Projet d'Algorithmique et de Développement Logiciel

Les Aventuriers du Rail

Incrément 1

Séance 9

Sommaire

1 Introduction

ESEO-Soft, entreprise française de développement, d'édition et de distribution de jeux vidéos, a souhaité redévelopper le jeu de société « Les Aventuriers du Rail ».

La mise en production, initialement prévue le 01/09/2021, a été annulée suite à la négligence des développeurs qui ont perdu la plus grande partie de leur code.

La société MicroTroll a récemment annoncé que leur version, uniquement disponible sur OS Windozz, est en cours de développement.

Pour faire face à la concurrence, ESEO-Soft compte sur vous, très chers codeurs en herbe, pour redévelopper le code source perdu et recréer le jeu « Les Aventuriers du Rail »en moins de 6 mois.

2 Préambule

Le Projet d'Algorithmique et de Développement Logiciel (PADL) comporte huit incréments dans l'objectif de produire une version simplifiée du jeu. Chaque incrément consiste à réaliser un objectif particulier qui se reflète dans l'implémentation, soit par l'ajout de nouvelles classes ou attributs, soit par l'ajout de méthodes, soit les deux.

L'incrément 1 de PADL initie le développement de la version électronique du jeu des Aventuriers du Rail (ADR). L'objectif principal du premier incrément est de réaliser l'affichage du plateau de jeu des ADR, en visualisant la carte du plateau de jeu et les éléments qui la constituent : états, villes et routes.

