Université de Paris

# PROJET – PI4

Groupe ED2b

## AVENTURIERS DU RAIL

Dans Les Aventuriers du Rail, les joueurs se lancent dans une aventure palpitante à travers différents pays et continents, cherchant à relier des villes entre elles en construisant des chemins de fer. Chaque joueur doit planifier soigneusement son itinéraire et choisir judicieusement ses cartes de wagons pour construire ses voies ferrées tout en bloquant les routes de ses adversaires. L'accent est mis sur la stratégie, la gestion des ressources et la prise de décisions tactiques pour optimiser ses routes et marquer le plus de points possibles. Chaque partie est une course effrénée pour dominer les voies ferrées et conquérir le terrain.

MELART Arthur, MERCIER Christophe

# Table des matières :

# I. Présentation du sujet

## a. Contexte

Les Aventuriers du Rail est un jeu de plateau qui consiste à placer des wagons sur un réseau ferré dans le but de relier des villes entre elles. Au début du jeu, chaque joueur reçoit des objectifs à remplir. Chaque objectif consiste en une paire de villes que le joueur devra connecter à l'aide du réseau qu'il va former en occupant des portions de voie ferrée avec ses wagons.

Chaque objectif apportera des points ou des malus en fin de partie, suivant qu'il est atteint ou non : plus les destinations d'un objectif sont éloignées l'une de l'autre, plus elles apportent ou enlèvent de points ! Pour pouvoir poser des wagons sur une voie, il faut avoir le bon nombre de cartes wagons de la couleur de cette voie. Par exemple, si on veut poser des wagons sur une route bleue de longueur 5, il faudra avoir en main 5 cartes wagons bleues.

À chaque tour, le joueur peut choisir de :

- Prendre des cartes wagons : il peut choisir deux cartes wagons visibles sur la table ou cachées dans la pioche, ou alors, si disponible, une carte locomotive, qui sert de joker et remplace un wagon de n'importe quelle couleur.
- Occuper une voie : s'il a les cartes nécessaires, il peut les défausser pour occuper une voie en y posant ses wagons.
- Pêcher de nouveaux objectifs.

La partie se termine quand un joueur a moins de 2 wagons dans la réserve. Dans ce cas, chacun, y compris celui ou celle qui a mis fin à la partie, joue une dernière fois, puis on compte les points. La personne qui a le plus de points gagne ! Pour compter les points :

- Points pour chaque route : chaque voie occupée apporte des points, respectivement 1, 2, 4, 7, 10, 15 pour des voies de longueur 1, 2, 3, 4, 5, 6.
- Points d'objectif : chaque objectif réussi (ou pas !) apporte (ou enlève !) un nombre de points égal à la longueur du plus court chemin entre les destinations (qui n'est pas forcément le plus court chemin dans le réseau du joueur).
- Chemin le plus long : la personne qui a construit le chemin le plus long reçoit 10 points supplémentaires.

## b. Mise en œuvre

- Construction du Réseau : Pour ce projet, nous devons implémenter un plateau de jeu en modélisant un réseau de villes et de voies ferrées sous forme de graphe. Le réseau peut être inspiré de l'une des versions du jeu ou être complètement inventé.

- Implémentation d'une Partie avec un Joueur : Nous commencerons par programmer une partie avec un seul joueur. Le joueur aura un nombre maximal de tours pour jouer, au terme desquels nous compterons les points.

- Calcul du Plus Long Chemin : Un des objectifs du projet est de calculer le plus long chemin dans le réseau du joueur. L'algorithme utilisé pour cette tâche sera détaillé dans le rapport.

- Interface Graphique : Enfin, nous créerons une interface graphique pour le jeu en utilisant la librairie Swing de Java. Cette interface permettra au joueur d'interagir avec le jeu de manière intuitive et visuelle.

- Ce projet permettra de mettre en pratique diverses compétences en programmation et en algorithmique, tout en créant une application ludique et intéressante.

- Ce projet doit être abouti et sa version finale délivrée après 12 semaines.

# II. Implémentations

## a. Présentation du modèle

Pour organiser notre développement de manière structurée et modulaire, nous avons choisi d'utiliser le modèle MVC (Model-View-Controller). Ce modèle permet de séparer clairement les différentes composantes de l'application, facilitant ainsi la maintenance et l'évolution du code.

Model (Modèle) : Le modèle représente la logique métier et les données du jeu. Dans notre projet, cela inclut les classes qui gèrent le réseau de villes, les voies ferrées, les cartes wagons et les objectifs. Le modèle est responsable de la gestion des règles du jeu, du calcul des points et de la validation des actions du joueur.

