Projet d'Algorithmique et de Développement Logiciel

Les Aventuriers du Rail

Incrément 4

Séances 14 et 15

Sommaire

1 Préambule

L'incrément 4 de PADL s'appuie sur la version produite pendant l'incrément 3 du jeu des Aventuriers du Rail (ADR).

Le premier objectif du quatrième incrément est d'implémenter une nouvelle version de la prise de possession d'une route : suite à la prise de possession d'une route (voir incrément 3), le score et le nombre de wagons du joueur sont mis à jour.

Le second objectif du quatrième incrément est de détecter la fin de partie : la mise à jour du nombre de wagons du joueur permet de (1) détecter la fin de partie et (2) démarrer le dernier tour de jeu pour l'ensemble des joueurs de la partie.

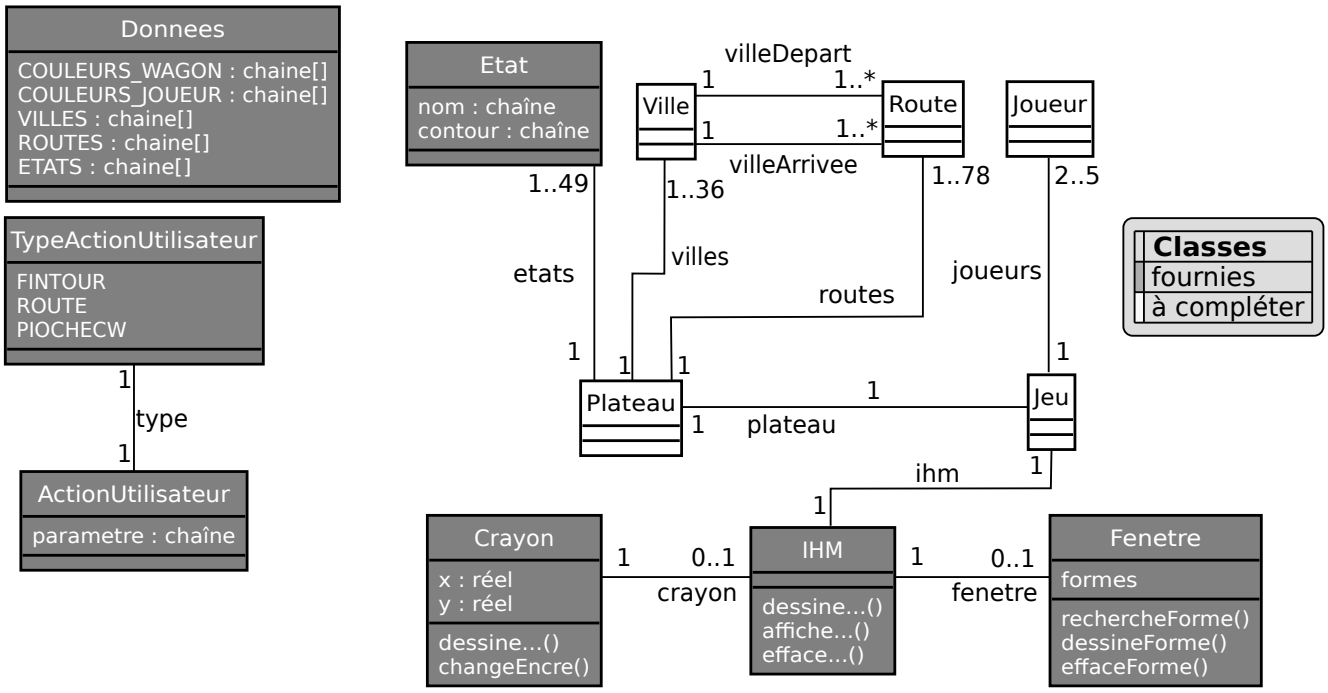

Figure 1: : Classes identifiées dans le jeu des ADR - Incrément 4

Pour implémenter cette fonctionnalité, nous nous appuyons sur les classes **Joueur** et **Jeu** déjà implémentées.

