Rapport de projet – SAE 2.01/2.02

1. Introduction

Dans le cadre de la SAÉ 2.01 et 2.02, nous avons développé une application Java visant à gérer un réseau de livraison de cartes Pokémon entre des membres répartis dans plusieurs villes. L'objectif est de modéliser ces échanges sous forme de graphe orienté, de lire des scénarios de transactions, de construire dynamiquement le graphe associé, puis d'en analyser les propriétés comme le degré d'entrée des sommets, les cycles éventuels, ou encore de prévoir des itinéraires de livraison efficaces.

L'application respecte l'architecture MVC (Modèle-Vue-Contrôleur) et est conçue pour être facilement extensible. Nous avons également prévu une interface utilisateur via JavaFX permettant de sélectionner les scénarios à charger.

---

2. Conception

2.1 Architecture globale

Le projet suit une structure MVC :

- **Modèle** : contient la logique métier (lecture des fichiers, structure du graphe, algorithmes).
- **Vue** : interface utilisateur en JavaFX permettant de choisir un scénario.
- **Contrôleur** : fait le lien entre la vue et le modèle ; il gère les événements de l'interface.

2.2 Présentation des composants

2.2.1 Modèle

Le package modele contient les classes principales suivantes :

- **LectureScenario** : lit les fichiers de scénario, génère les listes de villes, les membres et construit le tableau d'adjacence représentant les liaisons entre villes.

- **Structures de données utilisées** :
    - HashMap<String, String> : pour associer les membres à leurs villes.
    - ArrayList<String> : pour conserver les noms des villes.
    - int[][] : matrice d'adjacence représentant le graphe des échanges.
- **Graphe** : représente un graphe orienté. Il est construit à partir d'un tableau de voisins (liste d'adjacence) généré par LectureScenario.

  Structures de données utilisées :
    - ArrayList<Integer>[] : pour stocker les voisins de chaque sommet.
    - int[] : pour calculer les degrés entrants de chaque sommet.
- **Fonctionnalités clés** :
    - Calcul du degré entrant de chaque sommet.
    - Représentation textuelle du graphe.
- **IDException** : exception personnalisée lancée si des erreurs d'identifiant ou de ville sont détectées lors du chargement d'un scénario.

2.2.2 Vue

Le package vue contient :

- **ChoixScenarioHBox** : composant JavaFX contenant une ComboBox listant les fichiers de scénarios disponibles dans le dossier Scenario, ainsi qu'un bouton "Valider".

  Lors du clic, le contrôleur est appelé pour charger le graphe correspondant au fichier sélectionné.
- **VBoxRoot** : conteneur principal de la fenêtre JavaFX, qui intègre ChoixScenarioHBox.

2.2.3 Contrôleur

Le contrôleur (nouvellement créé) se charge de :

- Réagir à la sélection et validation d'un scénario.
- Appeler les méthodes du modèle (LectureScenario, Graphe).
- Transmettre les informations à afficher à la vue (console ou interface graphique future).

Cela permet une séparation claire entre la logique de traitement et l'affichage.

2.3 Algorithmes

Les algorithmes principaux sont :

- **Construction du graphe à partir du scénario** : lecture du fichier ligne par ligne, association des membres aux villes, puis construction d'un graphe orienté où une arête est ajoutée si un membre envoie une carte à un autre.
- **Calcul des degrés entrants** : pour chaque sommet, on compte le nombre de fois qu'il apparaît comme destination dans les listes de voisins. Cela donne une idée des points de convergence dans le réseau.

---

3. Conclusion

Bilan

Le projet a permis de mettre en place une architecture claire, modulable et bien structurée. La gestion des scénarios fonctionne, la lecture des fichiers est robuste et la génération du graphe est opérationnelle.

Nous avons pu séparer proprement les responsabilités entre le modèle, la vue et le contrôleur.

Tâches non réalisées

- **Affichage visuel du graphe** dans l'interface JavaFX (sous forme de nœuds et arêtes).
- **Rédaction de tests unitaires** pour vérifier les méthodes du modèle de manière automatique.
- **Algorithme qui calcule les k meilleures solutions**

Perspectives d’évolution

- Ajouter une **représentation graphique du graphe** en JavaFX (avec la bibliothèque GraphStream ou une librairie maison).

- Implémenter des **algorithmes de parcours** (DFS, BFS), de **détection de cycles**, ou encore **de tri topologique**.

- Ajouter des **tests JUnit** pour fiabiliser l’application.

- Permettre la **sauvegarde/export** du graphe ou de ses propriétés (CSV, JSON, etc.).