toutes les permutations de ces chemins.

- A,B,C
- A,C,B
- B,A,C
- B,C,A
- C,A,B
- C,B,A

Elle sont au nombre de 6, ce qui correspond aux permutations des 3 villes ($6 = 3!$).



Cependant, nous remarquons que comme l'ordre n'importe pas (par exemple : A,B,C \iff C,B,A), nous pouvons éliminer la moitié des permutations à tester. Dans notre cas il nous restera donc que $\frac{3!}{2}$ possibilités, c'est à dire 3 permutations, qui sont les suivantes :

- A,B,C (équivaut à C,B,A)
- A,C,B (équivaut à B,C,A)
- B,A,C (équivaut à C,A,B)

Plus généralement :

Soient A ville de départ, B ville d'arrivée et N villes que nous appelerons villes étapes. Nous souhaitons relier A à B en passant par les N villes étapes.

Nous commencerons par trouver tous les chemins entre ces villes à l'aide de notre procédure récursive présentée précédemment (cela représentera donc $\frac{X!}{2}$ listes de chemins générées, avec X le nombre de villes). Pour chacune de ces listes, nous allons déterminer le meilleur chemin selon notre critère du R_c , ainsi que le plus court chemin afin de permettre au joueur de choisir en fonction de sa situation.

Une fois cette étape faite, pour chaque permutation possible (sans prendre en compte l'ordre des villes, nous avons donc $\frac{N!}{2}$ permutations pour N villes), nous allons additionner les meilleurs chemins précédents permettant de relier A à B en passant par les N villes étapes, ce qui nous donnera un chemin composé de tous les meilleurs chemins entre ces étapes. de cette manière, nous pourrons déterminer lequel parmi toutes ces permutations est le meilleur (et déterminer également le plus court).

3.6.2 Méthode avec la liste de tous les chemins

Une seconde méthode - beaucoup plus simple, mais aussi plus limitée - est envisageable. Comme la procédure *trouverChemins(...)* nous renvoie la liste de tous les chemins entre deux destinations, parmi ces chemins il en existe forcément qui passent par les villes étapes précisées. Ainsi, il suffit de prendre parmi ces chemins le plus optimisé selon le critère R_c et le plus court.

Le désavantage principal avec cette méthode est que la ville de départ et la ville d'arrivée sont fixes. C'est à dire que quoi qu'il arrive, notre chemin commencera par la ville de départ et s'arrêtera à la ville d'arrivée. De cette manière nous devrons probablement faire de gros détours pour passer par les villes étapes si elles sont relativement éloignées du chemin qui relie les deux villes cibles.



4 Implémentation en Java

Tout ce qui a été défini auparavant nous a permis de concevoir efficacement et rapidement une implémentation concrète de cette étude.

En termes techniques, nous utilisons la version 1.8 de Java.

4.1 Format de donnée

Nous avons commencé par définir le format de données du plateau de jeu. Ceci est la base sur laquelle s'appuie tout le logiciel puisqu'il s'agit des données d'entrées. Nous avons choisi de le représenter dans un fichier au format XML. C'est un format tout à fait adapté à notre problématique, qui a également l'avantage d'être facilement exploitable en Java.

Nous représentons dans ce fichier l'ensemble des tronçons du plateau de jeu. Chaque tronçon est défini par :

- Ses extrémités (les deux villes qu'il relie)
- Son coût (nombre de wagons nécessaire à sa construction)
- Sa couleur (ou 2 couleurs dans le cas d'un chemin double)

Plus concrètement, voici donc à quoi ressemble un tronçon dans ce fichier :

```
<route>
  <ville1>Cadiz</ville1>
  <ville2>Lisboa</ville2>
  <cout>2</cout>
  <couleur>B</couleur>
</route>
```

FIGURE 9 – Représentation XML du tronçon Cadiz-Lisboa

On y retrouve donc les deux villes reliées (par convention, `<ville1>` contiendra la première des deux villes dans l'ordre alphabétique), le coût du tronçon ainsi que sa couleur.