Le déroulement de ce quatrième incrément contient aussi le cours sur les diagrammes d'activités pour la thématique « Algorithmique et Modélisation » ainsi que la suite du cours sur la thématique « Vérification et Validation».

2 Évaluation du travail

Le quatrième incrément consolide vos compétences en programmation Java en général.

3 Travail demandé

La prise de possession d'une route a un impact sur les données du joueur, si l'on se réfère aux règles du jeu des ADR. Il est demandé de répercuter ces changements sur les données du joueur courant et de les afficher sur l'interface graphique.

3.1 Préparation de l'environnement

Importez l'incrément 4 dans votre projet existant (incréments 1, 2 et 3) en suivant la procédure décrite dans la documentation du logiciel DevCube.

3.2 La classe Joueur

La classe Joueur dispose des attributs permettant de stocker son score et son nombre de wagons. Pour simplifier la modification de ces informations, nous allons définir deux nouvelles méthodes, l'une pour ajouter des points au score du joueur, l'autre pour retirer des wagons (voir figure 2).

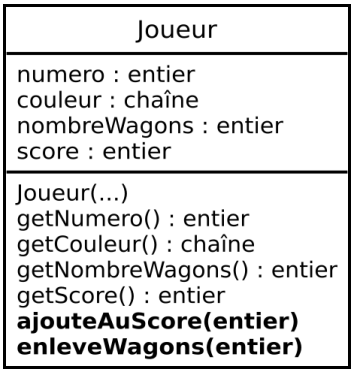

Figure 2: Diagramme de classe de la classe Joueur

3.2.1 Modifier la classe Joueur

La méthode `ajouteAuScore()`, comme son nom l'indique, ajoute au score du joueur le nombre de points passé en paramètre.

La méthode `enleveWagons()`, elle, enlève au joueur le nombre de wagons passé en paramètre.

Travail à faire

Modifier la classe **Joueur** en implémentant les méthodes de la figure 2.

3.2.2 Valider l'implémentation de la classe Joueur

La plupart des classes de test sont fournies par l'équipe pédagogique pour valider l'implémentation des fonctionnalités du jeu. Dans les incréments 1 à 3, seul le résultat compte dans le travail réalisé : l'implémentation d'une fonctionnalité répond à un nombre suffisant de critères pour considérer que le programme réalise correctement une opération.

À partir de l'incrément 4, le travail demandé implique à la fois l'implémentation, l'exécution et le report des tests dans le cahier de test du Projet Informatique (cahier de test initié durant l'incrément 3).

Travail à faire

Exécuter les classes **TestJoueurInitialisation**, **TestJoueurScore** et **TestJoueurWagons** pour vérifier l'implémentation des méthodes de la classe Joueur. Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez-les avant de poursuivre.

Travail à faire

Réfléchir aux tests devant être effectués afin de valider les fonctionnalités implémentées et compléter le cahier de test avec les tests de validation. Le résultat de l'exécution de ces tests sera consigné par la suite une fois les parties 3.3.1 à 3.3.5 réalisées.

3.3 La classe Jeu

Dans cet incrément, la classe **Jeu** est modifiée pour afficher les informations du joueur courant sur l'interface graphique et pour mettre en oeuvre les mécanismes de fin de partie.

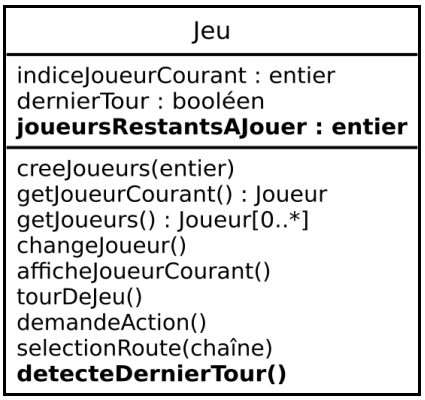

Figure 3: Diagramme de classe de la classe Jeu

3.3.1 Modifier la classe Jeu

Travail à faire

Modifier la définition de la classe Jeu telle qu'elle est représentée dans la figure 3.

