Yassine El graoui - Alexandre Raybaut

Ajouts bonus faits au projet et leur mise en oeuvre

Ajout 1 :

Nous avons fait de sorte que lorsque l'on clique sur un avion, l'on reste centré sur ce dernier jusqu'à ce que l'on en sélectionne un autre. Cela résulte au fait que l'on suit le déplacement sur la carte avec l'avion au centre de notre canvas.
Nous avons ajouté à la classe AircraftController un attribut BaseMapController afin de pouvoir manipuler directement le recentrage de l'avion.

Mise en œuvre : Pour faire cela, nous avons placé un listener sur la position de l'avion sélectionné par le clic de la souris. Par conséquent, dès que sa position change, nous faisons un recentrage automatiquement sur sa nouvelle position (si cette dernière n'est pas nulle).

Ajout 2 :

Nous avons ajouté une colonne à notre table qui contient les images les plus récentes selon l'adresse OACI (elle sont récupérées sur le site planeSpotters). Pour cela, nous avons installé la librairie org.json de java pour pouvoir récupérer plus simplement l'image jpg du json envoyé (après avoir fait une requête à l'api de planeSpotters).

Mise en œuvre : On télécharge les images depuis l'API de planeSpotters en utilisant org.json pour afficher l'image de l'avion correspondant à l'adresse OACI. On utilise l'API org.json pour parser le JSON reçu. On utilise l'URL reçue par l'API et on sauvegarde les images dans un dossier "image-storing". Puis, on affiche chaque image dans la colonne Images composée de ImageView (qui sont des nœuds utilisés pour stocker des images).

Ajout 3 :

Nous avons dessiné des cercles, centrés sur le coin haut-gauche du rolex. Ces derniers représentent la distance les séparant de la bibliothèque. En effet, si un avion se situe dans le cercle vert, cela signifie qu'il est à moins de 50km du RLC. La même logique s'applique pour les cercle jaune (100km de rayon) et rouge (200km de rayon).

Mise en œuvre : pour cela, nous avons utilisé les fonctions offertes par le graphic context de la classe Canvas de JavaFx. Plus précisément les méthodes :
- setFill() afin de changer la couleur de dessin du canvas. En utilisant Color.RED pour le rouge par exemple.
- fillOval() afin de pouvoir dessiner les cercles. Ces derniers avaient pour coordonnées haut-gauche les coordonnées du RLC sur le canvas moins le vecteur v(rayon, rayon). Enfin, les dimensions de cet oval étaient les même en largeur et en hauteur pour lui donner une forme circulaire, son rayon étant calculé grâce à la méthode ci-dessous :

Nous calculons la distance que représente un pixel sur la projection Mercator selon un zoom et une latitude précise (cette dernière étant constante car elle correspond à celle du rolex). Cette distance est obtenu par la formule :
*distance pixel = C * cos(latitude) / 2**(zoom + 8)*
où C représente la circonférence de la Terre à l'Équateur (soit environ 40 075,016 686 km). Enfin, nous manipulons cette distance afin de voir combien de pixel il faut mettre de côté pour représenter un certain nombre de kilomètres. Par exemple pour représenter 50km, l'on fait NbPixelsPour50km = 50 / distancePixel. Cela donne le rayon de notre cercle.

- setGlobalApha() permettant de rendre les cercles un peu transparents, le but étant de les voir sans qu'ils gênent les dessins de la carte
Les cercles sont redessinés en même temps que les tuiles pour qu'ils puissent se mouvoir normalement sur la carte.

Sources :
Distance d'un pixel selon la projection Mercator :
https://wiki.openstreetmap.org/wiki/Zoom_levels (section : Distance per pixel math)
PlaneSpotters API utilisé pour la récupération des images:
https://api.planespotters.net/pub/photos/hex/
Librairie JSON installée pour Java :
https://github.com/stleary/JSON-java