Pour les couleurs, nous les représentons par une initiale, qui symbolise leur nom en anglais. Par conséquent nous obtenons :

- R : Red (rouge)
- P : Pink (rose)
- G : Green (vert)
- B : Blue (bleu)
- D : Dark (sombre, nous avons volontairement pas choisi Black car l'initiale aurait été identique à celle de Blue)
- W : White (blanc)



- Y : Yellow (jaune)
- O : Orange (orange)
- N : Null (couleur neutre, il s'agit des tronçons gris, qui laissent le choix de la couleur au joueur)

Dans le cas des tronçons doubles, par exemple Edinburgh-London, nous représentons cela par un doublon de couleur (ici 'D,O' pour 'Dark, Orange') :



```
<route>
  <ville1>Edinburgh</ville1>
  <ville2>London</ville2>
  <cout>4</cout>
  <couleur>D,O</couleur>
</route>
```

FIGURE 11 – Représentation XML

FIGURE 10 – Tronçon Edinburgh-London

4.2 Conception des objets principaux

Les objets principaux à manipuler dans le logiciel sont les chemins et les tronçons. Nous avons réalisé deux classes UML de ces objets afin de mettre au clair ce qu'ils doivent contenir en terme d'attributs et de méthodes.

Nous avons décidé de représenter les villes sous forme de chaînes de caractères (String). Voici le diagramme UML de la classe Route, qui symbolise un tronçon.

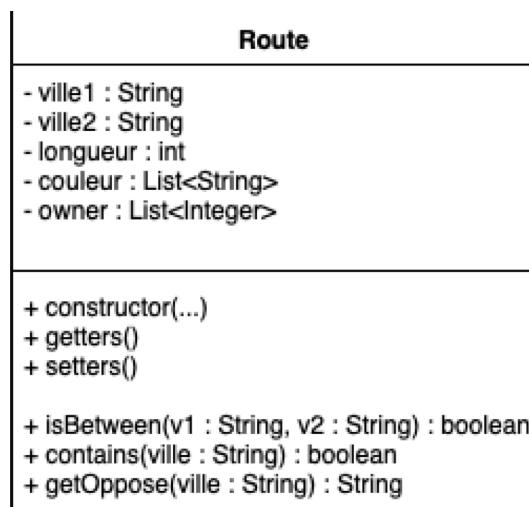


FIGURE 12 – Diagramme UML de Route (tronçon)

On y retrouve :



- Tous les attributs décrivant un tronçon (ou route) entre deux villes : les noms des deux villes, son coût, la liste de ses couleurs (il y en a 2 dans le cas d'un chemin double) et la liste des joueurs qui la possèdent (attribut "owner", liste vide au début, qui possède éventuellement deux valeurs dans le cas d'un chemin double).
- Les méthodes classiques : *setters* et *getters* qui permettent de modifier et récupérer les valeurs des attributs. Le constructeur qui permet de créer l'objet.
- Quelques méthodes de services utiles dans les différentes vérifications et processus de traitement des chemins.
 - Une méthode *isBetween(...)* qui prend en argument deux villes et qui permet de vérifier que la route est bien entre ces deux villes (sans prendre en compte l'ordre).
 - Une méthode *contains(...)* qui prend en argument une ville et qui renvoie vérifie si la ville est bien dans la route (une des deux extrémités).
 - Une méthode *getOppose(...)* qui à partir d'une ville (supposée dans la route) renvoie l'autre extrémité.

L'objet représentant un chemin est construit de manière similaire :

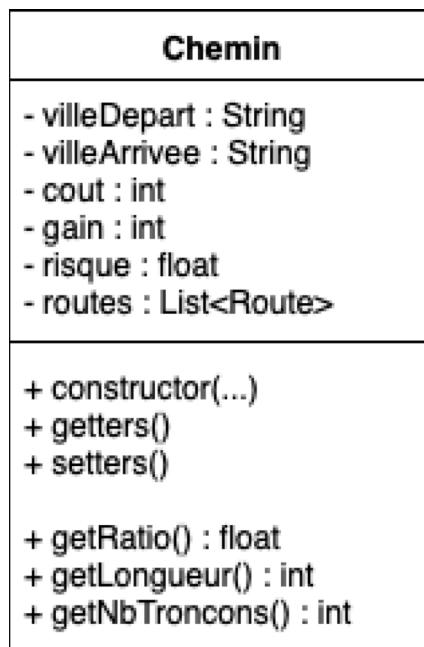


FIGURE 13 – Diagramme UML de Chemin

On y retrouve également :

