Elevator Simulator

Manuel utilisateur

Projet Java — ESIEE

19 novembre 2025

1 Introduction

Ce projet simule le fonctionnement d'ascenseurs dans une tour d'habitation.
L'utilisateur peut :
— configurer le bâtiment, les résidents et les ascenseurs via un fichier JSON ;
— choisir une stratégie de contrôle des ascenseurs (FCFS, nearest) ;
— lancer une simulation d'une journée ;
— consulter des métriques (temps d'attente, temps de trajet, énergie, occupation) ;
— analyser des rapports JSON (global, par ascenseur, par résident).
Ce document explique comment installer, lancer et utiliser le simulateur.

2 Installation

Prérequis

— Java 17 ou supérieur installé (`java -version`) ;
— Maven 3 ou supérieur (`mvn -v`) ;
— Git recommandé pour récupérer le dépôt.

Récupération du projet

```
git clone https://github.com/...../elevator-simulator.git
cd elevator-simulator
```

Compilation

Depuis la racine du projet :

```
mvn clean package -DskipTests
```

Cette commande génère un JAR exécutable dans `target/`, de type :

```
target/elevator-simulator-1.0-SNAPSHOT-jar-with-dependencies.jar
```

C'est ce fichier qu'il faut utiliser pour lancer le simulateur.

3 Lancement de la simulation

Le programme se lance avec `java -jar` :

```
java -jar target/elevator-simulator-1.0-SNAPSHOT-jar-with-dependencies.jar [options]
```

Les options principales sont :

— `-config=CHEMIN` : fichier JSON de configuration (par défaut `config/demo-config.json` dans le classpath) ;
— `-strategy=fcfs|nearest` : stratégie d'ascenseur (`nearest` par défaut) ;
— `-report=CHEMIN` : chemin du rapport JSON global, par exemple `target/reports/demo-report.json` ;
— `-interactive` : lance l'interface utilisateur en mode menu.

Exemples

Exécution simple (non interactive)

```
java -jar target/...-jar-with-dependencies.jar \
  --config=config/demo-config.json \
  --strategy=nearest \
  --report=target/reports/demo-report.json
```

Le programme :

1. charge la configuration ;
2. exécute la simulation ;
3. affiche un résumé dans la console ;
4. écrit plusieurs fichiers JSON dans `target/reports/`.

Mode interactif

```
java -jar target/...-jar-with-dependencies.jar --interactive
```

Ce mode affiche des menus pour modifier la configuration en mémoire avant de lancer la simulation.

4 Interface utilisateur (mode interactif)

En mode `-interactive`, un menu texte s'affiche.

Menu principal

Exemple de menu principal :

```
=== Elevator Simulator 1.1 ===
Stratégie : nearest
Rapport   : target/reports/demo-report.json

Menu principal
1) Résumé
2) Ascenseurs
3) Stratégie
4) Changer chemin rapport
5) Lancer simulation
0) Quitter
```

— **1) Résumé** : affiche les informations de base (nombre d'étages, nombre de résidents, liste des ascenseurs).
— **2) Ascenseurs** : ouvre le sous-menu de gestion des ascenseurs.
— **3) Stratégie** : permet de choisir `fcfs` ou `nearest`.

— **4) Changer chemin rapport** : modifie la destination du JSON global.
— **5) Lancer simulation** : exécute la simulation avec les paramètres courants.
— **0) Quitter** : termine le programme.

Un historique des dernières actions et un message d'état sont affichés en haut de l'écran pour garder le fil des opérations.

Sous-menu Ascenseurs

Le sous-menu « Ascenseurs » permet :
— de lister les ascenseurs existants (id, capacité, vitesse, etc.) ;
— d'ajouter un ascenseur (avec un identifiant libre) ;
— de modifier un ascenseur existant (capacité, vitesse, temps d'ouverture des portes) ;
— de supprimer un ascenseur ;
— d'annuler la dernière modification (undo simple) ;
— de revenir au menu principal (`9`).

Les entrées utilisateur sont vérifiées :
— en cas d'identifiant déjà utilisé, l'ajout est refusé ;
— l'utilisateur doit confirmer une suppression ;
— les valeurs numériques sont contrôlées (entiers positifs, etc.).

Sous-menu Stratégie

Ce sous-menu permet de choisir la stratégie de contrôle des ascenseurs :
— `fcfs` : First-Come, First-Served ;
— `nearest` : l'ascenseur cible la requête la plus proche.

Le choix est appliqué immédiatement pour les simulations suivantes.

5 Configuration JSON

Les fichiers de configuration sont dans `src/main/resources/config/`. Un exemple typique est `config/demo-config.json`.

Ce fichier décrit :
— le bâtiment (nombre d'étages, hauteur d'étage) ;
— les résidents (nombre par étage, habitudes de déplacement) ;
— la liste des ascenseurs (id, capacité, paramètres physiques) ;
— les paramètres de simulation (durée de la journée, pas de temps, graine aléatoire).

Pour tester d'autres scénarios, il suffit de :

1. créer un nouveau fichier JSON dans `src/main/resources/config/` ;
2. lancer le programme avec l'option `-config=...`.

6 Rapports générés

Après une simulation, plusieurs fichiers JSON sont écrits (d'après le chemin fourni par `-report`) :

— **Rapport global**
`<report>.json`
Contient :
— les statistiques globales (temps d'attente moyen, médian, max, etc.) ;
— les temps de trajet ;
— les temps d'attente moyens par étage d'origine ;
— l'énergie totale et l'énergie moyenne par requête ;

  - le taux moyen d'occupation des cabines ;
  - la liste détaillée de toutes les requêtes.
- **Rapport des arrêts par ascenseur**
  `<report>-elevators.json`
  Pour chaque ascenseur :
  - liste des arrêts (heure, étage) ;
  - nombre de passagers montés / descendus ;
  - passagers restants après l'arrêt.
- **Rapport par résident**
  `<report>-residents.json`
  Pour chaque résident :
  - étage de domicile ;
  - historique de ses trajets (origine, destination, horaires, ascenseur utilisé).

Ces rapports peuvent être exploités par des outils externes (scripts, tableurs, etc.) pour produire des graphiques et alimenter l'étude comparative des heuristiques.

7 Problèmes fréquents

- **Config introuvable** : vérifier que le fichier JSON est bien dans `src/main/resources/config/` et que le chemin passé à `-config` correspond (par exemple `config/demo-config.json`).
- **Modifications de config non prises en compte** : recompiler avec `mvn clean package` puis relancer le JAR généré.
- **Erreur Java/Maven non trouvés** : ajouter les binaires de Java et Maven au `PATH` du système.