Projet d'Algorithmique et de Développement Logiciel

Les Aventuriers du Rail

Incrément 8

Séances 25 à 28

Sommaire

1 Préambule **3**

2 Évaluation du travail **3**

3 Travail demandé **3**
- 3.1 Préparation de l'environnement . . . 4
- 3.2 Routes, plateau et graphe . . . 4
  - 3.2.1 Modifier la ville et le plateau . . . 4
- 3.3 La classe Graphe . . . 4
  - 3.3.1 Définir la classe Graphe . . . 5
  - 3.3.2 Créer et initialiser un graphe . . . 5
  - 3.3.3 Créer un graphe pour un joueur . . . 5
  - 3.3.4 Parcours en profondeur du graphe . . . 5
  - 3.3.5 Est-ce qu'une ville est accessible à partir d'une autre ville ? . . . 5
  - 3.3.6 Vérifier le parcours de graphe . . . 6
  - 3.3.7 Graphe et connexité . . . 6
  - 3.3.8 Chemin le plus long (Floyd) . . . 6
- 3.4 Identifier et afficher le vainqueur . . . 6
  - 3.4.1 Modifier la classe Jeu . . . 7
  - 3.4.2 Calcul des bonus et affichage du vainqueur . . . 7
  - 3.4.3 Affichage du vainqueur . . . 7

4 Analyse du code ADR **8**

5 Fin de l'incrément 8 **8**

6 Extensions **8**
- 6.1 Ex-aequo . . . 8
- 6.2 Afficher le tableau des scores . . . 8

1 Préambule

L'incrément 8 du Projet Informatique s'appuie sur la version électronique produite pendant l'incrément 7 du jeu des Aventuriers du Rail (ADR).

L'objectif principal du huitième incrément est de détecter la victoire d'un joueur : à la fin du dernier tour de jeu (voir incrément 4), le jeu calcule les bonus et les scores pour désigner le vainqueur de la partie.

