haute école
neuchâtel berne jura
arc ingénierie
saint-imier le locle delémont

Boxels

Rapport de projet: Projet de Printemps

Numéro de projet :	12inf-tb200
Mandant(s) :	Cécile Babiole (cecilebabiole@wanadoo.fr) David Gruenwald (David.Grunenwald@he-arc.ch)
Année :	2012

Résumé

Boxels est une installation artistique consistant en une matrice 2D de vérins (utilisés pour relever les lits d'hôpitaux). En faisant varier individuellement les niveaux d'élévation des vérins, on peut représenter en 3D dans la réalité un relief ou une image issue d'un ordinateur.

L'application (celle que ce rapport présente) permet de créer facilement des morceaux de cartes du territoire afin de créer un fichier qui sera exploitable par une carte Arduino, qui contrôlera l'œuvre dans le monde physique.

Malgré un temps assez réduit et quelques difficultés, l'étudiant estime avoir rempli le cahier des charges minimal et obtenu une application fonctionnelle.

Table des matières

1. Introduction

Le projet Boxels est né de la collaboration entre l'artiste Cécile Babiole (http://www.babiole.net), le studio de développement électronique Soixante Circuits (http://soixantecircuits.fr) et la He-Arc.

Boxels est une installation artistique consistant en une matrice 2D de vérins (utilisés pour relever les lits d'hôpitaux). En faisant varier individuellement les niveaux d'élévation des vérins, on peut représenter, en 3D et dans le monde réel, un relief ou une image issue d'un ordinateur.

2. Analyse

2.1. Analyse des besoins

L'objectif de ce projet (de printemps) est de réaliser une application permettant de réaliser des voyages immobiles, c'est-à-dire une succession de sélections, définies par un timing d'affichage précis, de morceaux de carte du territoire français accessible librement par l'Institut national de l'information géographique et forestière (IGN) sur internet comme stipulé dans la licence ouverte open licence dont elle fait l'objet. Le but étant de produire en sortie un fichier représentant le voyage immobile précédemment crée avec l'application.

2.2. Spécifications

Le système conçu devait initialement répondre aux exigences suivantes :

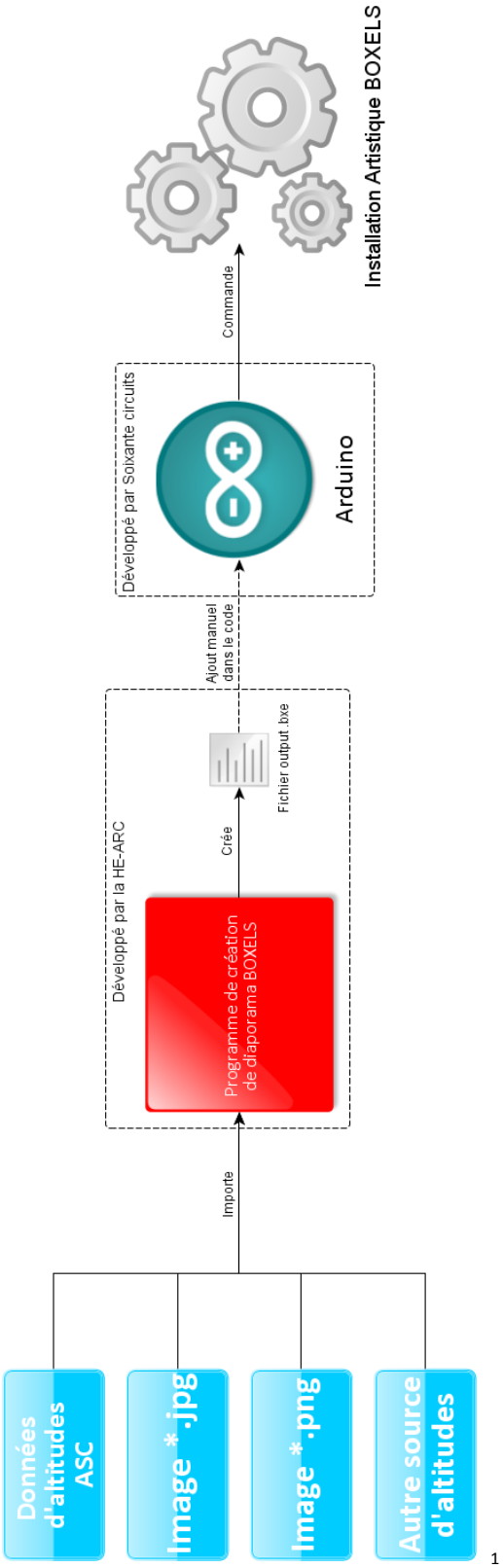
- Lecture et affichage d'une carte de relief de l'IGN.
- Visualisation interactive des cartes (sélection, zoom, rotation, …)
- Sélection des zones pour le voyage immobile
- Export des valeurs dans un format adapté au traitement par Arduino (Arduino Team, 2012)