- Tous les attributs décrivant un chemin entre deux villes : les noms des deux villes (de départ et d'arrivée), son coût total, gain total, risque total et la liste des routes qu'il emprunte.
- Les mêmes types de méthodes classiques : *setters*, *getters* et le constructeur.
- Des méthodes de services.
 - Une méthode *getRatio()* qui va calculer et renvoyer le ratio $R_c = \frac{\text{cout}}{\text{gain}} * \text{risque}$ à partir des valeurs de coût, risque et gain du chemin.



- Une méthode `getLongueur()` qui renvoie le nombre total de wagons nécessaire à la construction du chemin.
- Une méthode `getNbTroncons()` qui renvoie le nombre de tronçons (de routes) utilisées dans le chemin.

4.3 Choix d'implémentation

Certains choix ont été effectués afin d'aller plus vite dans l'implémentation logicielle et d'avoir un outil fonctionnel avant la fin du semestre. Avec plus de temps, nous aurions pu pousser plus loin nos recherches, et pris le temps de tout implémenter dans le programme.

Algorithme principal

Comme mentionné plus haut, la complexité de la procédure `trouverChemins(...)` est très élevée. Ainsi, l'implémentation telle quelle de la procédure donne un temps de calcul bien trop conséquent. Nous avons donc choisi de garder l'algorithme, mais de se limiter en terme de tronçons empruntés par un chemin. De toutes manières un joueur ne sera pas intéressé par des chemins d'un nombre important de tronçons. Pour le plateau de jeu des aventuriers du rail, nous avons fixé à 13 la limite de tronçons pour un chemin. Cela nous permet ainsi d'avoir un résultat dans un temps de calcul raisonnable.

Implémentation des chemins avec étapes

Afin d'utiliser la première méthode de traitement de chemins avec étapes (avec les combinaisons de chemins) il est nécessaire de manipuler un certain nombre de listes de chemins, et de les traiter. Les temps nécessaires à la génération de ces listes de chemins sont conséquents, et plus on rajoute d'étapes plus le temps d'attente avant l'obtention d'un résultat est important. En effet, pour une étape, on effectue $\frac{3!}{2} = 3$ appels à notre procédure `trouverChemins(...)`, pour deux étapes on a $\frac{4!}{2} = 12$ appels, trois étapes $\frac{5!}{2} = 60$ et ainsi de suite.

Étant donné la complexité de la procédure ($O(N^M)$ en théorie, en pratique $N \geq 7$ et $M \geq 13$ pour le plateau de l'Europe limite la complexité, mais on a tout de même en temps d'attente de quelques secondes pour avoir un résultat), l'implémentation concrète de cette méthode nécessite d'optimiser grandement la procédure pour que le joueur n'ait pas à attendre plusieurs minutes.

C'est pourquoi nous avons choisi ici la solution la plus simple (la seconde méthode), qui utilise directement la liste des chemins générées pour relier deux villes, en gardant dans cette liste uniquement les chemins qui passent par les étapes. Les chemins étant limités à 13 tronçons empruntés (pour le plateau Europe) il se peut qu'aucun chemin ne soit trouvé qui passe par toutes les étapes voulues.

En pratique, il n'est de toutes manières pas pertinent d'avoir des chemins trop longs. Si l'utilisateur met des destinations éloignées avec des étapes tout autant éloignées, le meilleur chemin sera extrêmement long et donc peu pertinent car irréalisable (d'autres joueurs auront "capturé" certains des tronçons du chemin).

Prise en compte des couleurs

La prise en compte des couleurs nécessiterait que l'utilisateur rentre en continu ce que chaque joueur pioche (quand il s'agit d'une des 5 cartes de la pioche visible), ce qu'il a dans la main, ce qui a été joué... C'est un travail fastidieux, qui prendrait un temps conséquent au joueur.