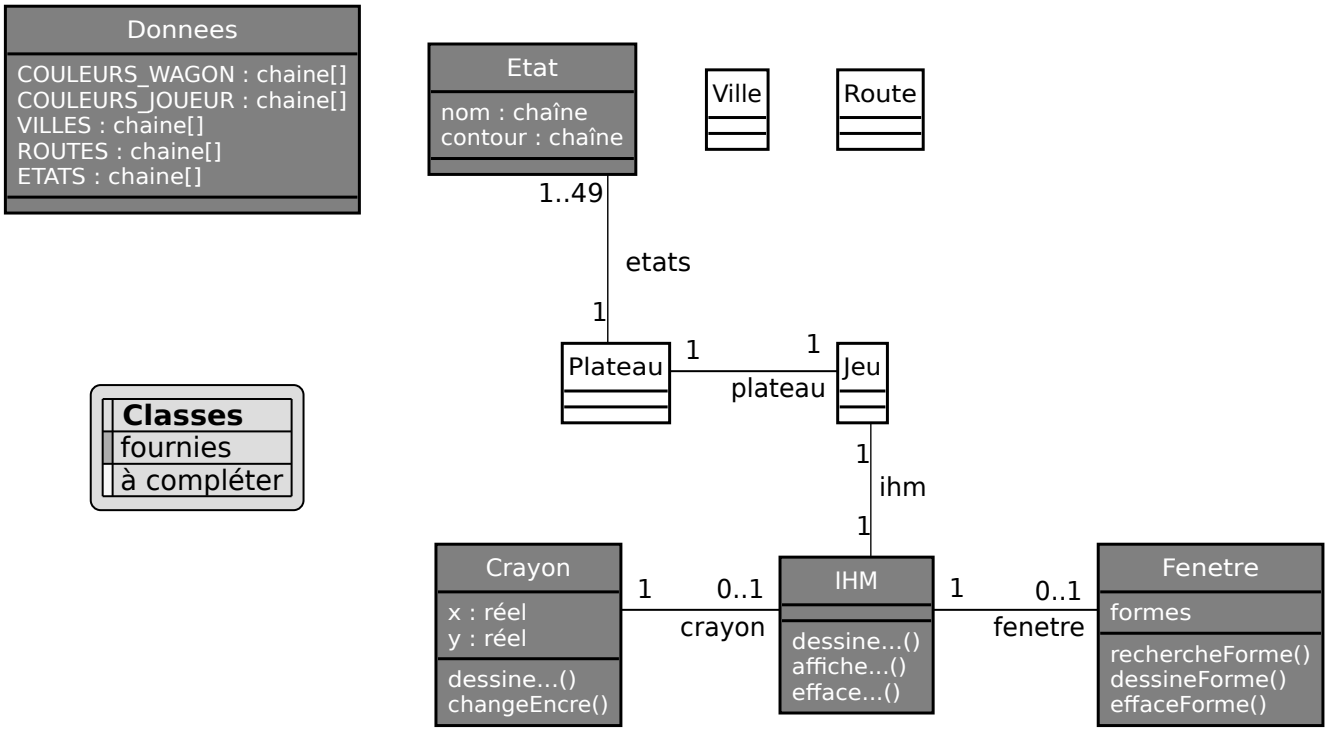

Figure 1: : Classes identifiées dans le jeu des ADR - Incrément 1

L'implémentation de l'incrément 1 se concentre sur les éléments de base du jeu (voir le diagramme de classes UML simplifié et incomplet en Illustration 1) :

- un **Plateau** qui représente la carte « géographique »et les informations du jeu
  - des **Etats** pour dessiner la carte des États-Unis
  - des **Villes** pour représenter les villes du jeu
  - des **Routes** pour relier les villes entre elles

- un **Jeu** qui initialise et affiche le plateau de jeu.

À ces éléments de base s'ajoutent des « utilitaires »(voir Illustration 1), qui permettent de stocker des données, de structurer le code ou de créer l'interface graphique :

- l'interface graphique (**IHM**, **Crayon** *et* **Fenetre**)
- les données du jeu original (e.g les noms des villes et leurs coordonnées) (**Donnees**)

Les « utilitaires »sont des classes qui fournissent des méthodes utiles pour l'implémentation des incréments du projet.

3 Évaluation du travail

Le premier incrément consolide vos compétences en programmation Java en général et en particulier la définition de classes et d'algorithmes.

4 Travail demandé

Pour afficher le plateau de jeu, il est nécessaire de définir les objets de base tels que **Ville**, **Route**, **Etat** et **Plateau** et de proposer un premier algorithme d'exécution du jeu.

4.1 Préparation

Avant de commencer la réalisation de votre projet ADR, il est important de créer un répertoire pour ranger vos fichiers projet.

4.1.1 Le *répertoire racine* de votre travail

Le répertoire racine d'un système de fichier est le répertoire se trouvant le plus haut dans l'arborescence des répertoires. Pour le travail de PADL, le répertoire racine correspond au répertoire contenant tous les documents nécessaires au bon déroulement et à l'implémentation du projet.

Travail à faire

Créer le répertoire racine de votre application dans un endroit approprié (e.g. *~/i1/padl/projet*)

4.1.2 DevCube

Récupérer le fichier DevCube correspondant à l'incrément 1 et disponible sur le campus numérique. Le travail demandé dans ce document est à réaliser avec l'outil DevCube mis à disposition.

4.2 La classe Etat

La classe Etat représente les états des Etats-Unis que l'on affiche sur la carte du plateau de jeu. Ces états servent pour l'interface graphique du jeu. La classe Etat est fournie pour l'incrément 1 et sert d'exemple pour illustrer le travail à faire lors de la déclaration et l'implémentation d'une classe du jeu.

Un état dispose d'un nom et d'un contour : le nom permet de distinguer de manière unique deux états et le contour est une chaîne de caractères qui contient les coordonnées des points permettant de dessiner l'état sur la carte.

Ces informations sont stockées dans les attributs de la classe. Un attribut est nommé en respectant les conventions de nommage du Google Java Style Guide.

Le code Java de la classe **Etat** fourni implémente les attributs et les méthodes proposées dans la figure 2. Il est recommandé de s'inspirer du code Java fourni dans la classe **Etat** pour les implémentations des autres classes du projet.