La démarche proposée était la suivante :

- Analyse et rédaction de spécifications détaillées sur la base des interviews de la mandante
- Les fonctionnalités seront priorisées de façon à garantir la livraison d'une application utilisable.
- Choix justifié des méthodes et outils de développement
- Assimilation des compétences utiles au projet (algorithmes, méthodes et outils)
- Conception, développement et test de l'application décrite ci-dessus

2.3. Répartition des tâches

Comme le présent projet est le fruit de la collaboration entre divers intervenants, comme expliqué dans l'introduction de ce rapport, il a fallu clairement séparer la responsabilité de chacun. A l'heure de la rédaction de ce rapport, voici les divers découpages du projet ainsi que la responsabilité des diverses équipes avec leurs tâches respectives :



[1]

[1] *Ce diagramme reflète la compréhension de l'élève après discussion auprès de l'enseignant et risque de ne pas refléter la version finale adoptée durant le projet de Bachelor, car à l'heure actuelle cette partie n'a été que partiellement fixée !*

2.4. Architecture

Comme le programme est censé être lancé sur un ordinateur sans communication avec l'extérieur de la même machine (autre qu'avec éventuellement la board Arduino) et comme un seul intervenant utilise l'application durant toutes les procédures d'utilisation, l'architecture choisie a été une implémentation stand-alone dans un seul et unique programme lancé sur la machine client, sans infrastructure réseau particulière ni aucune interaction avec l'extérieur de l'environnement d'exécution hormis l'accès I/O aux fichiers des disques durs locaux.

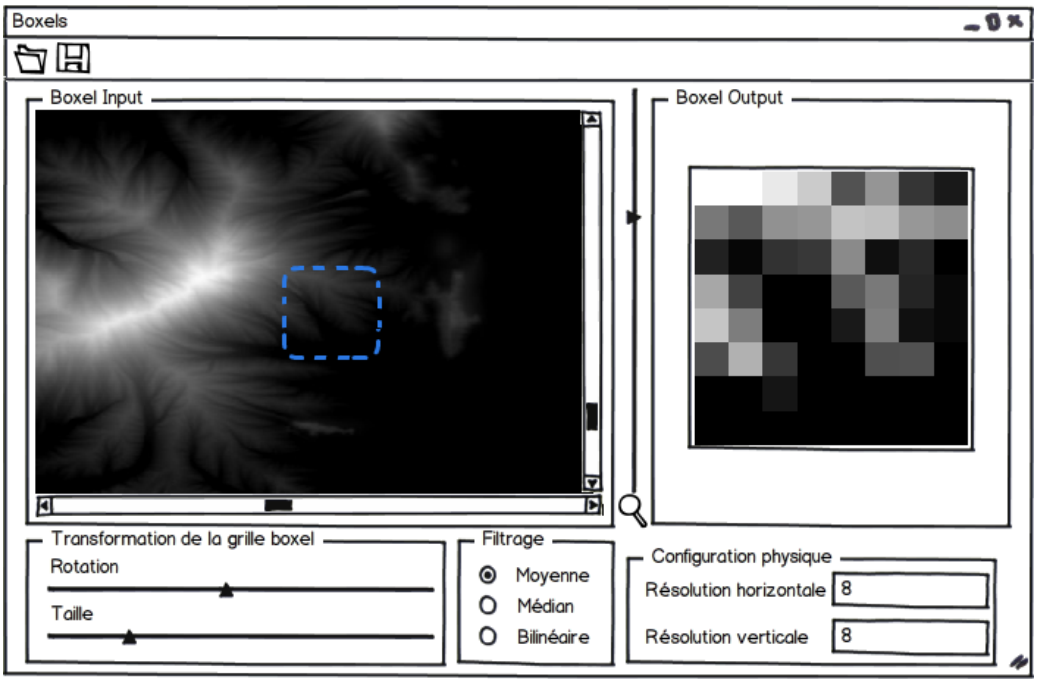
Le modèle d'architecture utilisé se rapproche fortement d'une approche MVC bien qu'elle ne respecte en aucun cas tous les aspects de ce type de construction. Elle a été utilisée principalement pour sa capacité à rendre le code clairement séparé et modulaire. Boxels ne fait que s'inspirer de MVC et en respectant pleinement cette architecture, on aurait probablement ajouté en complexité à l'ensemble du programme, ce qui n'était au sens de l'étudiant pas nécessaire pour le simple fait que le programme est relativement limpide dans son fonctionnement et par souci de simplicité et de clarté de l'implémentation future, une solution moins contraignante était toute indiquée.