Il faudrait réfléchir à l'ergonomie que prendrait l'intégration d'une telle fonctionnalité afin de ne pas enlever au joueur l'intérêt principal du jeu : s'amuser. Le but de cet outil étant l'aide à la décision il ne faut pas que l'utilisateur passe la majeure partie de son temps sur l'outil et pas sur la partie en cours.

Les couleurs ne sont donc pas prises en compte de le processus de détermination des chemins les plus optimisés.

4.4 Interface graphique

Nous utilisons pour l'interface graphique la librairie Swing, qui est la librairie de base pour les interface graphiques en Java.

Voici ce à quoi ressemble l'interface graphique de notre programme :

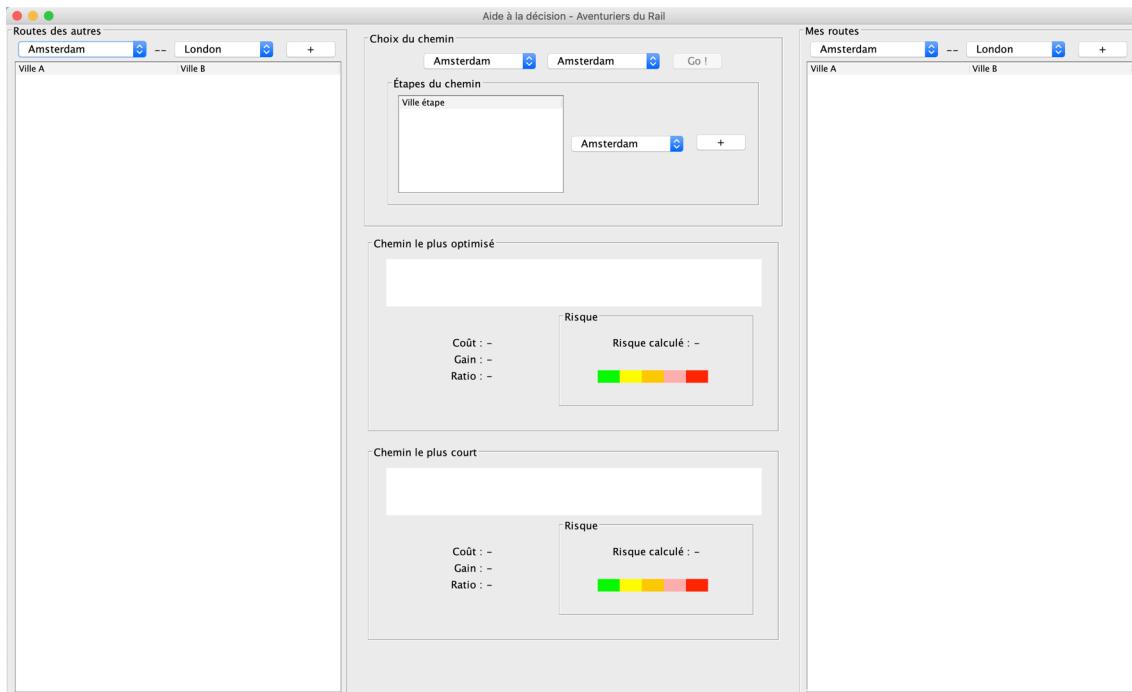


FIGURE 14 – Interface de l'application

Nous l'avons divisée en 3 panneaux principaux :

- Les deux panneaux latéraux : possessions des joueurs.
- Le panneau central : choix et résultats du chemin sélectionné.



4.4.1 Les panneaux latéraux

Les deux panneaux latéraux sont là pour indiquer les possessions des joueurs. Celui de gauche indique les possessions des joueurs ennemis au joueur utilisant l'application, et celui de droite les possessions du dit joueur.

Les deux panneaux sont identiques, voici donc comment les utiliser.

On retrouve en haut du panneau deux boîtes déroulante et un bouton d'ajout. La première boîte contient toutes les villes disponibles sur le plateau (triées alphabétiquement). La seconde contient une uniquement les villes voisines à la première boîte déroulante.

