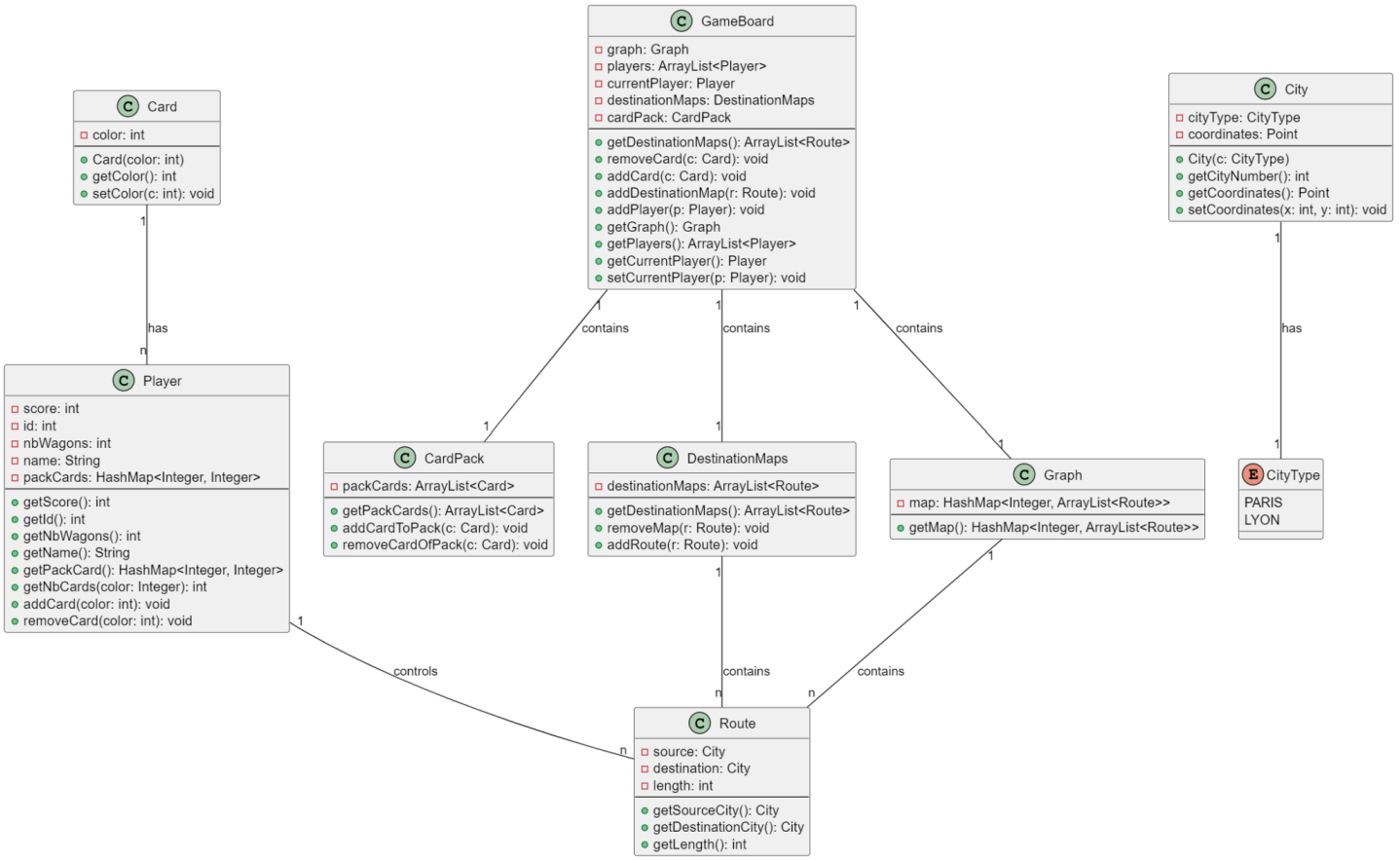
ED2 : Les aventuriers du rail

SEMAINE 1-2 (29/01/2024 - 05/02/2024)

- Tester la version en ligne du jeu afin de mieux comprendre les règles
-Réfléchir à la modélisation du jeu (classes, énumération, représentation des villes, routes…)
-Recherche de ressources visuels (cartes wagons)