3.3.2 Accesseurs

Travail à faire

Implémenter l'accesseur en lecture pour l'attribut `joueursRestantsAJouer` de la classe Jeu. Implémenter l'accesseur et le mutateur pour l'attribut `dernierTour` (défini dans l'incrément 2). Se référer aux Règles de programmation et bonnes pratiques pour choisir la signature de vos accesseurs.

3.3.3 Méthode afficheJoueurCourant

La méthode `afficheJoueurCourant()` (produite dans l'incrément 2) est modifiée pour afficher la couleur du joueur, un score et un nombre de wagons tel que présenté en figure 4.

Figure 4: Affichage de la couleur, du score et du nombre de wagons du joueur courant (e.g. premier joueur et quatrième joueur)

Travail à faire

Modifier la méthode `afficheJoueurCourant()` pour afficher la couleur du joueur, le score du joueur et son nombre de wagons restants en s'inspirant de l'affichage du numéro du joueur (incrément 2) et en consultant la documentation de la classe IHM.

3.3.4 Méthode selectionRoute

Dès lors que le nombre de wagons du joueur peut être modifié, la méthode `selectionRoute()` doit vérifier le nombre de wagons restants au joueur pour autoriser la prise de possession d'une route.

Lorsque la prise de possession est possible, le joueur « gagne » les points correspondants à la longueur de la route prise et « perd » les wagons utilisés pour prendre la route.

La modification de l'algorithme initial de la méthode `selectionRoute()` est présentée en illustration 5.

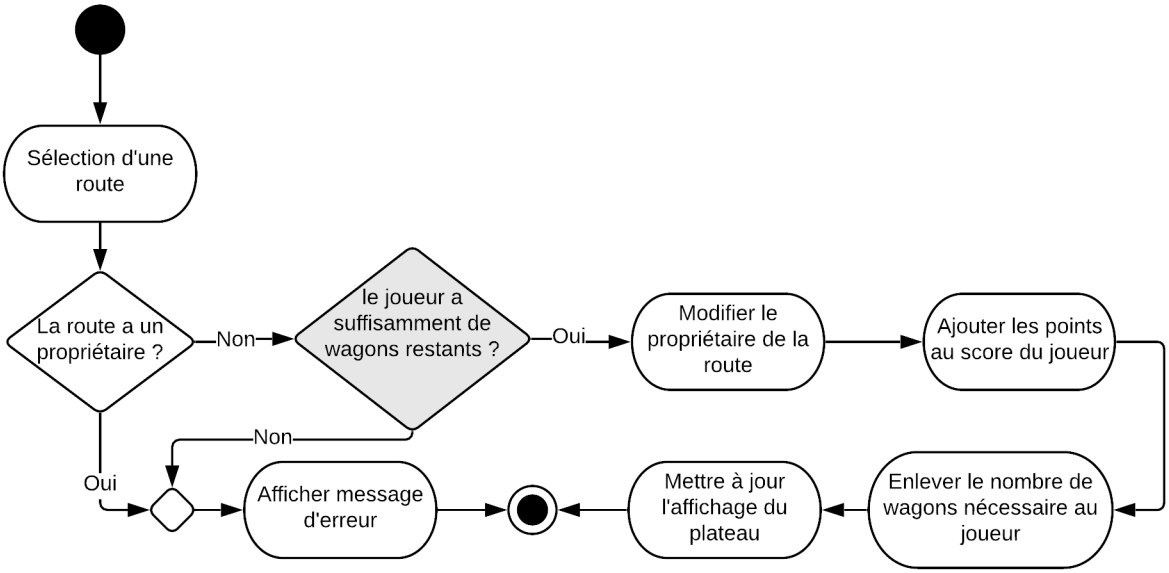

Figure 5: Algorithme mis à jour de la sélection d'une route par le joueur courant

Travail à faire

Compléter la méthode `selectionRoute()` suivant le diagramme d'activité fourni.

3.3.5 Valider l'implémentation de la méthode `selectionRoute()`

Travail à faire

Réfléchir aux tests de validation devant être réalisés afin d'avoir confiance en l'application et remplir le cahier de tests.
L'implémentation des tests de validation peut être réalisée dans les extensions.
Compléter le cahier de test avec les résultats de l'exécution des tests de validation.