View (Vue) : La vue est chargée de l'affichage graphique et de l'interface utilisateur. Dans notre projet, elle utilise la librairie Swing de Java pour représenter le plateau de jeu. La vue se met à jour en fonction des changements dans le modèle pour refléter l'état actuel du jeu.

Controller (Contrôleur) : Le contrôleur gère les joueurs et le modèle. Il interprète les actions de l'utilisateur, comme le choix de cartes ou l'occupation de voies, et met à jour le modèle en conséquence. Le contrôleur assure la liaison entre le modèle et la vue, garantissant que toute modification dans le modèle est correctement affichée dans la vue.

## b. Jeu de base

'Plateau' (model) : Cette classe comprend : une liste des joueurs, une pioche pour les cartes Wagon, une pioche pour les cartes Objectif, une liste de toutes les villes du plateau, une liste de toutes les lignes composant le plateau/ Avoir des listes de toutes les lignes et de toutes les villes permet un accès rapide et efficace à ces éléments pendant le jeu. De plus, la classe Plateau gère l'ordre des joueurs.

'Ville' (model) : 'Ville' contient le nom, les coordonnées graphiques, la liste des lignes qui la relient, un champ Joueur pour le propriétaire de la gare, et un champ Ligne pour la ligne reliant à la gare. Cette structure permet de vérifier les objectifs à la fin de la partie grâce à un parcours en largeur.

'Ligne' (model) : Elles correspondant aux arêtes de notre graphe, sont représentées par la classe 'Ligne'. Cette classe contient deux champs, 'ville1' et 'ville2', initialisés de manière que ville1 soit toujours la ville la plus proche de (0,0) en triant d'abord par les coordonnées Y. Elle inclut également un champ Couleur représentant la couleur de la ligne, un entier 'nmbCases' pour le nombre de cases de la ligne, un champ double pour l'angle de la ligne afin d'une meilleure représentation graphique, ainsi qu'un booléen.

'CarteWagon' (model) : Les cartes Wagon sont représentées par la classe 'CarteWagon', qui contient simplement un champ 'Couleur', indiquant la couleur de la carte pour faciliter son affichage.

'CarteObjectif' (model) : La classe 'CarteObjectif' représente une carte objectif avec deux villes, le succès de l'objectif et le nombre de points associés.

‘Couleur’ (model) : elle est définie comme une énumération (‘enum’), est utilisée pour représenter les couleurs dans le jeu. Son objectif est d'assurer une référence uniforme des couleurs pour les cartes et les lignes.

‘PiocheWagon’ et PiocheObjectif’ (model) : Elles sont composées d'un champ de type ‘Queue‘ de Java.util pour avoir un comportement similaire à celui d'une pioche dans le jeu d'origine.

La pioche Wagon contient également un champ ‘cartesVisibles’ qui correspond aux cinq premières cartes de la pioche, celles parmi lesquelles le joueur peut piocher. Cet attribut est représenté par un tableau pour ne pas avoir à modifier l’ordre de tous les éléments à chaque ajout ou suppression.

‘Jeu’ (model) : La classe ‘Jeu’ sert à vérifier les conditions de la partie, telles que s'assurer qu'aucun joueur n'a moins de 5 wagons ou que le temps maximal de la partie n'est pas écoulé. Si c'est le cas, elle permet aux joueurs de sélectionner toutes leurs lignes de gare avant le décompte final des points. Cette classe est lancée dans un thread séparé des autres pour ne pas bloquer les interactions entre le joueur et l'interface graphique. Elle se charge également de mettre à jour le temps écoulé de la partie ainsi que le nombre de tours écoulés, ce qui permet de fluidifier au maximum les interactions entre le joueur et l'interface graphique, sans ralentir cette dernière.

‘Joueur’ (controller) : Elle est utilisée pour recréer les joueurs. Elle contient toutes les informations spécifiques à un joueur, telles que son pseudo, le nombre de cartes wagon qu'il possède, le nombre de petits wagons qu'il lui reste pour acheter des lignes, ainsi que sa couleur et son avatar choisi lors de la sélection de pseudonyme. De plus, un champ ‘Ville’ appelé ‘click’ stocke la première ville sur laquelle le joueur a cliqué lors de son tour. Ces informations sont stockées dans la classe pour faciliter l'accès lors de l'affichage graphique et pour permettre des modifications plus simples en fonction des actions du joueur. Cela garantit une efficacité constante lors de chaque modification.

