Bonus : Le « Flipper »

Le but de ce bonus est de simuler la chute de billes sur le relief numérique utilisé dans le projet. Nous utilisons pour cela la classe Application de JavaFX. Nous transférons toute la génération de carte du projet de base dans une classe Flipper qui hérite d'Application, et qui contient deux méthodes : start, qui génère les éléments nécessaires à la simulation, et addBall, qui ajoute une bille à la simulation selon des paramètres qui définissent une bille.

Pour les billes, nous avons défini une classe Ball, qui contient une taille, une masse, une position, une vitesse, et une accélération, ces trois dernières représentées par des Vector3. Vector3 a par ailleurs été agrémentée de méthodes permettant de faire des addition/multiplications entre Vector3 pour simplifier les calculs. Nous avons aussi modifié les DigitalElevationModel et sa déclinaison HGT pour qu'on puisse obtenir des normales et des altitudes selon des coordonnées x et y.

Voici une description du comportement de la méthode start, qui gère l'application (à noter que notre implémentation est loin d'être optimale ; nous avons dû apprendre le fonctionnement de JavaFX par nous-mêmes et avons avant tout essayé d'obtenir un résultat fonctionnel. Le résultat pourrait bénéficier de nombreuses corrections et améliorations, car nous n'avons malheureusement pas réussi à résoudre tous les problèmes que nous avons rencontrés et avons donc dû parfois un peu « broder ») :

Tout ce qui se passe jusqu'au dessin de l'image est connu. Il s'agit du contenu du Main du projet de base, avec des récupérations d'arguments supplémentaires pour la simulation. Nous arrivons ensuite à un if qui permet de choisir si on veut afficher la carte topographique, ou un relief en niveau de gris (à utiliser si on n'a pas les données OSM nécessaire pour un lieu souhaité, c'est aussi beaucoup plus rapide à générer).

Après ce if, nous convertissons notre BufferedImage en Image JavaFX, afin de l'utiliser comme background pour la simulation. Nous créons ensuite deux Canvas JavaFX dont nous extrayons pour chacun un GraphicsContext : un pour le background, qui reste fixe, et un pour les billes, qui sera effacé puis redessiné à chaque frame.

Nous créons ensuite un Group JavaFX, qui contiendra les deux Canvas (le Canvas JavaFX hérite de la classe Node de JavaFX, qui est un élément de base pour tout élément graphique qu'on utilisera dans une application).

On définit maintenant un EventHandler qui détecte les clics de l'utilisateur sur la fenêtre et qui ajoutera une nouvelle boule de taille et de masse aléatoires à l'endroit du clic. On associe directement cet EventHandler au Group « root ». Il ne manque plus que la taille et le nom de la fenêtre, puis on peut faire un show() qui lancera l'application. À savoir qu'une application JavaFX ne s'instancie pas ; il faut faire Flipper.launch(Flipper.class, args) avec args étant les arguments du Main dans notre cas.

La dernière étape (et la plus compliquée) est l'AnimationTimer, qui effectue les calculs physiques pour notre simulation à des intervalles réguliers. On doit redéfinir la méthode handle d'AnimationTimer, qui décrit le comportement de la simulation. Le principe est le suivant : pour chaque bille dans la liste de l'application, on définit un accélération de g vers le bas, une vitesse selon

la formule v(t) = a0 * t + v0, et une position selon la formule r(t) = 0.5 * a0 * t^2 + v0 * t + r0, avec a0 valant g vers le bas et v0 et r0 étant les anciennes valeurs de v et r pour la bille sur laquelle on travaille actuellement. Toutes ces valeurs sont bien sûr calculées sur chacun des 3 axes à l'aide de Vector3. Ensuite viennent les cas des bords de l'image : si on sort de l'image, la bille est replacée au bord et sa vitesse inversée par rapport au bord, avec une multiplication par le facteur de rebond. Enfin, on gère le cas où la bille passe sous le sol. Dans cette situation, on simule un rebond en mettant la bille au niveau du sol, en faisant un miroir de la vitesse par rapport au sol, et en compensant l'accélération avec la force de soutien du sol, ce qui donne temporairement une accélération parallèle au sol.

On a ensuite un if qui dessine un vecteur vert représentant la vitesse et un vecteur bleu représentant l'accélération si le boolean showVectors vaut true. On finit par une mise à jour des données de la Ball, puis de son dessin sur le Canvas.

Cette implémentation est un peu archaïque, et on est encore loin de NVidia PhysX, mais par ce travail, nous avons essayé de créer une simulation simple avec ce qu'on avait sous la main, et nous sommes plutôt satisfaits du résultat, que nous trouvons même assez ludique.

NOTE SUPPLEMENTAIRE :

Les arguments ci-dessous sont pour l'exemple standard, à savoir la carte d'Interlaken (à mettre dans les Run Configurations) :

interlaken.osm.gz N46E007.hgt 7.8122 46.6645 7.9049 46.7061 300 interlaken.png 10 0.05 1 0.95 200 false false

Un autre bon exemple utilise les arguments suivants pour les longitudes et latitudes :

7.0144 46.4194 7.2430 46.5334

Il s'agit de la région du Pays-d'Enhaut (Rossinière – Château-d'Oex – Rougement – Gstaad – Saanen) qui est assez vaste et assez démonstrative, malgré les bugs résiduels de notre bonus….

On peut aussi bien représenter les cas ne possédant pas de données OSM avec le relief noir-blanc qu'avec le relief du projet, on n'aura juste pas de carte topographique dans le 2ème cas.