3. Conception

3.1. Diagrammes

3.1.1. Prototype d'interface

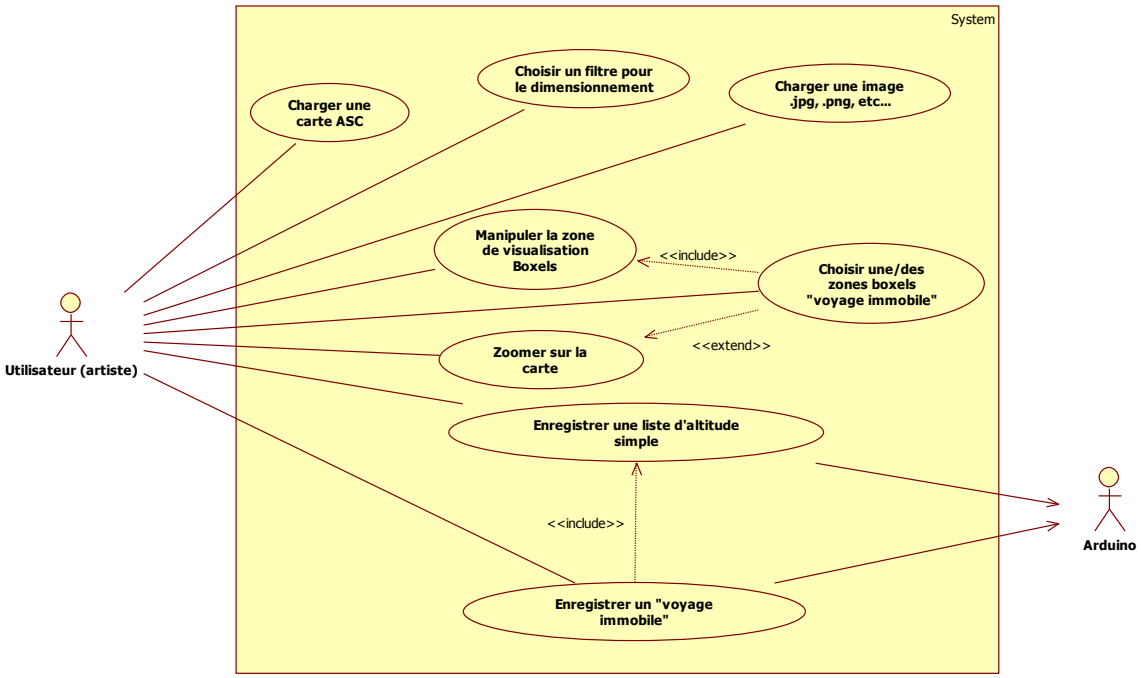
Comme le programme est destiné à un public non-initié, il faut que l'interface reste simple et intuitive.



La partie Boxel Input présente la carte une fois chargée. La partie Boxel Output représente la sortie sur le système Boxel physique. La partie Transformation de la grille boxel permet de transformer la zone sélectionnée (représentée en bleu). La partie Filtrage permet de choisir le type de filtrage pour l'agrandissement ou le rétrécissement pour le Boxel Output.

3.1.2. Diagramme Use-case

Comme expliqué précédemment dans le chapitre concernant l'Architecture, le fonctionnement interne de Boxels est très simple. En revanche, sa spécification n'étant pas tout à fait triviale, un diagramme UML use-case n'est pas de trop pour compléter le diagramme répartition des tâches du chapitre 2.3 :



3.2. Choix technologiques

3.2.1. Langage utilisé

Le choix du langage s'est rapidement révélé évident : La mandante travaille principalement sur Mac, l'étudiant travaille sur PC avec Windows. Il fallait donc choisir un langage qui permette de créer des applications multiplateforme. Le choix s'est rapidement porté sur **Java**, d'une part pour les constats énoncés plus haut, mais également et surtout parce que l'univers des langages gravitant autour du projet Arduino est principalement codé en Java (SDK en Java, IDE basé sur Processing[2] également en Java, etc…).

Qt avait été envisagé à cause de la facilité de l'étudiant pour ce langage, cependant ce seul argument n'était pas assez pertinent par rapport aux autres invoqués pour le choix de Java.