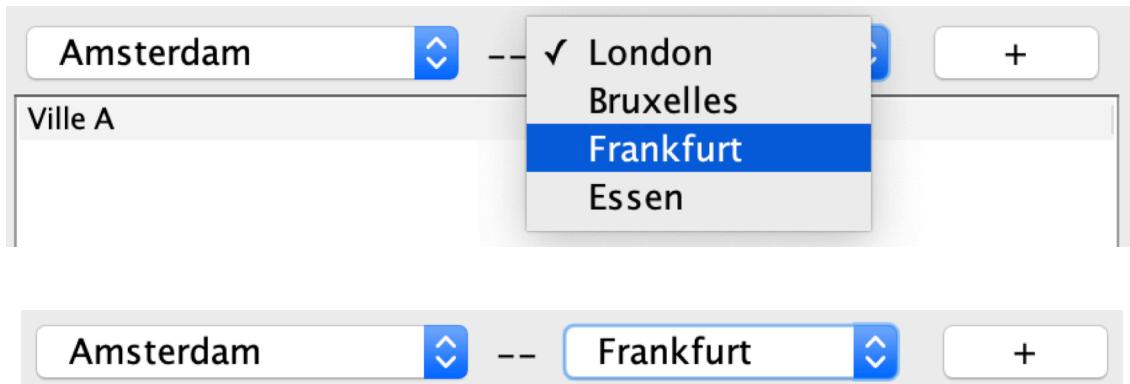


FIGURE 15 – Ajout d'une nouvelle possession

Une fois le choix de la route effectué, un clic sur le bouton d'ajout (+) permet de l'ajouter à la liste d'en-dessous. S'il s'agit d'un tronçon simple (certains tronçons sont doubles) la route ne sera plus disponible dans les boîtes déroulantes de sélection des possessions des joueurs (utilisateur et ses ennemis).

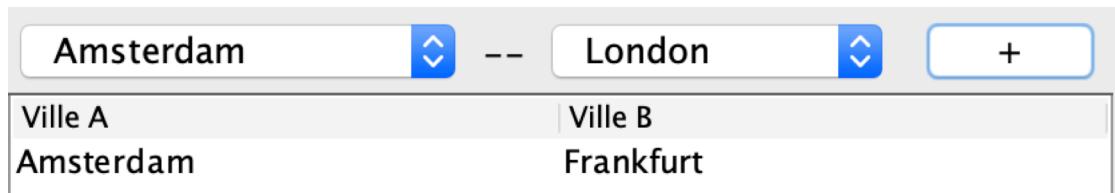


FIGURE 16 – Listes des possessions (on a ajouté Amsterdam - Frankfurt ici)

Si jamais l'utilisateur s'est trompé dans la route sélectionné, il peut double-cliquer sur la route correspondante dans la liste pour la supprimer de la liste et donc la rendre à nouveau disponible dans les boîtes déroulantes.



Pour le panneau des routes possédées par le joueur utilisant l'application, nous affichons le gain total des tronçons que le joueur a indiqué posséder. Cette fonctionnalité lui simplifie la tâche de comptage des points en fin de partie.



FIGURE 17 – Gain total du joueur (Il ne possède ici que Amsterdam-London)

4.4.2 Panneau central

Le panneau central est divisé en deux parties :

- Le choix du chemin
- Les résultats (chemin le plus optimisé et le plus court)

Choix du chemin

Ce panneau permet la sélection du chemin. Sur le dessus on retrouve la boîte déroulante permettant de choisir la ville de départ ainsi que celle pour la ville d'arrivée. On clique sur le bouton "Go !" pour valider le chemin et lancer l'algorithme de recherche.