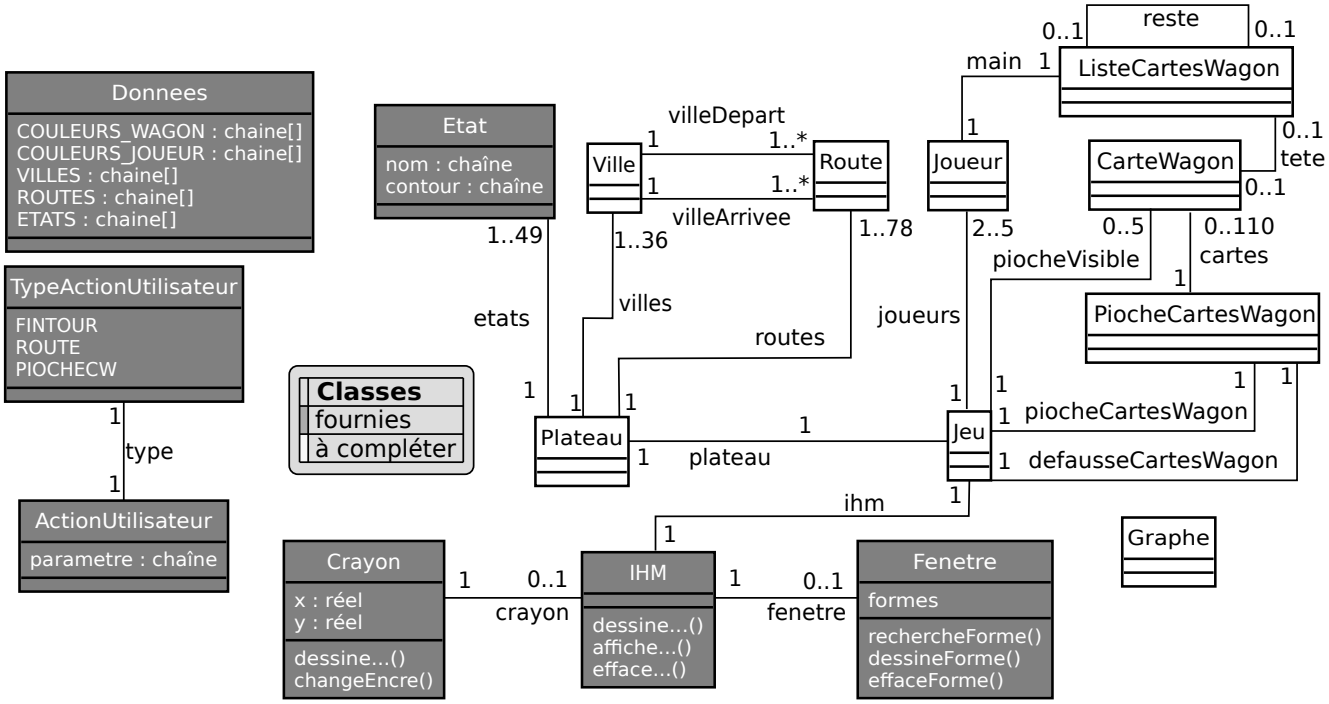

Figure 1: : Classes identifiées dans le jeu des ADR - Incrément 8

Pour implémenter cette fonctionnalité, il est nécessaire de définir une nouvelle classe **Graphe** et de s'appuyer sur les classes **Plateau**, **Ville**, et **Jeu** précédemment implémentées (voir le diagramme de classes UML simplifié et incomplet en figure 1).

2 Évaluation du travail

Le huitième incrément mesure vos compétences à appliquer la méthodologie associée à la théorie des graphes pour analyser une situation et en extraire de l'information utile.

L'évaluation des compétences liées à ce huitième incrément est réalisée au travers d'un examen sur la théorie des graphes.

3 Travail demandé

Une partie des Aventuriers du Rail se termine lorsque les joueurs ont effectué leur dernier tour (voir incrément 4). Le jeu analyse alors la partie et essaie d'identifier :

- le joueur qui détient le chemin le plus long (bonus présent dans les règles du jeu)
- le joueur qui détient le « réseau le plus dense » (bonus non présent dans les règles du jeu)

En fonction des résultats obtenus, le jeu peut alors calculer les scores de tous les joueurs, déterminer le vainqueur et afficher les résultats.

ATTENTION Le calcul du chemin le plus long est légèrement différent de celui de la règle du jeu original : **les boucles ne sont pas autorisées**.

REMARQUE Le calcul des points liés aux « objectifs » réalisés par le joueur n'est pas pris en compte dans cet incrément. Les cartes destination et la gestion associée ne sont pas mises en oeuvre dans cette version du jeu.

REMARQUE Si deux joueurs ou plus obtiennent le même score après calcul des bonus, ils sont déclarés vainqueurs ex-aequo (voir section 6).

3.1 Préparation de l'environnement

Importez l'incrément 8 dans votre projet existant (incréments 1 à 7) en suivant la procédure décrite dans la documentation du logiciel DevCube.

3.2 Routes, plateau et graphe

Pour introduire la théorie des graphes et son utilisation dans le jeu des ADR, il est nécessaire d'effectuer quelques modifications préliminaires sur la classe **Ville** et la classe **Plateau**, de façon à faciliter la représentation d'un graphe, de ses sommets et de ses arêtes.

Le type abstrait *Graphe* qui est utilisé pour représenter le plateau de jeu s'appuie sur des opérations qui manipulent des entiers. Ces opérations peuvent être définies, dans l'absolu pour utiliser les objets du jeu (e.g. routes, villes), mais cela impacte fortement les performances lors des calculs, qui sont nombreux et qui nécessitent dans certains cas d'effectuer des opérations arithmétiques.

De plus, l'information qui nous intéresse dans cette représentation de graphe, est simple : quelles villes (sommets) font partie du graphe et quelles routes (arêtes) relient ces villes. Ce qui est pertinent est donc la possession, par le joueur, d'une route entre une ville de départ et une ville d'arrivée, c'est-à-dire, existe-t-il une arête entre deux sommets. La représentation sous forme d'une matrice d'entiers est suffisante pour stocker cette information.