3.2.2. Technologie de traitement d'image

Une fois le choix du langage effectué, restait la question de trouver un moyen simple et efficace de faire du traitement d'image avec Java. Les besoins pour le projet sont les suivants :

- Effectuer des opérations de base telles qu'ajustement du contraste et de la luminosité (afin de pouvoir, en cas de besoin, ajuster les niveaux de l'image de la carte)
- Des opérations de redimensionnement de l'image
- Utilisation de filtres

Après analyse des solutions possibles, plusieurs possibilités se sont démarquées :

[2] *Langage orienté visualisation, principalement utilisé par les artistes car facile d'utilisation.*

- API ImageJ (ImageJ, 2012)
- Marvin (Muñoz, 2012)
- Java2D (Oracle Corporation, 2010)
- Imgsclr (The Buzz Media, 2011)

Une majorité de ces librairies (Marvin, ImageJ, Java2D) sont très complètes et couvrent un très large panel d'opérations sur des images standard utilisées en Java (principalement les BufferedImages, qui semblent être le format principal utilisé dans ce type de librairies/domaine).

En revanche, Imgsclr a très vite été écarté car il ne couvrait pas les besoins de l'application[3]. Cette librairie, comme son nom l'indique, a pour seule utilisation le redimensionnement d'image.

Après renseignement sur les autres librairies restantes, c'est Java2D qui a été choisi et ce pour plusieurs raisons :

- Java2D (Oracle Corporation, 2010) fait partie de la librairie standard de Java.
- M. Bilat en préconise l'utilisation.
- Développé et maintenu par la même équipe qui crée le langage, ce qui sous-entend un support sur le long terme (en principe).
- Elle permet de faire tout ce dont on a besoin pour le projet.

Il existe également un bon nombre d'implémentations commerciales payantes, cependant ces produits n'ont pas été analysés par l'étudiant, car elles sont souvent extrêmement chères à l'achat de licences.

3.2.3. Format du fichier d'entrée

L'IGN fournit deux types de fichiers : les *.xyz et les *.asc, tous deux consistant en des fichiers textes lisibles par l'homme. La différence fondamentale entre ces deux formats est la façon de présenter les altitudes. Pour le premier, les points d'altitude sont stockés ligne par ligne, soit une coordonnée x, y et z par ligne avec son altitude respective, soit 4 valeurs sur une même ligne. Le second format quant à lui se présente sous la forme d'une matrice d'altitude, soit une ligne de la matrice qui correspond à une ligne dans le fichier séparé par des espaces entre chaque coordonnée de la matrice pour les colonnes.

Dans les deux formats, les informations d'altitudes sont complétées par des metadata telles que la taille de la carte en hauteur et en largeur ainsi que la valeur numérique associée aux coordonnées dont les données font défaut, etc…

Après analyse approfondie des deux formats, le choix s'est finalement porté sur le support des *.asc dans le programme pour les raisons suivantes :

- Les *.asc sont moitié moins lourds que les *.xyz
- Les *.xyz sont stockés aléatoirement du point de vue de l'emplacement dans le fichier : Deux lignes contigües ne représenteront pas forcément des coordonnées voisines sur la carte finale, ce qui complique la tâche de parsing et surtout nécessite le stockage en mémoire vive de la carte avant son exploitation, ce qui représente déjà ~40Mo pour la carte de la France pour ~13Mo en version *.asc de la même carte.
- Le parsing prendra potentiellement plus de temps (bien que théoriquement la différence est infime) sur les fichiers *.xyz parce que contenant plus de lignes que les fichiers *.asc car

[3] *Il est nécessaire de pouvoir faire du filtrage sur l'image créée.*

chaque ligne correspond à une coordonnée d'altitude. On chargera plus de données par ligne qui seront traitées en interne du programme avec des *.asc qu'avec des *.xyz. L'utilisation de traitement de plus de données en mémoire vive restera toujours plus rapide que la récupération plus fréquente sur le disque dur. Pour résumer, charger plus mais moins souvent et traiter plus rapidement en mémoire que de charger un fichier plus long donc plus d'échange I/O avec un disque dur vraisemblablement plus lent que la mémoire vive. Ces hypothèses partent du principe que l'on charge le fichier ligne par ligne.

3.2.4. Format du fichier de sortie

Initialement le format de sortie de l'application avait été envisagé en XML sans réflexion autre que cette approche intéressait l'étudiant.

