Retour d'Expérience du Projet de Simulation d'Ascenseur

Valentin PLATON

22 novembre 2025

Résumé

Ce document récapitule l'expérience de développement du projet de simulation d'ascenseur, en se concentrant sur les contributions aux parties **Configuration** et **Front-end (Vue)**. Il détaille la conception des classes, l'implémentation des animations JavaFX, et les méthodes clés pour la gestion de l'affichage et de la dynamique de la simulation.

---

1 Mon Rôle et Contributions (Configuration & Front-end)

Mon travail s'est concentré sur la mise en place de l'environnement, la gestion de la configuration, et la création de l'interface utilisateur graphique (le Front-end) pour visualiser la simulation.

1.1 Partie Configuration (`Configuration.java`)

J'ai mis en place le système de chargement et de parsing des fichiers de configuration (JSON).

- **Méthode `load(String path)`** : Charge le contenu d'un fichier de configuration spécifié par son chemin. Utilise un `Scanner` pour lire le contenu et le retourne sous forme de chaîne de caractères unique.
- **Méthode `parse(String path, Class<?> record)`** : Lit le fichier JSON en utilisant `load(path)` et désérialise le contenu en un objet Java du type spécifié (`record`) à l'aide de la bibliothèque **Gson**.
- **Méthode `formatAll(Enum<?>[] ListConfig)`** : Point d'entrée centralisé pour l'initialisation de la configuration. Elle itère sur la liste d'énumérations (`RecordList`) pour charger, parser et rassembler toutes les données de configuration (`SimulationConfig`, `TowerConfig`, `LiftConfig`, `HabitudeConfig`) dans un objet `ConfigurationRecord`.

---

2 Partie Front-end (Vue et Rendu)

J'ai implémenté le Front-end en JavaFX pour l'affichage de la tour et la gestion des animations.

2.1 Classes de Composants et Vues (Views)

Ces classes modélisent les éléments graphiques et gèrent leurs propriétés fondamentales.

- `FloorView.java` : Représente visuellement un étage (une simple `Rectangle`).
- `ElevatorView.java` : Représente la cabine d'ascenseur.
  - **Méthode `moveToY(double y)`** : Crée et lance une `TranslateTransition` pour déplacer l'ascenseur verticalement jusqu'à une coordonnée Y cible sur une durée de 1 seconde. Elle retourne la transition.
- `PersonView.java` : Représente une personne (un `Circle`) avec son étage actuel (`floor`) et sa destination (`floorDest`).
  - **Méthode `moveTo(double x, double y)`** : Crée et lance une `TranslateTransition` pour déplacer la personne vers des coordonnées X et Y spécifiques sur 1 seconde. Elle retourne la transition.
- `TowerView.java` : Conteneur principal de la scène (`Pane`). Il initialise le décor, crée les étages, les deux ascenseurs, et gère les contrôles utilisateurs (boutons, champs de texte).

2.2 Rendu et Orchestration (`TowerRenderer.java`)

Cette classe sert de couche intermédiaire pour orchestrer les mouvements et synchroniser les éléments graphiques.

- **Gestion des Mouvements Simples**
  - `moveElevator(int id, int targetFloor)` : Appelle `moveToY` de l'ascenseur ciblé.
  - `movePersonTo(int id, int targetFloor, double targetX)` : Calcule la coordonnée Y de l'étage cible et appelle `moveTo` sur la personne.
  - `moveElevatorToFloor(int targetFloor)` : Déplace l'ascenseur à l'étage cible et met à jour l'état interne de l'ascenseur (`currentElevatorFloor`).
- **Gestion des Séquences et Synchronisation (Parallèle)**
  - `fullUp` / `fullDown` / `move` : Ces méthodes gèrent le déplacement complet d'un groupe de personnes et de l'ascenseur, en utilisant `ParallelTransition` pour synchroniser les animations.
    1. Phase d'Attente / Embarquement : Les personnes se déplacent vers l'ascenseur.
    2. Mouvement Vertical (Parallèle) : L'ascenseur (`moveElevatorToFloor`) et toutes les personnes embarquées (`movePersonTo`) sont déplacés simultanément à l'étage de destination en utilisant une `ParallelTransition`.
    3. Phase de Débarquement : L'animation se termine, puis les personnes sont déplacées hors de l'ascenseur (appel à `exitLift`).
  - `waitLift(List<Integer> ids, int floor)` : Déplace un groupe de personnes vers une position de file d'attente sur un étage, en les décalant en X.
  - `exitLift(List<Integer> ids, int floor)` : Déplace un groupe de personnes vers une position de sortie (à droite de la tour), en les décalant en X pour éviter la superposition.
- **Logique du Bouton "Up" (Dans `TowerView`)** : Une logique complexe a été implémentée pour gérer les appels multiples. Elle trie les personnes sélectionnées du plus haut étage au plus bas et utilise une séquence récursive de `Runnable` pour enchaîner les étapes suivantes :
  1. Aller chercher la personne qui attend à l'étage le plus élevé non desservi (Mouvement de l'ascenseur et des personnes déjà à bord).
  2. Embarquement de la nouvelle personne.
  3. Répéter (1) pour l'étage le plus élevé restant, jusqu'à ce que toutes les personnes soient à bord.
  4. Déplacer l'ascenseur et les personnes jusqu'à l'étage de destination (`destFloor`) en parallèle.
  5. Débarquement.