3.3.6 Détection du dernier tour de jeu avant la fin de la partie

Dans les règles du jeu des ADR, lorsqu'il reste moins de trois wagons à un joueur, la fin de partie démarre : chaque joueur (dont celui qui a moins de trois wagons) dispose d'un dernier tour de jeu avant le décompte des points.

Dans l'implémentation du jeu des ADR, le dernier tour de jeu est détecté par le système, puis le dernier tour de jeu est simulé au moyen d'un compteur. Ce compteur est initialisé au nombre de joueurs de la partie et décrémenté à chaque action du joueur courant. Lorsque ce compteur atteint la valeur 0, la partie est considérée comme terminée.

Détection du dernier tour de jeu

La détection du dernier tour de jeu pour l'ensemble des joueurs s'appuie sur le nombre de wagons restants que le joueur courant possède.

La méthode `detecteDernierTour()` vérifie le nombre de wagons du joueur courant et stocke le résultat (booléen) dans l'attribut `dernierTour` (voir incrément 2) de la classe **Jeu**.

Travail à faire

Implémenter la méthode `detecteDernierTour()` dans la classe Jeu (voir figure 3).
ATTENTION l'affectation de l'attribut `dernierTour` doit prendre en compte le cas suivant : si le dernier tour a été détecté pour un joueur précédent (`dernierTour` à « vrai »), et si le nombre de wagons du joueur courant est supérieur ou égal à 3, la valeur de `dernierTour` ne doit pas repasser à « faux » !
ASTUCE l'utilisation du « bon » opérateur simplifie l'implémentation.

3.3.7 Valider l'implémentation de la méthode detecteDernierTour()

Travail à faire

L'implémentation des tests de validation peut-être réalisée dans les extensions.
Réfléchir aux tests de validation devant être réalisés afin d'avoir confiance en votre application et remplir le cahier de tests.
Compléter le cahier de test avec les résultats de l'exécution des tests.

3.3.8 Gestion du dernier tour de jeu avant la fin de la partie

La détection du dernier tour de jeu est étroitement liée à la « perte » de wagons par le joueur. Cette « perte » est réalisée lorsque le joueur devient propriétaire d'une route. Il semble de fait logique que la détection du dernier tour soit réalisée lors de la sélection d'une route, après que le nombre de wagons du joueur courant ait été modifié.

Travail à faire

Modifier la méthode `selectionRoute()` pour appeler la méthode `detecteDernierTour()` lorsque le joueur devient propriétaire de la route sélectionnée.

Version naïve du dernier tour de jeu

Lors du dernier tour de jeu, chaque joueur peut réaliser une dernière action. Pour simuler ce comportement, on utilise un compteur qui est initialisé lorsque le dernier tour de jeu est détecté. Après l'appel de la méthode `tourDeJeu()`, c'est-à-dire après que le joueur courant ait joué, on décrémente le compteur. Lorsque le compteur atteint la valeur 0, tous les joueurs ont réalisé leur dernier tour de jeu et la partie s'arrête.

Travail à faire

Modifier la méthode main() pour implémenter la version naïve du dernier tour de jeu :

1. Déclarer une variable entière qui représente le nombre de joueurs restants à jouer et lui affecter le nombre de joueurs de la partie.
2. Déclarer une seconde itération **do** ... **while**() dans laquelle on exécute un tour de jeu et on décrémente le nombre de joueurs restants à jouer. L'itération se termine lorsque le nombre de joueurs restants à jouer est inférieur ou égal à 0.

La méthode `testeJeuActionsDernierTour()` de la classe **TestJeuDernierTour** est fournie pour valider l'implémentation de la version naïve du dernier tour de jeu. Cette méthode est un test automatique : l'utilisateur réalise des actions sur l'interface et la méthode vérifie automatiquement les sorties observables du programme.