3.2.1 Modifier la ville et le plateau

On associe une ville à un numéro correspondant à l'indice du sommet qui représente cette ville dans la matrice et on utilise la longueur d'une route entre deux villes pour pondérer l'arête existante entre les deux sommets correspondants.

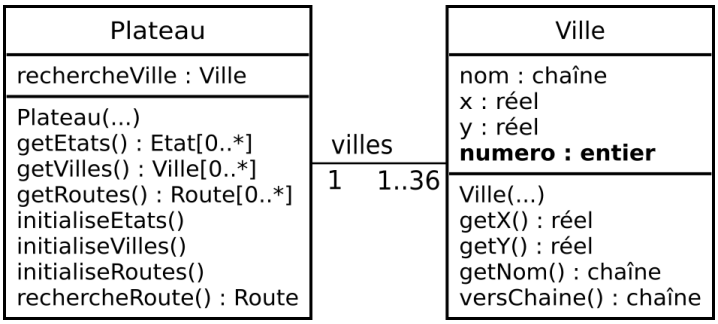

Figure 2: Diagramme de classe des classes Plateau et Ville

Travail à faire

Modifier la classe **Ville** telle qu'elle est représentée dans la figure 2.

Travail à faire

Modifier la méthode `initialiserVilles()` de la classe **Plateau** pour intégrer les changements de la classe Ville.

3.3 La classe Graphe

Le calcul des bonus nécessite d'analyser les « chemins » que le joueur a réalisés pendant la partie.

Pour faciliter l'analyse, on utilise le type abstrait *Graphe*.

La classe **Graphe** est l'implémentation de ce type abstrait qui contient des attributs et des méthodes pour l'analyse du plateau de jeu en fin de partie.

Pour les besoins du jeu, la classe **Graphe** est enrichie de méthodes supplémentaires qui sont décrites dans la figure 3.

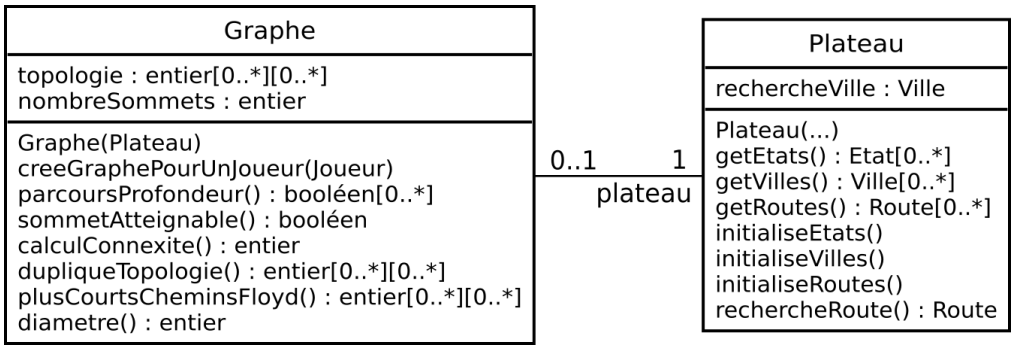

Figure 3: Diagramme de classe de la classe Graphe

3.3.1 Définir la classe Graphe

Travail à faire

Définir la classe **Graphe** telle qu'elle est représentée dans la figure 3.

3.3.2 Créer et initialiser un graphe

La topologie du graphe représente les sommets et les arêtes d'un graphe. Dans le jeu des ADR, les sommets correspondent aux villes et les arêtes aux routes existantes sur le plateau de jeu.

Travail à faire

Écrire le constructeur `Graphe()` de la classe **Graphe**.
Penser à initialiser le nombre de sommets et la topologie du graphe : par défaut, la matrice qui représente la topologie est initialisée avec la valeur 0.
Exécuter la classe **TestGrapheInitialisation** pour vérifier l'implémentation du constructeur de la classe **Graphe**.

3.3.3 Créer un graphe pour un joueur

Pour calculer le score d'un joueur, la topologie du graphe est créée en fonction des routes dont le joueur est propriétaire, et des villes associées à ces routes : pour chaque couple (ville de départ, ville d'arrivée) d'une route dont le joueur est propriétaire, la topologie contient la longueur de la route concernée.