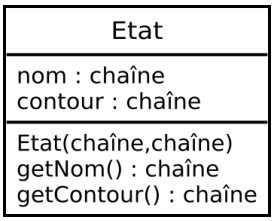

Figure 2: Diagramme de classe de la classe Etat

4.2.1 Accesseurs

Les méthodes commençant par « get »sont des méthodes permettant d'accéder aux attributs de la classe. Par respect des bonnes pratiques de programmation, une classe X ne peut généralement pas accéder directement aux attributs de la classe Y en utilisant l'opérateur d'envoi de message « `.` ». Il faut alors mettre à disposition une méthode particulière dont l'implémentation ne fait que renvoyer la valeur de l'attribut.

Pour l'attribut « nom »de la classe **Etat**, on écrira l'accesseur `getNom()` qui renvoie la valeur de l'attribut « nom »:

```
// Accesseur
String getNom(){
  return this.nom;
}
```

Listing 1: Exemple d'accesseur de la classe Etat

4.2.2 Création d'un état et initialisation

Pour chaque état à afficher sur le plateau, il est nécessaire de créer un nouvel objet de type **Etat**. La création des objets se fait dans une méthode spécifique appelée constructeur qui contient les instructions permettant de « construire »un nouvel objet. Cette méthode doit obligatoirement porter le nom de la classe qui la contient et ne déclare aucun type de retour.

Le constructeur peut utiliser des paramètres pour initialiser le nouvel objet. Dans le cas de la classe **Etat**, on souhaite créer et initialiser un état à partir d'un nom et d'une chaîne de caractères contenant le contour de l'état.

On peut donc écrire le constructeur `Etat()` de la manière suivante :

```
/**
 * Cree un etat
 * @param nom le nom de l'etat
 * @param contour les coordonnees des points a dessiner
 */
Etat(String nom, String contour) {
  this.nom = nom;
  this.contour = contour;
}
```

Listing 2: Constructeur de la classe Etat

4.2.3 Instances et utilisation

La classe **Etat** est définie par ses attributs et ses méthodes, dont le constructeur. Il est désormais possible de créer de nouvelles instances d'états pour chaque état des États-Unis.

4.3 La classe Ville

La classe **Ville** représente une ville du jeu. Une ville dispose d'un nom, d'une abscisse et d'une ordonnée pour l'affichage sur le plateau de jeu.

4.3.1 Définir la classe Ville

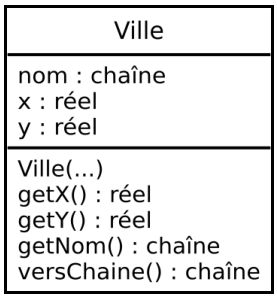

Figure 3: Diagramme de classe de la classe Ville

Travail à faire

Définir la classe `Ville` et ses attributs telle qu'elle est représentée dans la figure 3.

4.3.2 Accesseurs

Travail à faire

Implémenter les accesseurs (voir figure 3) de la classe **Ville** pour chacun des attributs.

4.3.3 Méthode versChaine

La méthode `versChaine()` (voir figure 3) a pour objectif de produire une représentation textuelle d'une **Ville**. Cela permet notamment de vérifier que l'initialisation des villes se déroule correctement et de présenter une information lisible par un humain.

On désire produire une chaîne de caractères qui contient le nom de la ville ainsi que ses coordonnées, séparées par une virgule, entre parenthèses.

Travail à faire

Écrire la méthode `versChaine()` dans la classe **Ville**.
Le résultat de l'appel à la méthode `versChaine()` pour la ville de Vancouver à la position (94,15) doit renvoyer :

```
Vancouver (94.0,15.0)
```

4.3.4 Création d'une ville et initialisation

Pour chaque ville du plateau, il est nécessaire de créer un nouvel objet de type **Ville**.

