*Maxime Kjaer, Timoté Vaucher*
*250694 246532*

# Rapport de bonus

## Téléchargement automatique des fichier `.osm` et `.hgt`

### Amélioration du point de vue de l'utilisateur

L'utilisateur n'a plus besoin de télécharger lui-même les données dont le programme a besoin pour générer la carte. Il lui suffit désormais de sélectionner les coordonnées du cadre de la carte, ce qui augmente la flexibilité et fait gagner beaucoup de temps.

Cependant, dû à la disponibilité limitée des fichiers HGT, cette fonctionnalité est limitée à la Suisse.

### Mise en oeuvre Java

Le téléchargement des données OSM se fait par une requête vers l'Overpass API (ressemblant à `http://overpass-api.de/api/interpreter?data=(node(minLat, minLon, maxLat, maxLon);<;>;);out;`), générée dans la classe `QueryGenerator`, à partir d'une expression régulière qui récupère les coordonées des *boundaries*. Cette requête est utilisée comme flux entrant dans `OSMMapReader` dont un constructeur particulier a du être généré pour l'occasion.

Le fichier HGT est téléchargé directement sur le site viewfinderpanorama.org par une requête générée dans `QueryGenerator` en utilisant principalement la fonction `format` de `DecimalFormat` pour récupérer le nom correct du fichier. Le fichier `.hgt` est enregistré dans un fichier temporaire, puis dézippé dans la méthode `readOnlineFile` de `HGTDigitalElevationModel` afin de pouvoir être lu comme un `File` dans le constructeur habituel de la classe.

## Ajout des noms de villes, parcs, étendues d'eau et forêts

### Amélioration du point de vue de l'utilisateur

L'ajout des noms augmente le nombre d'information sur la carte et permet à l'utilisateur de mieux situer les endroits sur celle-ci.

### Mise en oeuvre Java

Comme les noms de villes sont stockés dans des nodes possédant les attributs `place` (le type d'endroit) et `name` (le nom de l'endroit), il a fallu ajouter la liste des noeuds présents dans `OSMMap` ainsi qu'une liste d'`Attributed<Point>` dans `Map`. `OSMToGeoTransformer` a aussi du être modifié afin de trier et de ne garder que les noeuds qui possèdent les 2 attributs susmentionnés.

Tous les autres noms sont stockés dans des chemins fermés ou des relations multipolygon. Après avoir trié ces derniers, la méthode `generatePOI` va contrôler si les entités contiennent un des attribut de surface voulus (par exemple `natural : wood`), puis vérifie si elle possède l'attribut `name`, genère un point au millieu de la surface puis ajoute le nouveau `Attributed<Point>` à la liste du `Map.Builder`.

Un nouveau peintre appelé `place` prennant en argument une police et une couleur de texte a été créé. Il fait appel à une nouvelle méthode `drawPlace` de `Canvas` qui prend en argument un point, un nom, une police et une couleur pour dessiner le lieu. Dans l'implémentation de cette méthode dans `Java2DCanvas`, un alogrithme d'anti-collision a été ajouté. Il vérifie simplement si le nom à dessiner s'affiche sur un nom déjà peint en comparant un `Rectangle`, avec des dimensions obtenues grâce à la classe `FontMetrics`, à la liste de Rectangle de noms déjà présents. S'il s'avère qu'une collision a lieu, le nom est légèrement décalé sur l'axe Y en fonction de sa position relative avec le nom déjà dessiné.

# Ajout de textures à la carte

## Amélioration du point de vue de l'utilisateur

Les textures permettent de différencier des zones différentes. Elles ont été prises des cartes OpenStreetMap.

## Mise en oeuvre Java

Un nouveau peintre de `Polygon` prenant en argument le nom de la texture a été défini. Ces dernières sont spécifiées dans une `Map<String, TexturePaint>` afin qu'on puisse les récupérer avec l'argument du nouveau peintre.

Les `TexturePaint` sont construites avec une `BufferedImage` et une ancre pour la répétition de la texture. Pour dessiner ces textures, une surchage de `drawPolygon` a été crée. Celle-ci ne fait pas appel à `setColor` mais à `setPaint` qui prend une `TexturePaint` en attribut.

Pour l'instant seules les fôrets et les cimetières sont représentés par des textures. Mais l'ajout de nouvelles textures est très simple car il suffit de trouver une image adaptée et de modifier le peintre en conséquence.

# Ajout d'une interface graphique

## Amélioration du point de vue de l'utilisateur

Le fait d'avoir une interface facilite grandement la tâche pour l'utilisateur. Il n'a plus besoin de connaître l'ordre exact des arguments, puisque le formulaire possède des *labels* et *tooltips* qui le

guident dans la démarche. Ainsi, il n'a plus besoin de connaître le projet pour s'en servir. De plus, un utilisateur qui ne serait pas habitué à la ligne de commande ne sera pas dépaysé de son environnement habituel.

L'interface graphique fournit aussi un moyen plus simple de sélectionner les fichiers, surtout si l'utilisateur veut sélectionner des fichiers en dehors du dossier où est placé le projet (en ligne de commande, il aurait fallu rentrer le chemin absolu, ce qui est fastidieux).

Finalement, cette interface produit aussi une sortie verbeuse qui permet à l'utilisateur de suivre le progrès de l'exécution du programme.

# Mise en oeuvre Java

L'interface graphique est décrite dans la classe `GraphicalUserInterface`. Elle utilise Swing pour créer une fenêtre contenant de nombreux composants, dont des `JLabel`, `JTextField`, `JTextArea`, `JButton`…

Les boutons qui permettent de sélectionner et sauvegarder un fichier sont définis par la classe abstraite `FileButton` qui hérite de `JButton`. Un `FileButton` est intimement lié au `JTextArea` qui lui est donné par le constructeur. Le fait de sélectionner un fichier dans le `JFileChooser` créé en cliquant sur le bouton remplit automatiquement le `JTextArea` lié au bouton.

Les classes héritant de `FileButton` (à savoir `OpenFileButton` et `SaveFileButton`) doivent redéfinir la méthode `showDialog()` pour montrer une fenêtre d'ouverture ou de sauvegarde de fichier. Ils peuvent éventuellement aussi redéfinir une extension par défaut qui sera rajoutée aux noms des fichiers si besoin en est.

Le code pour créer une carte (défini dans `MapMaker`) tourne dans un fil d'exécution séparé afin de ne pas bloquer l'interface pendant la création de carte.

L'ajout de texte à la console de l'interface se fait par la méthode `System.out.println`. Pour cela, un nouvel `OutputStream` est défini par la classe `GUIConsoleOutputStream` qui ajoute le texte directement au `JTextArea` donnée en argument à son constructeur. La classe `GraphicalUserInterface` utilise cette classe pour rediriger l'output de `System.out`.