Travail à faire

Exécuter plusieurs fois la classe **TestJeuDernierTour** avec des actions utilisateurs différentes et observer les résultats.
Compléter le cahier de test avec les résultats de l'exécution des tests.

Version corrigée du dernier tour de jeu

Dans cette implémentation, un joueur peut effectuer une action (sélection d'une route) sans que son tour se termine automatiquement : le joueur effectue alors une seconde action pour terminer son tour et passer au joueur suivant. La version naïve peut, dans certains cas, empêcher un joueur de jouer son dernier tour, en fonction des actions des autres joueurs.

L'origine du problème vient de la gestion du nombre de joueurs restants à jouer : l'implémentation naïve décrémente le compteur à la fois lorsque le joueur sélectionne une route et lorsque le joueur passe son tour.

Pour corriger ce problème, on considère que la fin du tour n'est pas une action du joueur (dans le jeu original, un joueur ne peut pas « passer son tour » !). Le nombre de joueurs restants à jouer ne doit alors être décrémenté que lors de la sélection d'une route.

Travail à faire

Modifier la version naïve du dernier tour de jeu :

1. Supprimer le compteur déclaré dans la méthode `main()`.
2. Supprimer la seconde itération dans la méthode `main()`.
3. Modifier la méthode `tourDeJeu()` : lorsque le tour de jeu courant est le dernier tour, affecter le compteur `joueursRestantsAJouer` au nombre de joueurs de la partie.
4. Modifier la méthode `demandeAction()` : lors de la sélection d'une route, si le dernier tour de jeu est détecté, décrémenter le compteur.

Travail à faire

Exécuter plusieurs fois la classe **TestJeuDernierTour** avec des actions utilisateurs différentes et observer les résultats.
Compléter le cahier de test avec les résultats de l'exécution des tests.

4 Analyse du code ADR

Pour faire la synthèse sur la gestion du tour de jeu et la gestion d'une partie des ADR, il est demandé de fournir une analyse et une conception préliminaire de la classe **Jeu**. Le travail sera principalement orienté sur le fonctionnement qui peut être décrit avec les diagrammes UML vus en cours.

Travail à faire

Proposer un diagramme d'activités, similaire à la figure 5 pour modéliser le fonctionnement des méthodes `tourDeJeu()` et `main()` de la classe **Jeu**.

5 Fin de l'incrément 4

En arrivant à la fin de ce document, votre jeu modifie les informations du joueur lorsque celui-ci prend possession d'une route. La mécanique principale du jeu, notamment la détection de fin de partie est maintenant effective.

En fonction du temps qu'il vous reste avant l'incrément suivant, vous pouvez :

- Compléter et améliorer les documents de modélisation et de vérification/validation
  - Mettre à jour le ou les diagrammes de classes avec les modifications apportées aux classes dans cet incrément.
  - Compléter le cahier de test à partir des données de test identifiées.
- Travailler sur l'annuaire (en auto-évaluation).
- Réaliser les extensions de l'incrément 4 (optionnel - section 6)

6 Extensions

6.1 Nombre d'actions par tour

Dans la version actuelle du jeu des ADR, le joueur peut « passer son tour » à n'importe quel moment. Ce comportement de jeu n'est pas prévu dans les règles du jeu des ADR : le joueur ne termine son tour que lorsqu'une action de jeu est réalisée.

Travail à faire

Proposer une solution pour contraindre le joueur à réaliser une action de jeu avant de « passer son tour ».

6.2 Tests de validation

Utiliser le cahier de test et essayer d'implémenter les tests de validation (semi-automatiques ou manuels) dans le jeu.

Travail à faire

Compléter et exécuter la classe **TestJeuSelectionRoute** pour valider l'implémentation des modifications apportées à la classe **Jeu** concernant la prise de possession d'une route.
Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez-les avant de poursuivre.

Travail à faire

Compléter et exécuter la classe **TestJeuDernierTour** pour valider l'implémentation des modifications apportées à la classe **Jeu** et valider la détection du dernier tour de jeu.
Si vous rencontrez des erreurs pendant la compilation ou l'exécution, corrigez-les avant de poursuivre.