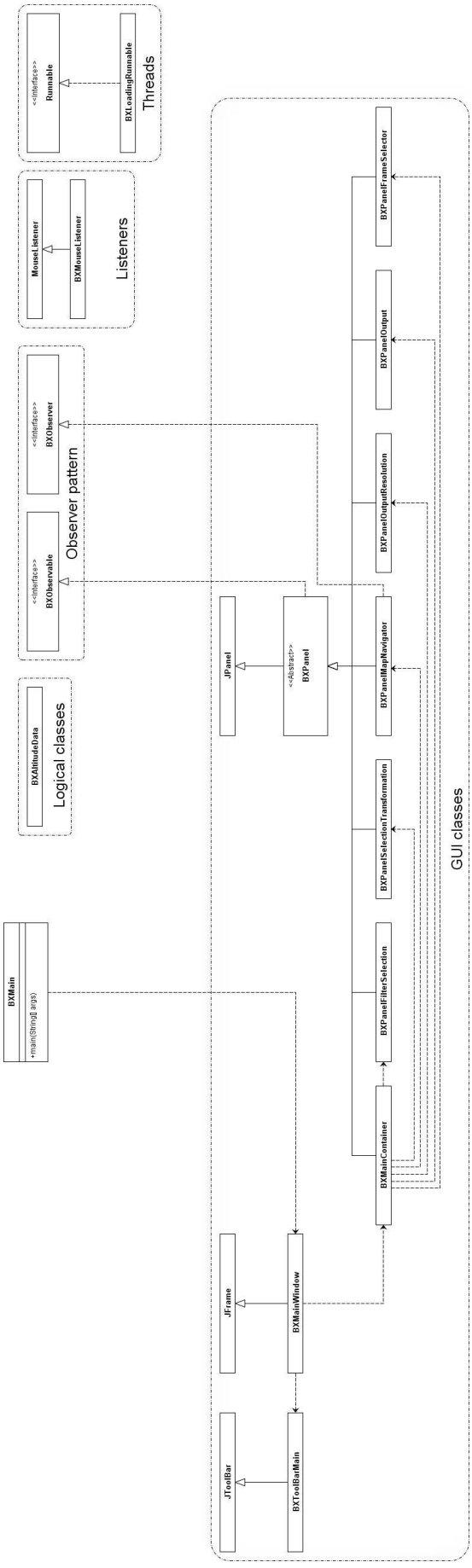
Après moult refontes des spécifications, il a été décidé d'abandonner le format XML au profit d'un format « maison ».

Ce choix s'est avéré nécessaire pour l'unique raison que le format XML est particulièrement lourd et verbeux, ce qui est particulièrement inadapté au monde de l'embarqué.

En effet, la procédure imaginée initialement était de sauvegarder le fichier sur le disque dur de l'ordinateur pour ensuite être post-traité pour l'envoi sur Arduino. De plus, il est facilement imaginable que le programme Boxels fournisse dans un futur proche des fonctionnalités telles que l'envoi direct des coordonnées à Arduino via le port USB, ce qui n'avait pas été envisagé à l'origine par l'étudiant.

3.3. Diagramme de classes

Le présent diagramme de classe rend compte de l'implication des classes, groupées par fonctionnalités :


Threads
<<interface>>
Runnable
BXLoadingRunnable
Listeners
MouseListener
BXMouseListener
Observer pattern
<<interface>>
BXObserver
<<interface>>
BXObservable
Logical classes
BXAltitudeData
BXMain
+main(String[] args)
GUI classes
JPanel
<<Abstract>>
BXPanel
BXPanelFrameSelector
BXPanelOutput
BXPanelOutputResolution
BXPanelMapNavigator
BXPanelSelectionTransformation
BXPanelFilterSelection
BXMainContainer
JFrame
BXMainWindow
JToolBar
BXToolBarMain

4. Réalisation

4.1. Classes

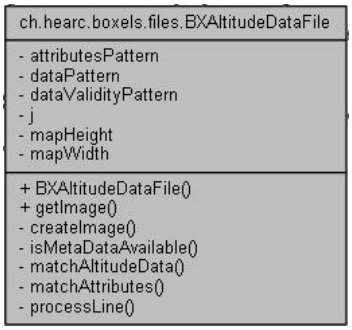
Comme le langage choisi ainsi que l'IDE utilisé le suggèrent[4], Boxels est composé de nombreuses classes. Cependant, elles ne sont nombreuses que pour séparer clairement les divers groupes logiques du système. Le présent chapitre présente uniquement les classes vraiment importantes du projet et de ce fait s'avère incomplète. Pour obtenir une documentation complète (en anglais), il est possible de générer la documentation au moyen de Doxygen (voir chapitre 4.6).

4.1.1. BXAltitudeDataFile

Cette classe représente un fichier *.asc dans le sens où il encapsule toutes ses propriétés, métatags et données d'altitude, mais aussi le chemin vers le fichier d'origine.