Diagramme :



Explications:

Concernant la représentation du graphe (des villes et des routes), nous avions le choix entre les matrices d'adjacence ou les listes d'adjacences (sous forme de tableau de listes, où l'indice i représente une ville, et la liste associée à cet index représente les villes qui lui sont adjacentes (si l'index i représente la ville de Lyon, alors la liste associée a cet index correspond aux villes qui possède un chemin la liant à Lyon) . Nous avons finalement opté pour la seconde option. Les arêtes représentent les routes, et les sommets correspondent aux villes. Pour résumer :

I. Matrice d'Adjacence :

**Avantages Majeurs :

1. Temps constant pour la récupération d'Arêtes (or le seul moment où l'on aura vérifié s'il existe une route entre deux villes, sera lors de la construction des chemins).

ED2 : Les aventuriers du rail

2. Efficacité d'Espace pour les Graphes Denses :

- Convient aux graphes avec beaucoup d'arêtes, et c'est bien notre cas (où la majorité des entrées sont remplies. Cela nous évite de parcourir toute la liste des villes pour savoir si elle est liée, mais ce n'est pas crucial).

**Inconvénients :

1. Inefficacité d'Espace pour les Graphes Dispersés (graphes avec moins d'arêtes) car espace de stockage est de l'ordre de O(n^2) ou n est le nombre de villes (donc consommation de mémoire importante)

2. Inefficace pour les Graphes Dynamiques : (ajout/ suppression de villes). Or ce n'est pas notre cas pour l'instant.

II. Liste d'Adjacence :

**Avantages Majeurs :

1. Efficacité d'Espace pour les Graphes Dispersés : car l'espace requis est de l'ordre de O(n+m) où n est le nombre de villes et m est le nombre de routes.

2. Efficacité Mémoire :

- Consomme moins de mémoire par rapport aux matrices d'adjacence, en particulier pour les grands graphes (mais nos entrées seront pratiquement toutes remplies)

3. Efficace pour les Graphes Dynamiques : (pas notre cas)

4. Itération Rapide sur les Voisins : (util lors de la construction de routes)

**Inconvénients :

1. Récupération d'Arêtes Plus Lente : (Or, on ne vérifiera pas si deux villes sont connectées, pour vérifier si le coup d'un joueur est autorisé ou pas)

2. Surcoût Mémoire Supplémentaire :

- Nécessite une mémoire supplémentaire pour stocker des pointeurs/références pour chaque sommet.

3. Plus Lent pour les Graphes Denses :

- Peut être moins efficace pour les graphes denses où une partie significative des sommets a de nombreux voisins

Conclusion :

Dans notre cas, le graphe n'est pas dispersé. On aurait donc tendance à opter pour les matrices. Or, vérifier s'il existe une route entre deux villes/ pays n'est pas une opération à laquelle on aura

souvent recours. Il serait donc meilleur d'opter pour l'option des listes afin d'éviter d'utiliser de la mémoire en vain (tableau de tableau).

Voici un simple diagramme pour représenter nos différentes classes du modèle. Nous n'avons pas encore mis en place

Comment sont représentés les villes/pays ?

```
Map.txt
 • France :
0,1,5 (Paris, Lyon, Toulouse)
 • Europe :
2,3,4 (Espagne,Belgique, Italie)
```

```
Routes.txt
 • Europe:
3,5 (Espagne Italie)
 •  France :
0,3 (correspond a Paris Toulouse, voir enum)
0,1 (correspond a Paris Lyon)
 • Amérique :
//De meme pour les autres cartes
```

Comment sont répartis les villes/Pays ?