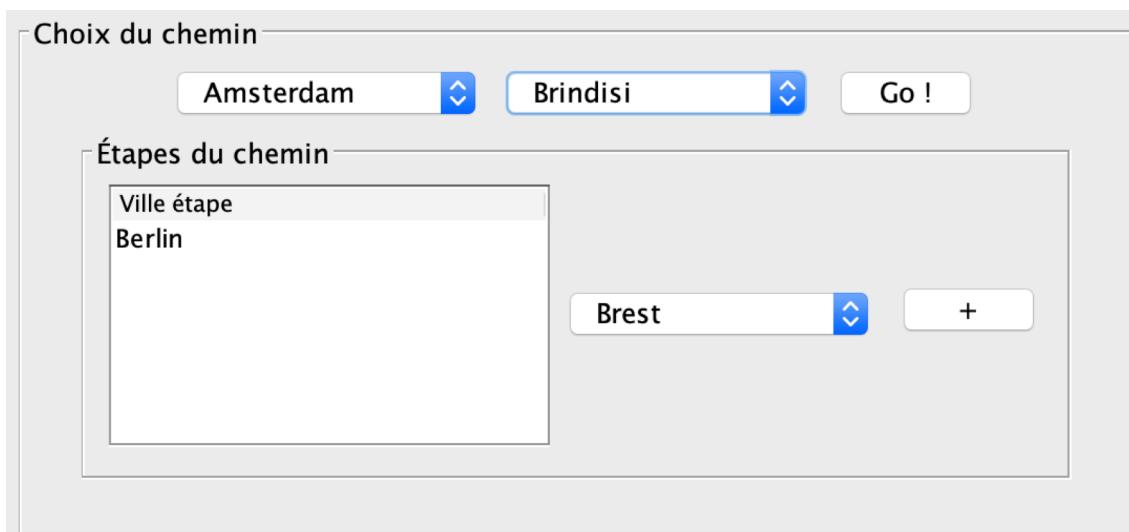


FIGURE 18 – Section du choix de chemin (Ici on va d'Amsterdam à Brindisi en passant par Berlin)

La section interne permet de sélectionner les villes étapes au chemin. Sur le même principe que les panneaux latéraux, une boîte déroulante permet de sélectionner une ville étape, un bouton d'ajout (+) permet d'ajouter la ville à la liste à gauche. Un double-clic dans la liste sur une ville l'enlève de cette liste des étapes.



Affichage des résultats

Les deux sections d'affichages ci-dessous présentent les résultats de l'algorithme : chemin le plus court et chemin le plus optimisé (toujours avec le rapport R_c).