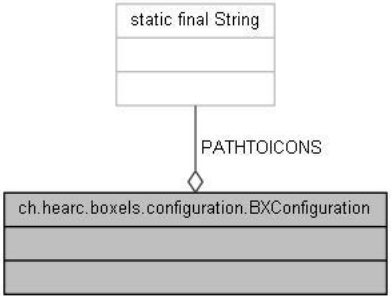
C'est sur cette classe que travaille le thread lancé pour le chargement de la carte (voir chapitre 4.1.6).

C'est entre autres, cette classe qui parse le fichier *.asc.



4.1.2. BXConfiguration

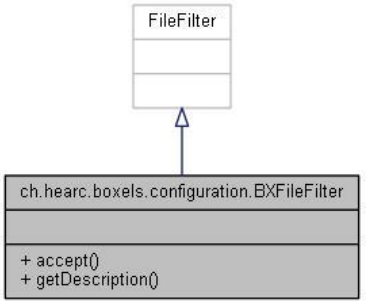
Cette classe est la classe de configuration du projet, elle a pour seule utilité de contenir toutes les constantes de configuration. A l'heure actuelle, elle ne contient que le chemin vers les icônes du programme.

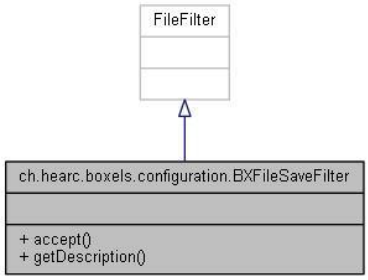


4.1.3. BXFileFilter/ BXFileSaveFilter

Héritant de *javax.swing.filechooser.FileFilter*, il s'agit d'une classe de filtre permettant à un utilisateur de ne sélectionner qu'un nombre réduit de fichiers. Dans le cas de ces classes nous avons réduit le choix à des fichiers de type *.asc pour le chargement, et uniquement à des fichiers *.bxe pour la sauvegarde.

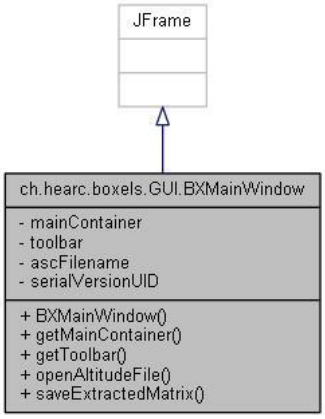
[4] *En effet, Eclipse force par défaut à créer une unité de compilation par fichier (un classe = un fichier)*





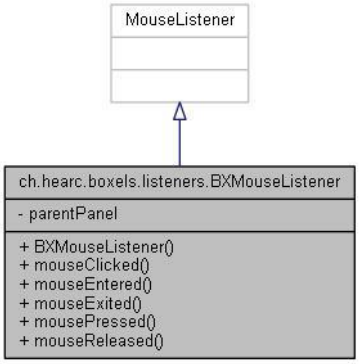
4.1.4. BXMainWindow

Il s'agit de la fenêtre unique et principale du programme héritant de *JFrame*. C'est ce frame qui va contenir l'ensemble des éléments graphiques du projet.



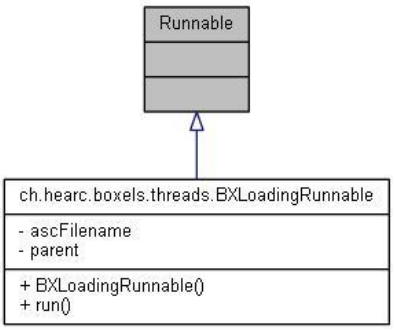
4.1.5. BXMouseListener

Cette classe héritant de *MouseListener* permet de gérer les clics sur la carte. On utilise une classe dérivée plutôt qu'une classe interne anonyme par souci de lisibilité et de maintien du code.



4.1.6. BXLoadingThread

Il s'agit d'une des classes les plus importantes du projet. C'est dans cette classe, implémentant l'interface Runnable, que sont lancées, dans un thread séparé, toutes les opérations de parsing du fichier *.asc. Le déplacement en thread était nécessaire car l'opération est longue et figeait l'interface.



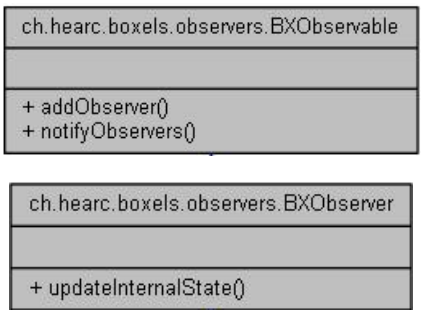
4.1.7. BXObservable/BXObserver

Implémentation du design pattern Observer, ces deux interfaces permettent de définir un sujet et un observateur.