Dans le cas de la classe **Ville**, on souhaite créer et initialiser une ville à partir d'un nom et de deux réels pour l'abscisse et l'ordonnée de la position de la ville sur la carte.

Travail à faire

Écrire le constructeur `Ville(...)` de la classe **Ville** en prenant exemple sur la classe **Etat**.

4.3.5 Vérifier l'implémentation de la classe Ville

Travail à faire

Exécuter les classes **TestVilleInitialisation** et **TestVilleAffichage** pour valider l'implémentation de votre classe **Ville**.
Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez les avant de poursuivre.

4.4 La classe Route

Le plateau de jeu contient un ensemble de routes prédéfinies qui relient les villes du jeu. Les routes sont utilisées par les joueurs pour « construire »des itinéraires entre plusieurs villes et remplir des objectifs précis.

Dans notre représentation, une route est caractérisée par une couleur, une ville de départ et une ville d'arrivée. La couleur d'une route est l'une des huit couleurs des cartes wagon ou la couleur grise.

Les villes de départ et d'arrivée sont représentées chacune par une instance de la classe **Ville**, comme indiqué dans la figure 4.

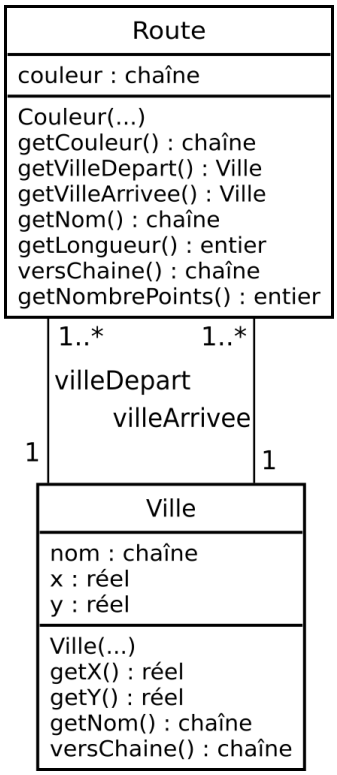

Figure 4: Diagramme de classe de la classe Route

4.4.1 Définir la classe Route

Travail à faire

Définir la classe `Route` et ses attributs telle qu'elle est représentée dans la figure 4.

4.4.2 Accesseurs

Travail à faire

Implémenter les accesseurs (voir figure 4) de la classe **Route** pour chacun de ses attributs.

4.4.3 Méthode getNom

La classe **Route** n'a pas d'attribut pour représenter un nom mais il existe une méthode `getNom()` (voir figure 4) qui génère le nom de la route à partir des villes de départ et d'arrivée.

Le nom de la route est constitué du nom de la ville de départ et du nom de la ville d'arrivée, séparés par un espace, un tiret « - »et un espace.

Travail à faire

Écrire la méthode `getNom()` dans la classe **Route**.
Le résultat de l'appel à la méthode `getNom()` pour la route entre *Vancouver* et *Seattle* doit renvoyer :

```
Vancouver - Seattle
```

4.4.4 Méthode getLongueur

La classe **Route** n'a pas d'attribut pour représenter une longueur. La longueur de la route est calculée lorsque cela est nécessaire, par la méthode `getLongueur()` (voir figure 4). Cette méthode calcule la distance qui sépare les villes de départ et d'arrivée, applique un ratio et renvoie le minimum de la distance entre 1 et 6, respectivement la valeur minimale et la valeur maximale d'une route. Se référer aux commentaires dans le code de la classe **Route** pour plus de détails sur le calcul.

Travail à faire

Écrire la méthode `getLongueur()` dans la classe Route.
Pour la route entre *Vancouver* et *Seattle*, `getLongueur()` doit renvoyer la valeur 1.

4.4.5 Méthode versChaine

La méthode `versChaine()` (voir figure 4) produit une chaîne de caractères qui contient le nom de la route, un espace, une barre oblique (« slash »), un espace, sa couleur et sa longueur séparées par un tiret « - ».