On pensait à créer des parseurs pour deux fichiers textes :

-« Map.txt » L'un pour définir quelles sont les villes/Pays à placer en fct de la carte choisie par le joueur

- « Routes.txt » Le second pour définir les différentes routes à dessiner entre les villes/pays

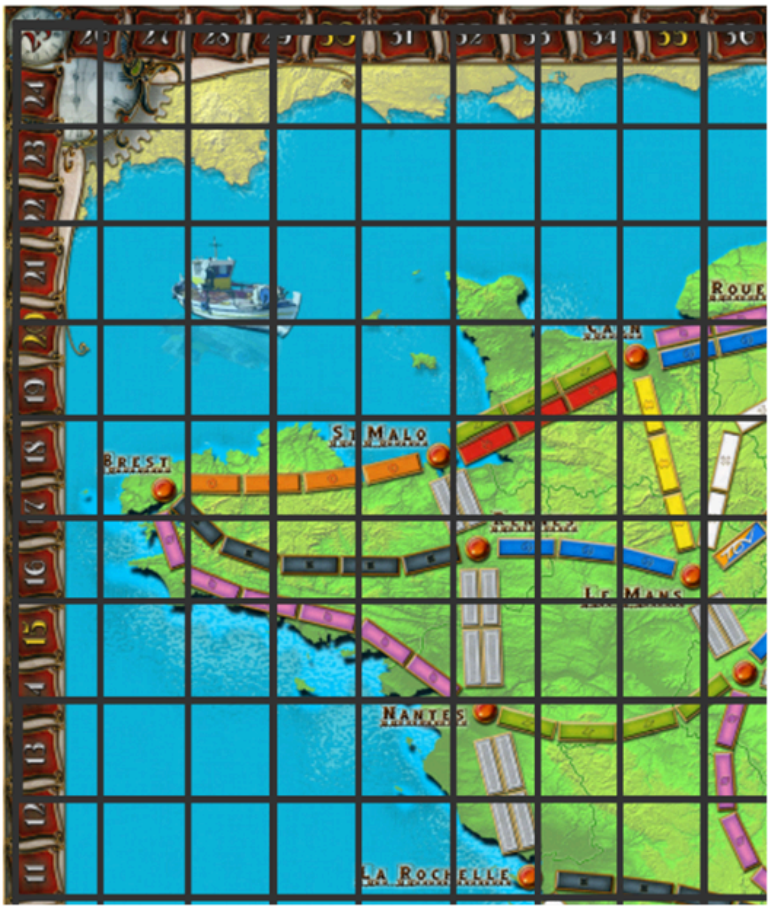
Comment remplir l'attribut «map » du fichier Graph (Map <City, List<Route>> map) ?

Le joueur choisit une carte, et en fct de son choix, on parcourt « Map.txt » et on crée des instances de City qu'on ajoute à la map (en tant que clé). Puis on parcourt le fichier « Routes.txt » et on ajoute à la map du Graph la liste des routes correspondantes (autant que valeur).

Comment placer les villes/pays sur la Carte ?

Les coordonnées de chaque ville/pays correspondent à la position à laquelle elle sera placée. Exemple, si les coordonnées sont 1,4 (pour BREST), on sait qu'on doit placer la ville comme suit (on n'aura qu'à convertir les coordonnées 1,4 en les coordonnées visuelles sur l'interface graphique):