‘PanelJeu’ (view) : Gère l'affichage du plateau de jeu, des lignes et des villes qui le composent. La méthode ‘draw’ parcourt toutes les villes pour les dessiner, puis dessine toutes les lignes. Il gère également toutes les interactions, telles que cliquer sur le bouton "Pioche", ou sur "Inventaire" pour afficher ces derniers. De plus, le joueur peut cliquer directement sur les villes représentées par des cercles pour placer une gare ou acheter une ligne. Lorsque le joueur a terminé son tour, il lui suffit de cliquer sur "Next Joueur" pour passer la main au joueur suivant. Cela permet au joueur, même s'il ne peut plus rien faire, de continuer à consulter son inventaire et de se préparer pour le prochain tour.

‘PanelInventaire’ (view) : Le panel Inventaire affiche les cartes en main du joueur, avec le nombre de cartes de chaque couleur, ainsi que toutes les cartes objectifs qu'il possède. Son objectif est de permettre au joueur d'avoir une vue d'ensemble sur ses possibilités et ce qu'il souhaite faire pour remporter la partie.

‘PanelPioche’ (view) : Le panel Pioche comporte quatre cartes face visibles représentant les différentes de cartes que le joueur peut piocher. Il contient également une pioche de cartes wagon où le joueur peut piocher s'il ne souhaite aucune des cartes visibles. De plus, il inclut une pioche de cartes objectif où le joueur à le choix entre trois cartes, sachant qu’il devra en conserver au minimum une parmi les trois.

‘EcranPresentation’ (view) : Cette classe, étant un JFrame, sert à lancer le jeu. Elle offre également la possibilité de reprendre une partie sauvegardée, de charger une partie précédemment enregistrée ou de choisir de jouer en ligne.

‘SelectionJoueur’ (view) : Elle permet au différent joueur de choisir leurs avatars et leur pseudo pour la partie.

‘PanelSelection’ (view) : Ce panel permet de choisir entre quatre modes de jeu (classique, nombre de tour limité, temps limité pour la partie et temps limité par tour). Ce panel est également utilisé pour déterminer l'ordre des joueurs dans la partie, en commençant par celui qui obtient le plus haut lancer de dés en utilisant ‘DicePanel’.

### c. Extensions

‘DoubleLigne’ (model) : Les double lignes sont une extension du jeu de base où, au lieu d'avoir une seule ligne entre deux villes, il peut y en avoir deux. Une spécificité des doubles lignes est qu'un joueur ne peut pas posséder les deux lignes d'une double ligne. Dans notre implémentation, les doubles lignes sont représentées par une classe étendue de la classe ‘Ligne’, avec un champ supplémentaire pour l'autre ligne. Ainsi, lorsqu'on vérifie si le joueur peut acheter la ligne, on vérifie d'abord si le joueur peut acheter la ligne actuelle, puis on vérifie également s'il n'a pas déjà acheté l'autre ligne de la double ligne.

Tunnels : Les tunnels sont implémentés par un booléen dans la classe ‘Ligne’. ‘estTunnel’, indique au plateau qu'il s'agit de tunnels. Ils ont une apparence différente, un contour noir, pour les distinguer des autres lignes. Lorsque le joueur tente d'acheter une ligne tunnel, le jeu tire trois cartes au hasard et demande au joueur de payer pour acquérir la ligne.

Ferries : Les ferries sont des cases qui nécessitent des cartes locomotives en plus des cartes normales pour être acquises. Ils sont représentés par un champ entier ‘nmbLocomotives’ qui indique le nombre de cartes locomotives nécessaires pour les acheter. Graphiquement, ils sont représentés par un contour rouge pour les distinguer des cartes de base.

Sauvegarde : Il est possible de sauvegarder une partie en cours et de reprendre une partie sauvegardée depuis ‘EcranPresentation’.

‘Serveur’ (Internet) : Le serveur fonctionne de manière qu'un PC lance le jeu en mode création de partie. L'utilisateur doit alors sélectionner le nombre de joueurs dans la partie ainsi que le port sur lequel il souhaite envoyer les requêtes. Le serveur attend ensuite que tous les clients des joueurs se connectent avant de lancer la partie.

Pour chaque connexion acceptée, le serveur crée un nouveau ‘ConnectionHandler’. L'utilisateur doit alors choisir le mode de jeu souhaité sur le PC hébergeant le serveur. Ensuite, le serveur récupère le pseudonyme de chaque joueur et envoie à tous les clients le mode de jeu sélectionné avec le plateau.