Travail à faire

Écrire la méthode `versChaine()` dans la classe Route.

Utiliser les méthodes `getNom()` et `getLongueur()` pour construire l'affichage attendu.

Le résultat de l'appel à la méthode `versChaine()` pour la route entre Vancouver et Seattle doit renvoyer :

```
Vancouver - Seattle /gris-1
```

4.4.6 Méthode getNombrePoints

Dans les règles du jeu des ADR (disponibles sur le campus), la prise de possession d'une route permet au joueur de gagner des points. Les points gagnés dépendent de la longueur de la route en fonction d'une table de correspondance présente dans le jeu original (voir tableau de décompte des points p.4 de la règle du jeu).

La méthode `getNombrePoints()` (voir figure 4) permet de renvoyer le nombre de points correspondant à la longueur de la route. Pour chaque longueur possible d'une route (entre 1 et 6), la méthode renvoie le nombre de points correspondants (entre 1 et 15).

Travail à faire

Écrire la méthode `getNombrePoints()` dans la classe **Route**.

4.4.7 Création d'une route et initialisation

Pour chaque route du plateau, il est nécessaire de créer un nouvel objet de type **Route**.

Dans le cas de la classe **Route**, on souhaite créer et initialiser une route à partir d'une ville de départ, d'une ville d'arrivée, et d'une couleur.

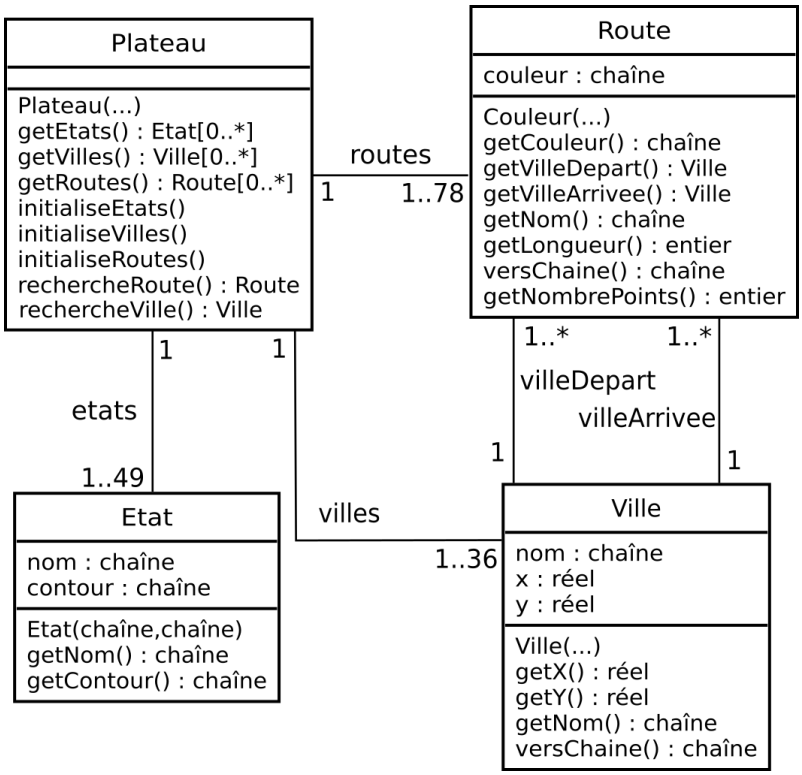

Figure 5: Diagramme de classe de la classe Plateau

Travail à faire

Écrire le constructeur `Route(...)` de la classe **Route**.

4.4.8 Vérifier l'implémentation de la classe Route

Travail à faire

Exécuter les classes **TestRouteInitialisation** et **TestRouteAffichage** pour valider l'implémentation de votre classe **Route**.
Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez-les avant de poursuivre.