Elle est utilisée pour notifier le panneau de sortie BXPanelOutput d'un clic sur la carte, afin d'actualiser sa vue.

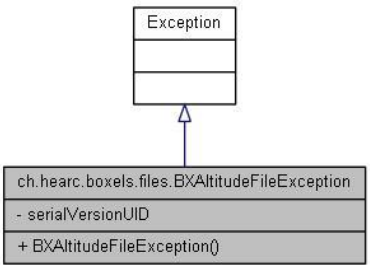
Le choix d'utiliser ce pattern permet de réutiliser ces concepts avec d'autres panneaux dans une future version du programme, plutôt que de faire une solution « maison » horrible à la maintenance.

De plus, comme présenté sur le diagramme de classe du chapitre 3.3, les BXPanels (chapitre 4.1.10) implémentent l'Observer et seul BXPanelMapNavigator (chapitre 4.1.10.2) implémente l'interface Observable.



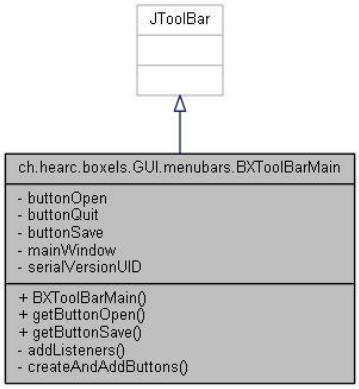
4.1.8. BXAltitudeFileException

Cette classe est en réalité une Exception qui est lancée lorsque le fichier *.asc est mal formé ou que quelque chose s'est mal déroulé durant le parsing du fichier.



4.1.9. BXToolBarMain

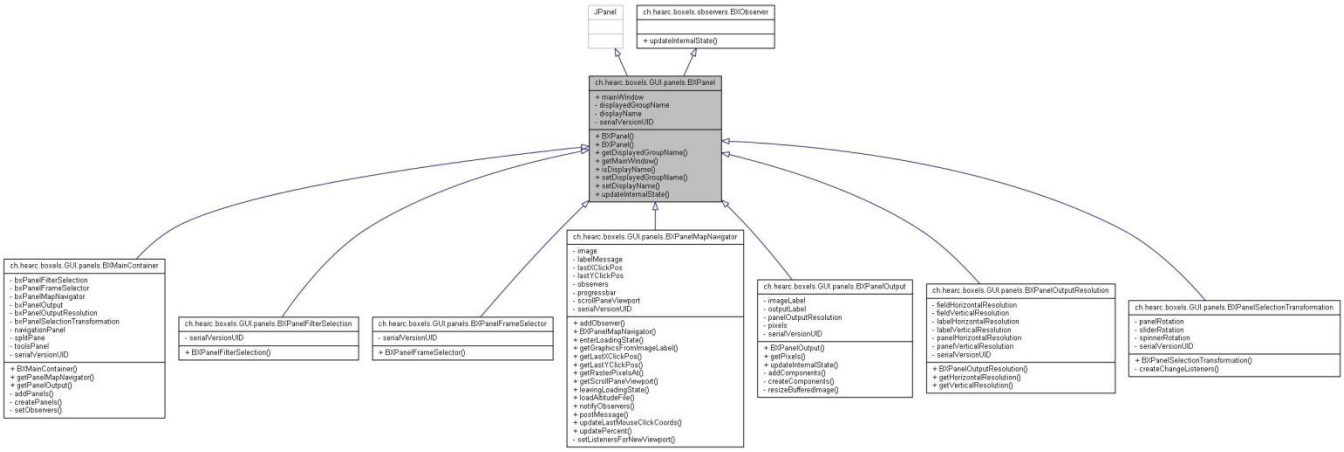
Il s'agit de la barre de menu du programme, avec les actions ouvrir, fermer et sauvegarder.



4.1.10. BXPanel

Il s'agit du cœur de la hiérarchie de classes graphiques du projet Boxels. Ces panneaux représentent l'intégralité de l'interface interne de la fenêtre principale du programme.

BXPanel est le sommet de cette hiérarchie. Cette classe est abstraite et fait hériter tous les autres JPanels du programme. La principale différence avec les JPanel conventionnels est qu'un titre est systématiquement requis pour les BXPanels.



Seuls les panels intéressants pour le travail de printemps sont expliqués en détails ici !