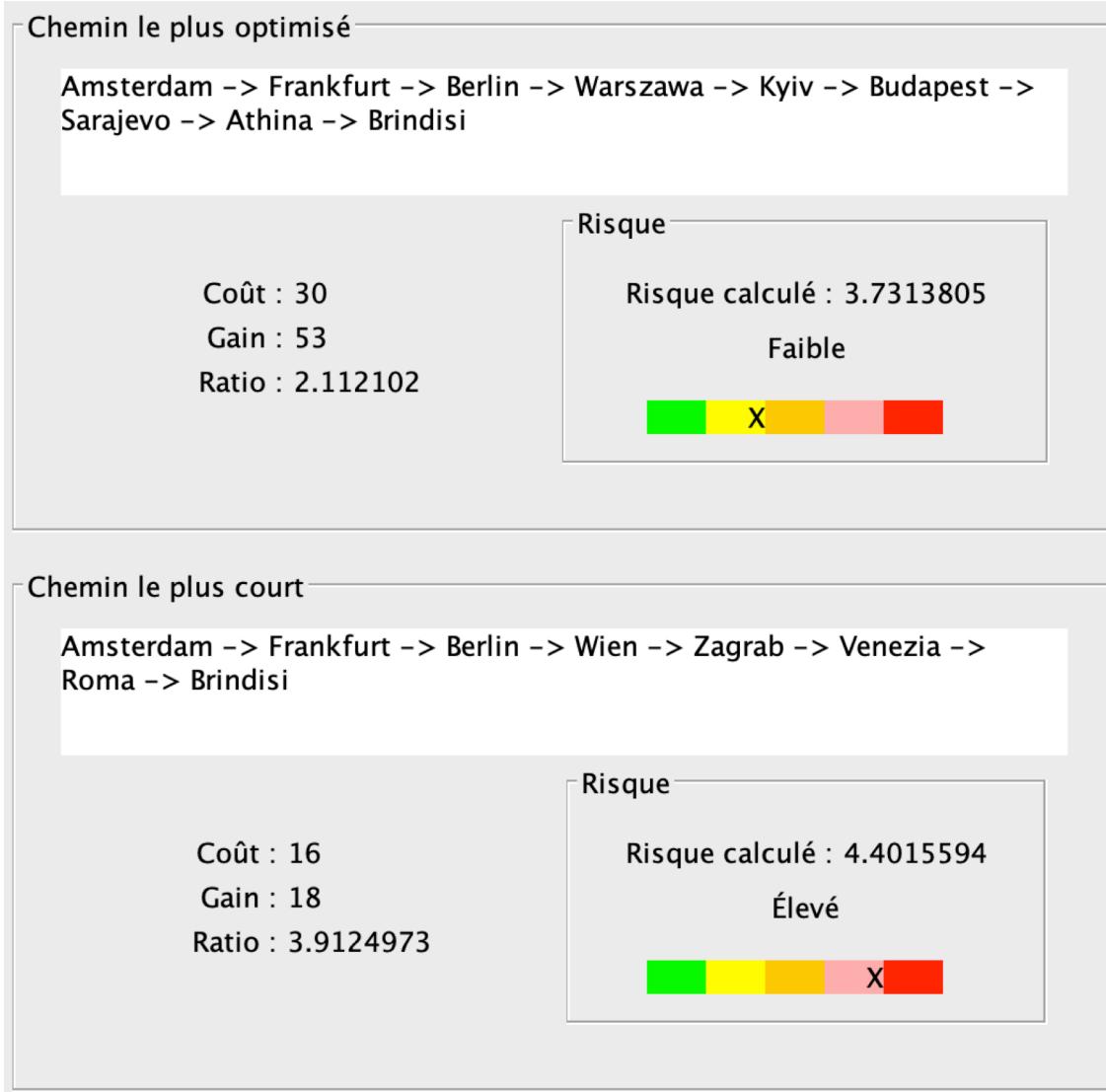


FIGURE 19 – Sections d'affichage des résultats du plus court chemin et du chemin le plus optimisé

Sur chaque section on retrouve les informations de coût, gain, risque et le ratio R_c calculé. On affiche également le chemin à emprunter par le joueur.

Pour le risque, nous le plaçons également sur l'échelle de risque définie précédemment. A chaque marche de l'échelle est associée une couleur :

- Minime : vert clair
- Faible : jaune
- Modéré : orange
- Elevé : rose
- Critique : rouge



En plus de cela, sur l'échelle, le positionnement du chemin se fait différemment en fonction de sa position dans la plage courante de l'échelle. Pour chaque plage de couleur il y a 4 sous-plages :



Cela permet de positionner avec plus de précision le risque total du chemin.

4.4.3 Configuration

Nous stockons pour le moment les informations nécessaires au lancement de l'application dans un fichier de configuration. Ces informations sont les suivantes :

- Si l'utilisation des tronçons doubles est autorisée
- Le chemin vers le plateau de jeu à utiliser

Le fichier se présente de la sorte, et il est importé dès le lancement de l'application.

```
<config>
    <routeDouble>false</routeDouble>
    <boardPath>./src/resources/board/Europe.xml</boardPath>
</config>
```

FIGURE 20 – Fichier de configuration

Par la suite nous pourrons imaginer avoir une interface graphique qui permette à l'utilisateur de rentrer ces informations sans devoir modifier directement un fichier.



5 Mise en conditions réelles



FIGURE 21 – Utilisation de l’application lors d’une partie

Nous avons testé notre programme dans des conditions réelles en faisant plusieurs parties à 3 joueurs. Chacun d’eux avait un rôle :

- Le premier a joué en utilisant uniquement sa propre stratégie
- Le second s’est restreint à utiliser les chemins les plus courts fournis par le programme
- Le dernier a utilisé les chemins optimisés

Dans les 3 parties, le joueur utilisant les chemins optimisés a remporté la victoire. La stratégie du plus court chemin n’est pas forcément pertinente pour toute une partie mais peut dans certains cas il est largement préférable de l’utiliser plutôt que celle du chemin le plus optimisé, notamment en fin de partie.