4.5 La classe Plateau

La classe **Plateau** est une version électronique du plateau de jeu original. La plateau est dessiné sous la forme d'une carte des États-Unis, sur laquelle les états, les villes et les routes sont dessinés.

Le plateau a donc la charge de stocker, créer et initialiser l'ensemble des éléments du plateau de jeu, dont les états, les routes et les villes. Le stockage de ces éléments de jeu (des objets Java dans notre implémentation) est réalisé par trois tableaux comme indiqué dans la figure 5. Le plateau a également la charge de fournir les premières méthodes de traitement des informations, notamment la recherche d'une ville ou d'une route particulière.

4.5.1 Définir la classe Plateau

Une première version de la classe **Plateau** est fournie dans le fichier *DevCube* de l'incrément 1. Cette version contient la déclaration de la classe **Plateau**, la déclaration de l'attribut `etats`, le constructeur de la classe et la méthode `initialiseEtats()`.

Travail à faire

Compléter la classe **Plateau** et ses attributs telle qu'elle est représentée dans la figure 5.

4.5.2 Accesseurs

Travail à faire

Implémenter les accesseurs (voir figure 5) de la classe **Plateau** pour chacun de ses attributs.

4.5.3 Initialisation des instances de villes, états et routes.

Le plateau de jeu initialise tous les éléments de jeu, c'est-à-dire, les états, les villes et les routes. Concrètement, cela se traduit par l'instanciation des objets correspondants à partir des données fournies dans le projet (voir classe **Donnees**). Les informations du jeu sont disponibles dans un ensemble de tableaux à une ou deux dimensions de la classe **Donnees**. L'objectif est donc de récupérer ces données et de les utiliser pour créer les différents objets.

La méthode `initialiseEtats()` est l'implémentation de cette récupération de données et d'instanciation des objets correspondants : la méthode `initialiseEtats()` récupère les données dans la classe **Donnees**. Ces données contiennent les noms et les coordonnées des points des contours de chaque état. Il est donc nécessaire de parcourir l'ensemble de ces données, et d'initialiser les nouveaux objets **Etat** correspondants.

```
/**
 * Initialise l'ensemble des etats du jeu a partir des donnees fournies.
 * Les donnees utilisees sont transformees pour isoler le nom de l'etat
 * et son "contour".
 * @FOURNI
 */
void initialiseEtats() {
  this.etats = new Etat[Donnees.ETATS.length];
  for(int i=0;i<Donnees.ETATS.length;i++){
    etats[i] = new Etat(Donnees.ETATS[i][0],Donnees.ETATS[i][1]);
  }
}
```

Listing 3: Implementation de la methode d'initialisation des etats du jeu

Les méthodes `initialiseVilles()` et `initialiseRoutes()` réalisent des traitements similaires en fonction des données fournies par la classe **Donnees**.

Travail à faire

Implémenter la méthode `initialiseVilles()` (voir figure 5) pour initialiser les villes du plateau de jeu.

4.5.4 Recherche d'une ville

Lorsque le jeu des ADR s'exécute, les informations disponibles sur l'interface graphique sont accessibles sous la forme de chaînes de caractères. Ainsi on manipule les villes et les routes par leur nom tel que « Boston »pour la ville de Boston ou encore « Seattle - Portland »pour la route entre la ville de Seattle et la ville de Portland.

Lors de la création des routes et de leur initialisation dans la classe **Plateau**, il est nécessaire de pouvoir obtenir un objet **Ville** à partir de son nom. Or dans l'implémentation actuelle, il n'existe aucune méthode disponible à cet effet.

Il faut donc développer une méthode de recherche qui facilitera l'accès aux objets **Ville** de la classe Plateau.

Travail à faire

Écrire la méthode `rechercheVille()` qui recherche le nom d'une ville donné en paramètre.

La méthode `rechercheVille()` parcourt le tableau `villes` et compare le nom passé en paramètre avec le nom de l'objet **Ville** du tableau.