Travail à faire

Écrire la méthode `creeGraphePourUnJoueur()` de la classe **Graphe**.
Exécuter la classe **TestGrapheCreationJoueur** pour vérifier l'implémentation de la méthode `creeGraphePourUnJoueur()`.

3.3.4 Parcours en profondeur du graphe

Le parcours en profondeur permet d'identifier tous les « chemins » disponibles à partir d'une ville donnée, et ceci récursivement.

Travail à faire

Écrire la méthode `parcoursProfondeur()` de la classe Graphe (vue en cours).

3.3.5 Est-ce qu'une ville est accessible à partir d'une autre ville ?

La méthode `sommetAtteignable()` permet de savoir s'il existe un « chemin » entre deux villes du plateau, sous-entendu, s'il existe un ensemble de routes dont le joueur est propriétaire qui relient une ville à une autre.

La méthode `sommetAtteignable()` parcourt en profondeur le graphe du joueur et vérifie si la ville d'arrivée est marquée (vu en cours).

Travail à faire

Écrire la méthode `sommetAtteignable()` de la classe **Graphe**.

3.3.6 Vérifier le parcours de graphe

Travail à faire

Exécuter la classe **TestGrapheParcours** pour vérifier l'implémentation des méthodes `parcoursProfondeur()` et `sommetAtteignable()`.

3.3.7 Graphe et connexité

Le jeu original des ADR propose un seul bonus de fin de partie qui détermine quel est le joueur qui détient le chemin le plus long. Dans la version électronique, un bonus supplémentaire est proposé qui correspond à la densité du réseau créé par le joueur. Plus le joueur a de villes connectées entre elles, plus son réseau est dense : il obtient donc un bonus de points. La densité du réseau est calculée à partir de la notion de connexité des graphes vue en cours et en TP. Une fois le graphe du joueur créé, on calcule le nombre de composantes connexes. On considère alors que la densité du réseau du joueur est maximum lorsque le nombre de composantes connexes est minimum.

La méthode `calculConnexite()` calcule le nombre de groupes connexes présents dans le graphe du joueur courant.

Travail à faire

Écrire la méthode `calculConnexite()` de la classe Graphe.
Exécuter la classe **TestGrapheConnexite** pour vérifier l'implémentation de la méthode `calculConnexite()`.

3.3.8 Chemin le plus long (Floyd)

Le « chemin » le plus long est le bonus original du jeu des ADR. Pour le calculer dans des conditions plus simples que le cas évoqué dans les règles du jeu, on utilise l'algorithme de Floyd-Warshall. Autrement dit, plutôt que de trouver une solution qui calcule le chemin le plus long dans un graphe où il existe des cycles, on calcule l'ensemble des chemins les plus courts entre toutes les paires de sommet du graphe, puis on recherche le maximum de ces chemins les plus courts. On obtient alors le diamètre du graphe, qui correspond au chemin nominal le plus long entre deux villes.

Le joueur qui obtient le plus grand diamètre reçoit le bonus du chemin le plus long dans le jeu.

Travail à faire

Implémenter l'algorithme de Floyd-Warshall vu en cours.
Écrire les méthodes `dupliqueTopologie()`, `plusCourtsCheminsFloyd()` et `diametre()` de la classe **Graphe** pour mettre en oeuvre le calcul du diamètre du graphe.
Exécuter les classes de test **TestGrapheDupliqueTopologie** et **TestGrapheDiametre** pour vérifier l'implémentation du calcul du diamètre (chemin le plus long).

3.4 Identifier et afficher le vainqueur

À la fin d'une partie originale des ADR, les joueurs recomptent les points obtenus pendant la partie et calculent le chemin le plus long pour attribuer le bonus correspondant.

La version électronique nous dispense du recompte des scores puisque les points obtenus pendant la partie sont automatiquement ajoutés au score du joueur (voir incrément 4).

Dans cet incrément, la classe **Jeu** est modifiée pour (1) calculer les bonus du jeu (i.e. chemin le plus long et réseau le plus dense), (2) identifier les joueurs auxquels ces bonus sont attribués, et (3) afficher le vainqueur de la partie.