ED2 : Les aventuriers du rail



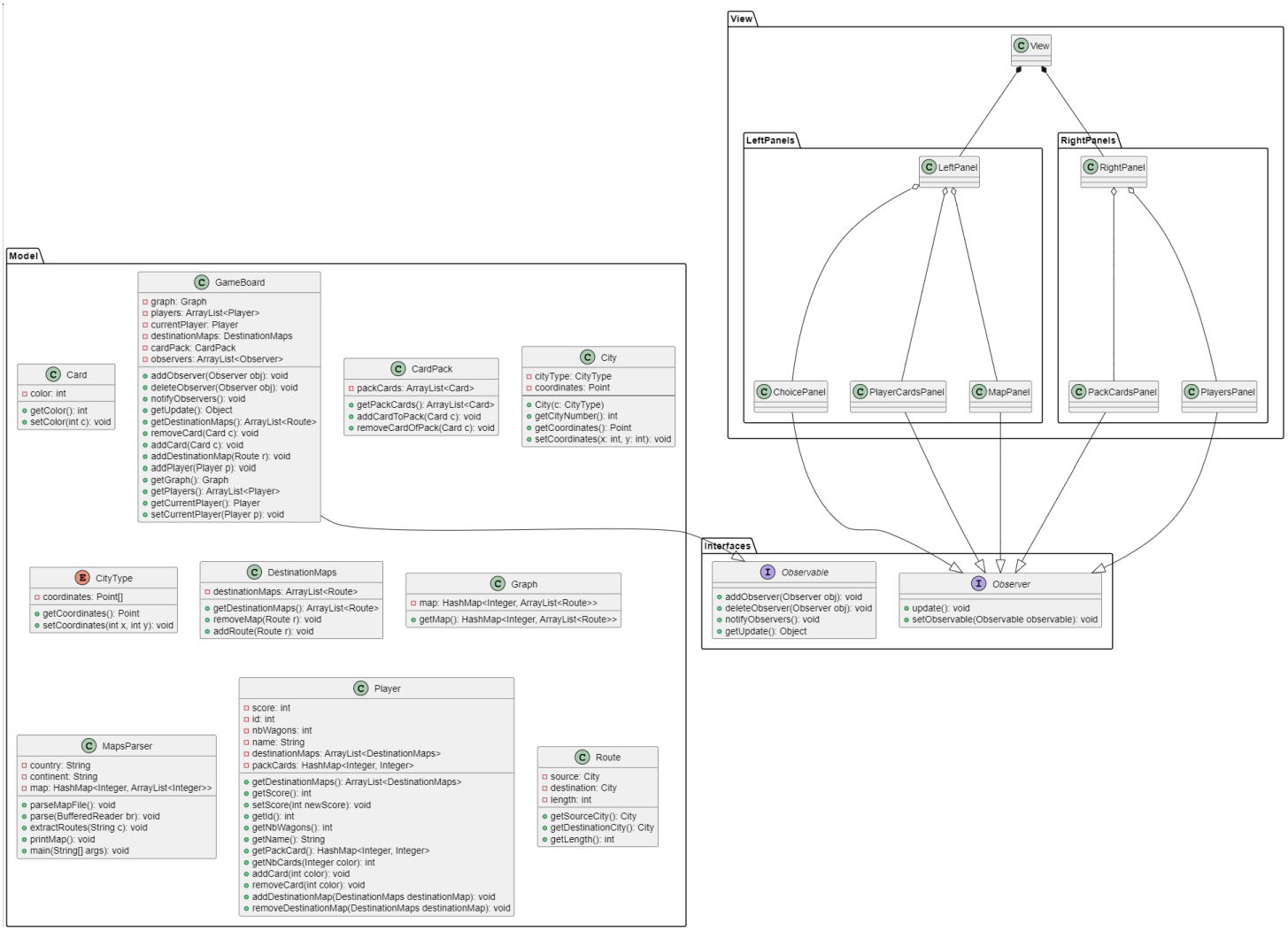
Cette solution n’est évidemment pas optimale (deux wagons peuvent être sur la même case, et solution pas très précise en général). Pourriez-vous nous orienter vers une meilleure solution ?
Nous n’avons pas encore réfléchi au système des « cartes destinations », et le diagramme pour le Controller n’a pas encore établi

ED2 : Les aventuriers du rail

SEMAINE 3 (12/02/2024)

- Choix du design pattern Observer (Lien avec MVC et Observer)
- Création du menu (avec choix du nombre de joueurs, des cartes de continents et de pays) -En cours d'amélioration-
- Création du parseur de fichier texte
- Création du Panel correspondant aux cartes du joueur courant -En cours d'amélioration-
- Création du Panel de choix des cartes destinations/cartes wagon
  -En cours d'amélioration-

Diagramme :



ED2 : Les aventuriers du rail

SEMAINE 4 (18/02/2024)

-Refactoriser le code (pas d'affichage dans les classes listener).
-Ajout des villes sur la carte
- Refactoriser le code pour l'affichage de la main du joueur
-Avancer sur le menu
-Avancer sur l'affichage des panels des joueurs
-Modification de l'affichage de la main du joueur.

SEMAINE 5 (25/02/2024)

- Effectuer le lien entre les cartes destinations/Wagons et le model (pattern observer)
- Repérer le clique du joueur sur une route et changer la couleur de la route
- Début de la modélisation du tour par tour
- Ajout de méthodes dans la classe GameBoard.
- Finition de la partie graphique de la partie Verte (Main du Joueur)