4.5.5 Initialisation des routes

La recherche d'une **Ville** à partir de son nom permet de réaliser l'implémentation de la méthode `initialiseRoutes()` qui crée l'ensemble des routes du jeu.

Travail à faire

Implémenter la méthode `initialiseRoutes()` (voir figure 5) pour initialiser les routes du plateau de jeu.

4.5.6 Création du plateau de jeu et son initialisation

Le constructeur de la classe **Plateau** a la charge d'initialiser les objets du jeu (états, villes et routes). L'initialisation des attributs de la classe est réalisée par les méthodes d'initialisation (voir Listing 3). L'affectation des attributs de la classe peut ainsi être déléguée à des méthodes pur faciliter la lecture du code.

Travail à faire

Compléter le constructeur `Plateau(...)` de la classe **Plateau** pour initialiser les villes et les routes du jeu.

4.5.7 Vérifier l'implémentation de la classe Plateau

Travail à faire

Exécuter la classe **TestPlateauInitialisation** pour valider l'implémentation de votre classe **Plateau**.
Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez-les avant de poursuivre.

4.6 La classe Jeu

La classe **Jeu** représente le programme principal de votre jeu. Cette classe permet l'exécution du jeu des Aventuriers du Rail.

Dans la version qui vous est fournie, la classe **Jeu** crée et initialise un plateau de jeu et affiche la carte des États-Unis avec les états dessinés sur l'interface graphique.

L'objectif, dans cet incrément, est de terminer l'affichage du plateau de jeu en y incluant les villes et les routes, de façon à obtenir le résultat présenté en illustration 6.

4.6.1 Méthode affichePlateau

La méthode `affichePlateau()` parcourt les informations du plateau de jeu et demande à ce que ces informations soient dessinées sur l'interface graphique. La version initiale affiche les états présents et initialisés par le plateau de jeu.

Travail à faire

S'inspirer de l'affichage des états sur la carte puis dessiner les villes et les routes du jeu.