Le serveur envoie également à tous les joueurs un signal pour leur permettre de jouer si c'est leur tour. Il attend alors à chaque fois le nouveau plateau après le coup du joueur, met à jour le plateau du serveur, l'envoie à tous les autres joueurs et permet au prochain joueur de jouer dans la liste.

‘ConnectionHandler’ (Internet) : Chaque ‘ConnectionHandler’ sert à recevoir tous les paquets envoyés par les clients des utilisateurs et les stocke jusqu'à ce que le serveur vienne les consulter. Chaque ‘ConnectionHandler’ est lancé sur son propre thread pour ne pas bloquer le serveur sur un ‘ConnectionHandler’ donné et faciliter la gestion des déconnexions. L'intérêt de cette approche est que si un joueur se déconnecte, le ‘ConnectionHandler’ correspondant reçoit un signal d'avertissement et se retire du serveur, permettant ainsi au serveur de ne pas perdre de temps à surveiller ce qui se passe. Cela permet également une gestion individualisée, ce qui rendrait assez facile l'ajout de fonctionnalités telles qu'un système anti-triche.

‘Client’ (Internet) : Le client est créé chaque fois qu'un joueur décide de jouer en ligne. Il est lancé dans un autre thread et est responsable d'envoyer et de recevoir toutes les données du joueur. Il enverra le nouveau plateau au ‘ConnectionHandler’ correspondant lorsqu'un joueur a joué un coup. Pour la réception des informations, le client est également lancé sur un autre thread afin d'éviter de bloquer le système.

# III. Méthodologie et organisation

## a. Organisation initiale

Nous avons choisi d'adopter des sprints d'une semaine pour notre organisation, en nous inspirant de la méthode 'Scrum'. Au départ, notre équipe comptait quatre membres, comprenant Christophe Mercier, Arthur Melart, Leo Piral, Melisende Lemaresquier. Cependant, Melisende était absente les trois premières semaines, donc aucune tâche ne lui était attribuée. Chacun s'était vu attribuer des responsabilités spécifiques : Arthur pour l'interface graphique, Christophe pour le modèle, et Léo en tant que polyvalent, apte à soutenir l'équipe dans les domaines requis. Cependant, au fil du temps, les circonstances ont évolué. Léo s'est retrouvé à travailler sur une même tâche pendant plusieurs semaines, entraînant une réaffectation de certaines responsabilités pour garantir la progression du projet. Arthur a dû reprendre une partie du travail de Léo, tandis que Christophe s'est focalisé sur le modèle, menant des échanges réguliers sur les choix d'implémentation.

## b. Difficultés et ajustements

Malgré notre planification initiale, nous avons rencontré des difficultés en termes de participation. Durant les trois premières semaines, l'absence d'un membre pour des raisons personnelles légitimes a été notable. À la fin de cette période, il est devenu évident que cette personne ne pourrait pas participer au projet du tout. En outre, à la septième semaine, Leo Piral a quitté le groupe, estimant qu'il ne pouvait pas contribuer efficacement. Bien qu'il ait produit quelques lignes de code, celles-ci ont dû être réécrites pour correspondre davantage aux attentes du projet. Face à ces imprévus, nous avons dû revoir la répartition des tâches et nous adapter pour mener à bien le projet avec les membres restants.

# IV. Problèmes rencontrés

Travailler à deux sur un projet conçu pour quatre ou cinq personnes a été particulièrement difficile. Nous avons dû assumer les tâches de plusieurs personnes chacun, ce qui nous a obligés à terminer chaque tâche sans pouvoir nous attarder, afin de ne pas ralentir notre progression.

Pour le jeu de base, une difficulté majeure a été le calcul des points en fin de partie pour déterminer le classement. Il fallait vérifier, pour chaque carte objectif, qu'un chemin existait entre deux villes possédées par le joueur, ce qui a été réalisé grâce à un parcours en profondeur.

Un autre défi a été d'afficher correctement les wagons sur les cases, quelle que soit leur orientation et leur angle, tout en corrigeant les valeurs.

Enfin, une dernière difficulté a été d'apprendre la gestion d'Internet en Java et de faire en sorte que le jeu puisse envoyer toutes les informations des joueurs tout en maintenant son fonctionnement normal. Cela incluait la correction des erreurs de transmission de paquets et la gestion des déconnexions des joueurs pour permettre à la partie de continuer.

# V. Perspectives d'évolution du projet

IA : Ajouter des IA contre qui le/les joueurs pourraient se confronter.

Génération de graph : Pouvoir générer une carte pseudo aléatoire sur laquelle les joueurs pourraient jouer.

Déconnexion : Faire terminer la partie pour chaque joueur quand le serveur se déconnecte.