3D- Java Labs 2024-2025

Catch me if you can !

3D 2024/2025

Résumé du TP :

Ce projet a visé à développer une application interactive permettant de visualiser et d'analyser des données relatives à l'aviation. Il combine des données statiques issues d'une liste d'aéroports au format CSV et des données dynamiques récupérées via l'API **AviationStack** pour afficher des informations en temps réel. La structure repose sur plusieurs classes Java et une interface graphique construite en JavaFX.

Ce que j'ai appris durant ce TP :

1. Gestion de fichiers et manipulation de données :

    - J'ai appris à lire et à traiter un fichier CSV pour en extraire des informations pertinentes sur les aéroports, telles que leurs codes IATA et leurs positions (longitude et latitude). Ces données sont ensuite stockées dans des structures adaptées comme les `ArrayList`.
    - J'ai également pu voir la méthode pour calculer des distances approximatives entre coordonnées GPS pour identifier l'aéroport le plus proche avec un clic en sachant que nous avons des données réelles dans le CSV en 3D alors que nous en réalité nous avons uneprojection plane de la Terre.

2. Création d'interfaces graphiques interactives avec JavaFX :

    - J'ai utilisé des composants graphiques comme `Scene`, `Group` et `PerspectiveCamera` pour afficher une sphère texturée représentant la Terre. Cela a permis de rendre le projet visuellement immersif et d'observer son évolution en temps réel.

3. Multithreading avec Java :

    - Grâce à l'utilisation de threads, j'ai appris à effectuer des tâches en arrière-plan, comme les requêtes API, tout en évitant de bloquer l'interface utilisateur. Cela a permis d'améliorer la fluidité de l'application.

4. Utilisation d'une API externe :

    - J'ai implémenté l'API **AviationStack** pour obtenir des données sur les vols. J'ai appris à formater des requêtes HTTP, analyser les réponses JSON et intégrer les informations récupérées dans l'application.

Méthodes utilisées :

1. Threads et Runnable :

    - Une classe dédiée (YellowBallUpdater) qui permet de gérer l'exécution des requêtes API sur un thread séparé, ce qui a permis de maintenir une fluidité dans l'application tout en effectuant des tâches lourdes en arrière-plan qui sont les requêtes API et tout ce qui vas toucher à l'API.

2. API et parsing JSON :

    - La classe ApiFlightManager envoie des requêtes HTTP, alors que JsonFlightFillerOracle parse les réponses JSON afin de pouvoir une fois récupérer extraire des informations sur les vols, ces informations sont ensuite utilisées pour afficher les aéroports de départ à l'écran sous forme de boules jaunes.

3. JavaFX pour la 3D :

    - La sphère représentant la Terre a été texturée à l'aide de la classe PhongMaterial, et un système de rotation a été implémenté via AnimationTimer pour simuler une rotation réaliste de la Terre

Problèmes rencontrés et solutions

1. Problème d'affichage de la sphère rouge sur la Terre :

    - Au début, j'ai suivi la méthode de calcul donné dans le TP pour l'affichage de la sphère rouge pour signaler un l'aéroport sélectionné cependant cette méthode est incorrecte. Lorsque que l'on cliquait sur l'hémisphère nord, les coordonnées n'étaient pas précises, et cela ne fonctionnait pas du tout sur l'hémisphère sud. Le problème venait d'une erreur dans la conversion latitude-longitude. La solution a été apportée par une méthode corrigée basée sur une projection Mercator.

2. Blocage de l'interface lors des requêtes API :

    - Lors de l'intégration de l'API, l'application se figeait pendant les différentes requêtes. Pour corriger cela, nous avons utilisé un thread afin de pouvoir déporter ces tâches en arrière-plan sur un autre processus afin de permettre de résoudre ce problème.

Conclusion :

Ce projet à été fort intéressant offrant une expérience complète, il nous a permis de combiner des concepts de programmation orientée objet, l'utilisation de javaFX et les APIs, et des techniques telles que le multithreading.Ces compétences ont été essentielles pour ce projet.