Il était quelque peu inconfortable pour les joueurs utilisant le programme de devoir rentrer toutes ses possessions et celles des autres joueurs. Une interface montrant la carte du plateau serait sans doute plus confortable (mais aussi plus longue à implémenter). Cependant, il était intéressant de pouvoir avoir les informations quant, au risque des chemins, et à leur coût ainsi que les points que l’on possède.

Lors de nos parties, nous avons joué avec la version Europe qui contient des éléments supplémentaires par rapport à d’autres versions, comme les gares ou les tunnels. Ces éléments n’étaient pas pris en compte dans le programme mais il serait intéressant de les implémenter. Dans l’idéal, il faudrait pouvoir indiquer à quelle version on joue et adapter le programme en fonction de ces spécificités.



6 Conclusion

6.1 Améliorations

Avec plus de temps nous aurions souhaité optimiser notre algorithme principal (pour ne pas avoir à limiter le nombre de tronçons par exemple), ainsi que travailler sur le processus de prise en compte des chemins avec étapes (méthode combinatoire) pour que nous puissions avoir d'avantage de précision qu'avec la méthode simplifiée. Il est pour le moment trop complexe et effectue beaucoup d'appels à notre procédure principale.

De plus il aurait été intéressant d'implémenter la prise en compte des couleurs et l'introduction des probabilités dans notre recherche. Nous n'avons pas priorisé ce sujet, mais sa prise en compte modifierait complètement la façon de jouer de l'utilisateur, car il aurait les informations sur les probabilités associées aux couleurs de wagons à venir. Il pourrait alors choisir les tronçons à construire en priorité à l'aide de ces informations.

Notre format décrivant le plateau de jeu permet de créer facilement les plateaux des différentes extensions du jeu et de pouvoir utiliser notre programme avec celles-ci. Nous pouvons imaginer un menu listant tous les plateaux disponibles et permettant de sélectionner l'extension avec laquelle on souhaite jouer.

6.2 Bilan

Nous sommes tous les deux satisfaits du résultat du travail que nous avons mené pendant ce semestre. Les parties des Aventuriers du Rail que l'on joue avec l'outil sont intéressantes et offrent une autre approche au jeu qu'une partie classique.

La démarche de mener notre propre réflexion à partir de rien était assez inhabituelle pour nous, et même si nous ne nous sommes pas toujours orienté dans la bonne direction dès le début, il était vraiment intéressant de réaliser une telle étude. De plus, la liberté que nous avions dans la réflexion nous permettait réellement d'explorer le sujet sous des axes variés.

La phase de recherche et d'étude a été très instructive, et a pris la majeure partie de notre temps. Nous faisions continuellement des essais pour tester nos formules, algorithmes, diverses idées et cette liberté de retourner le problème comme nous le souhaitions afin de le résoudre était vraiment enrichissante.

Synthèse personnelle de Thomas

Je suis très content d'avoir suivi cette TX ce semestre. On nous transmet souvent un cahier des charges à respecter, des consignes précises, des pistes pour s'orienter dans ce que je fais aujourd'hui à la fois à l'école mais aussi en entreprise : la part de la recherche est très minime. Ce n'est pas spécialement un domaine vers lequel je souhaite m'orienter, mais j'ai trouvé ce travail très intéressant et enrichissant.

J'ai également pu appliquer les compétences que j'ai acquis en entreprise en développement d'interfaces et de logiciel. Même si c'est à une échelle totalement différente, il était intéressant de mettre en pratique pour un tout autre projet ces compétences.



Synthèse personnelle de Laura

J'ai choisi de faire cette TX car, d'une part je connaissais le jeu "les aventuriers du rail" et que la problématique de réflexion autour des différentes stratégies de ce jeu m'intéressait beaucoup, mais aussi parce que j'avais envie de pouvoir toucher un peu à la programmation même en étant en IM. Ayant déjà quelques notions d'algorithmique cette TX m'a permis d'utiliser ces connaissances et de découvrir notamment le langage Java ainsi que des notions d'informatiques comme les graphes ou la complexité.