Pour ce faire, la classe **Jeu** utilise de nouveaux attributs et méthodes pour implémenter ces nouveaux comportements.

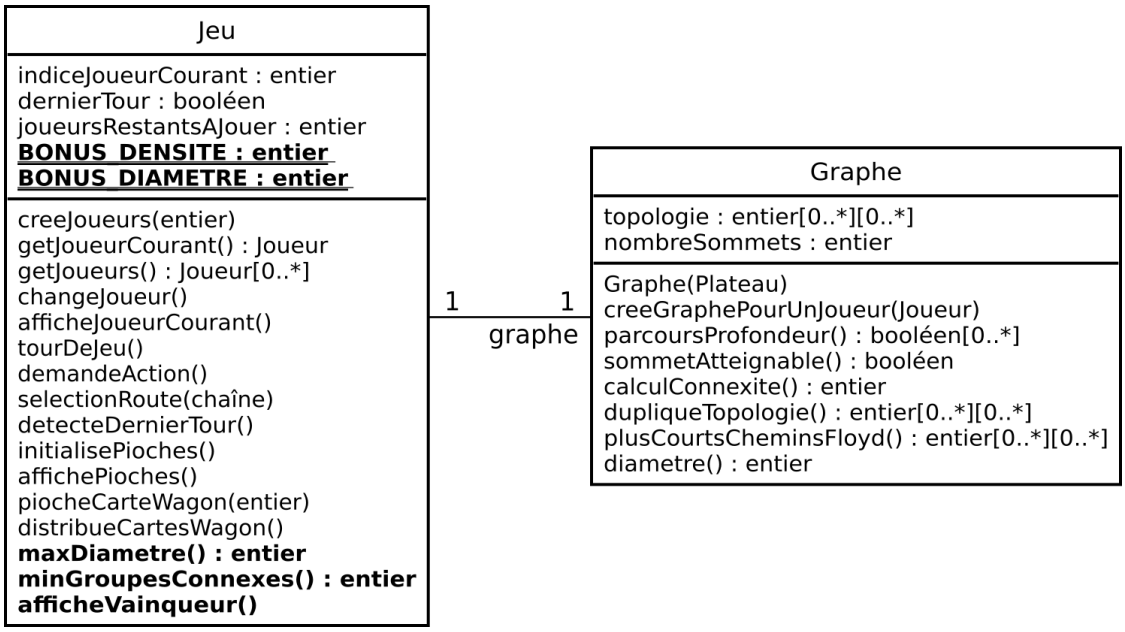

Figure 4: Diagramme de classe de la classe Jeu

3.4.1 Modifier la classe Jeu

Les méthodes `maxDiametre()` et `minGroupesConnexes()` sont des méthodes de recherche d'indice respectivement maximum et minium dans un tableau d'entiers. Elles renvoient respectivement l'indice du joueur dont le diamètre du graphe est maximal et l'indice du joueur dont le nombre de groupes connexes est minimal. Se référer à la documentation Java de l'incrément pour plus d'informations.

Travail à faire

Modifier la définition de la classe **Jeu** telle qu'elle est représentée dans la figure 4.
Implémenter l'accesseur pour l'attribut graphe.
Déclarer `BONUS_DENSITE` et `BONUS_DIAMETRE` comme constantes entières (i.e. variables/attributs de classe) et leur donner une valeur par défaut (10 points dans les règles du jeu)
Écrire les méthodes `maxDiametre()` et `minGroupesConnexes()`.

3.4.2 Calcul des bonus et affichage du vainqueur

La méthode `afficheVainqueur()` remplit trois objectifs :

1. Elle calcule les bonus du jeu et recherche l'indice du joueur qui bénéficie de ces bonus. Les bonus peuvent être attribués à des joueurs différents.
2. Elle calcule le score total de chaque joueur et recherche le score le plus élevé.
3. Elle affiche le vainqueur de la partie (se référer à la documentation de la classe **IHM**)

Travail à faire

Écrire la méthode `afficheVainqueur()`.