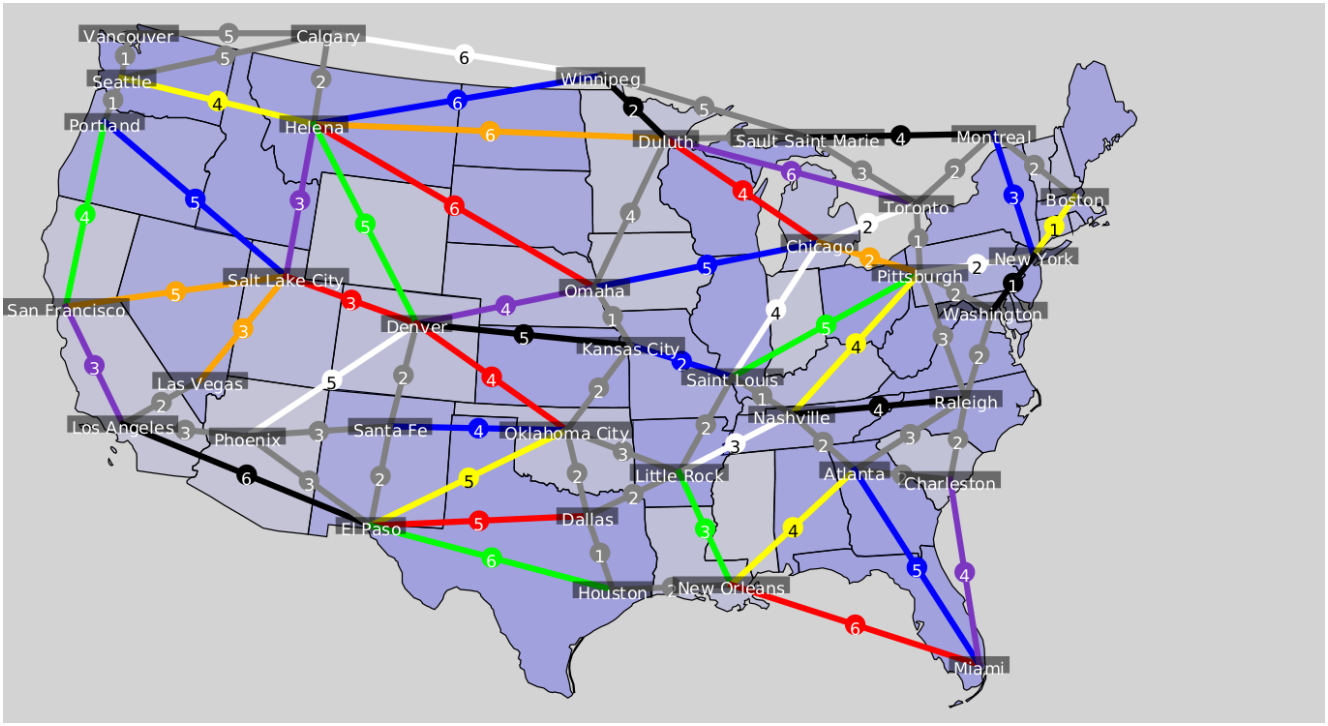

Figure 6: Interface graphique du jeu des ADR : résultat de l'incrément n°1

4.6.2 Méthode main

La méthode `main()` est la méthode principale d'un programme Java et le point d'entrée du programme. Le point d'entrée doit être unique, sans quoi il est impossible de savoir où commencer l'exécution du programme. La méthode `main()` ne renvoie aucune valeur et prend en paramètre un tableau de chaînes de caractères. Ce tableau peut contenir des paramètres qui sont passés sur la ligne de commande lors de l'exécution de l'application.

En Java, la méthode `main()` est déclarée publique et précédée du mot clef `static`. La nature et l'utilisation du modificateur `static` seront abordées au semestre 6.

Travail à faire

Dans la méthode `main()` de la classe **Jeu**, déclarer puis créer un nouvel objet **Jeu** et demander à afficher le plateau de jeu.

4.6.3 Exécution du jeu

Travail à faire

Utiliser DevCube pour exécuter la classe **Jeu** et observer le résultat de votre implémentation. En cas de problème, vérifier les implémentations des villes, routes, plateau et du jeu pour effectuer des corrections si nécessaire.

5 Analyse du code ADR

Pour faciliter la poursuite du développement du jeu et comprendre le contenu du projet, il est demandé de fournir une analyse et une conception préliminaire des classes **IHM** et **Crayon**.

Travail à faire

Proposer un diagramme de classes, similaire à la figure 5, qui liste les attributs et les associations entre ces classes.

Dans ces deux classes, lister les méthodes qui sont utilisées pour afficher le plateau de jeu.

6 Fin de l'incrément 1

Vous avez atteint le principal objectif qui est d'afficher le plateau de jeu et les éléments qui le composent.
En fonction du temps qu'il vous reste avant l'incrément suivant, vous pouvez :

- Réaliser les extensions de l'incrément 1 (optionnel - section 7)
- Travailler sur l'implémentation de la classe **CarteWagon** (en auto-évaluation)
- Commencer l'incrément 2.

7 Extensions

7.1 Recherche d'une route

L'implémentation actuelle de la classe **Plateau** ne permet pas de retrouver un objet **Route** à partir de son nom. Ce traitement est utile dans la suite du projet pour permettre de récupérer une **Route** sur laquelle on peut ensuite effectuer des traitements.
Cette nouvelle méthode s'inspire de la recherche d'une **Ville** (voir Section 4.5.4).

7.1.1 Recherche d'une route

Travail à faire

Écrire la méthode `rechercheRoute()` qui recherche le nom d'une route donné en paramètre.

La méthode `rechercheRoute()` parcourt le tableau `routes` et compare le nom passé en paramètre avec le nom de l'objet **Route** du tableau.