3.4.3 Affichage du vainqueur

Travail à faire

Modifier la classe **Jeu** pour afficher le vainqueur en fin de partie. Valider les calculs des bonus et l'affichage du vainqueur en fin de partie par l'exécution de la classe de test **TestJeuVainqueur**. Si des erreurs surviennent lors de la compilation ou de l'exécution, apporter les corrections nécessaires.

4 Analyse du code ADR

L'incrément 8 est le dernier incrément du projet et permet de mettre en place les calculs des bonus du jeu des Aventuriers du Rail. Pour préparer un éventuel travail ultérieur sur le code réalisé, il vous est demandé de faire la synthèse de l'ensemble des modifications apportées aux différentes classes Java du projet sur l'ensemble des incréments.

Travail à faire

En utilisant le formalisme UML vu en cours, proposer une conception la plus complète et la plus à jour possible de l'ensemble des classes du projet. Les modifications apportées à la structure et au fonctionnement des différentes classes doivent être visibles dans les documents de conception proposés.

5 Fin de l'incrément 8

En arrivant à la fin de ce document, votre jeu permet d'afficher le vainqueur d'une partie des ADR. Vous avez, dès lors, une version fonctionnelle du jeu bien qu'incomplète.

Félicitations ! Vous pouvez désormais :

- Compléter et améliorer les documents de modélisation et de vérification/validation
- Travailler sur les graphes (en auto-évaluation)
- Réaliser les extensions de l'incrément 8 (optionnel - section 6)
- Finaliser votre cahier de tests pour l'audit optionnel
- Proposer et implémenter des améliorations à votre version du jeu des ADR (voir Incrément 9)

6 Extensions

Les deux extensions proposées pour ce huitième incrément sont liées : l'extension n°2 peut difficilement être réalisée si l'extension n°1 n'est pas fonctionnelle. En revanche, l'extension n°1 peut fonctionner indépendemment de l'extension n°2 : les résultats obtenus permettront d'être plus précis sur l'attribution des bonus et donc sur la recherche du vainqueur de la partie.

6.1 Ex-aequo

Rien de plus désagréable pour un joueur d'obtenir les mêmes bonus qu'un autre joueur et de ne pas être récompensé de la même manière. La version actuelle du calcul des bonus est injuste : un seul joueur peut remporter le bonus du chemin le plus long ou celui du réseau le plus dense.

Travail à faire

Proposer une implémentation qui identifie tous les joueurs susceptibles d'obtenir l'un ou l'autre bonus.

INDICE N°1 Stocker les indices de tous les joueurs dont le réseau de routes remplit les conditions d'attribution des bonus

INDICE N°2 Utiliser une liste pour stocker les indices

ATTENTION Utiliser **obligatoirement** les listes Java sinon vous ne pourrez pas réaliser l'extension n°2. La JavaDoc de Java contient toutes les informations nécessaires sur l'implémentation et l'utilisation des listes Java.

6.2 Afficher le tableau des scores

L'affichage du vainqueur, dans sa version actuelle, n'est pas suffisant pour :

- afficher l'ensemble des résultats de la partie
- identifier et afficher plusieurs gagnants dans le cas où plusieurs joueurs obtiennent le score maximum (i.e. joueurs ex-aequo sur le score final)

Travail à faire

Remplir le cahier de test avec les tests de validation de l'affichage des scores.

Travail à faire

Écrire une nouvelle méthode `afficheScores()`, dont le fonctionnement est similaire à `afficheVainqueur()`, et qui permet d'afficher les vainqueurs ex-aequo.
INDICE N°1 Utiliser l'extension n°1 pour récupérer les indices des joueurs qui peuvent obtenir les bonus.
INDICE N°2 Lors du calcul des scores finaux, utiliser une liste pour stocker les indices des vainqueurs ex-aequo.
ATTENTION Construire les données nécessaires pour afficher le tableau des scores sur l'interface graphique. Se référer à la documentation de la classe **IHM** pour plus d'informations.

Travail à faire

Valider l'affichage du ou des vainqueurs d'une partie des ADR suite aux modifications apportées par les extensions 1 et 2.
Compléter le cahier de test avec les résultats de l'exécution des tests.
Si des erreurs surviennent lors de la compilation ou de l'exécution, apporter les corrections nécessaires.