4.1.10.1. BXMainContainer

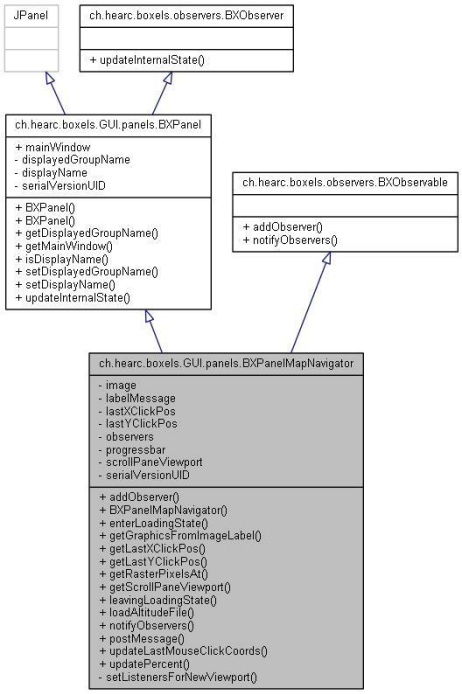
Ce panel ne fait que contenir tous les autres sous-panels et les instancie également.

4.1.10.2. BXPanelMapNavigator

Il s'agit sans conteste de la classe la plus lourde du projet. En effet c'est dans ce panel que l'on va afficher la carte qui sera générée après parsing.

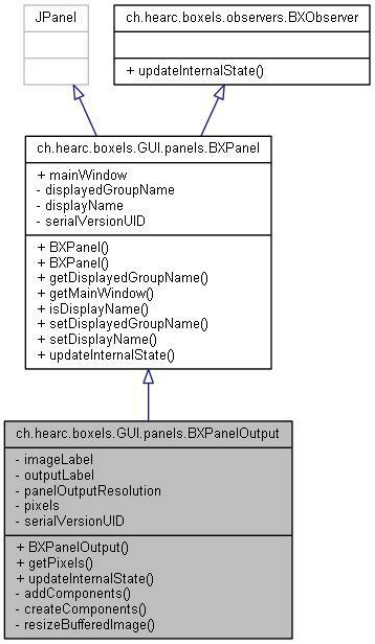
Malheureusement cette classe est encore trop couplée avec le reste de l'interface[5], il faudrait donc trouver un moyen de palier à ce problème dans des versions futures du programme.

[5] *Surtout à cause de la communication entre les panneaux*



4.1.10.3. BXPanelOutput

C'est dans ce panneau qu'est affiché le résultat de la sélection sur la carte chargée. C'est dans ce panel qu'est affiché le logo de l'application ainsi que son titre lorsqu'aucune zone n'a été choisie.



4.1.10.4. BXPanelOutputResolution

Dans ce panneau, deux lignes de texte éditables permettent de choisir la résolution de sortie de la grille Boxels (configuration physique de l'installation).

4.2. Problèmes rencontrés

4.2.1. Gestion des ressources

Lors du packaging en *.jar du projet, les icônes n'apparaissaient pas. Cela était dû au fait que les icônes était spécifiées avec des chemins « système », la solution fut d'utiliser un appel à la fonction suivante :

```
buttonOpen.setIcon(new ImageIcon(getClass().getResource(BXConfiguration.PATHTOICONS + "open.png")));
```

4.2.2. Traînées et normalisation progressive

Une solution d'abord envisagée au problème énoncé dans le chapitre 4.2.4 afin de ne pas stocker les données mais de pouvoir tout de même filtrer l'image résultante, à savoir la normalisation progressive (on normalise les valeurs trouvées sur la ligne avec une valeur qui se redéfinit au fur et à mesure du parcours de la ligne) a été implémentée. Malheureusement, cela a pour effet de créer des traînées noires sur l'ensemble de l'image, ce qui n'est pas souhaitable. La solution a été de stocker les données de la mémoire (ce qui a posé des problèmes supplémentaires comme ceux présentés par exemple dans le chapitre 4.2.4).

4.2.3. Images 8 bits ou 16 bits

Initialement, les images sont stockées sur 8 bits par composante, soit 8 bits par pixel pour une image en niveaux de gris.

Après génération de l'image (même après avoir résolu le problème du chapitre 4.2.1), il s'est avéré que l'image était bien trop sombre dans les zones de faible altitude.

Une idée proposée par M. Grunenwald qui est finalement adoptée dans la version finale est de passer de 8 bits en 16 bits, pour obtenir une meilleure résolution des niveaux de gris, ce qui a pour effet de rendre l'image plus visible dans les zones de faible altitude.

Pour rendre la carte encore plus visible, un traitement supplémentaire devra être fait comme suggéré au chapitre 4.5.2.

4.2.4. Stockage du tableau de valeurs

A l'origine, l'idée était simple : charger au fur et à mesure le fichier ligne par ligne puis parser les coordonnées trouvées sur la ligne pour les sauvegarder dans un tableau d'entier.

Le principal problème de cette approche est l'espace nécessaire en mémoire vive pour le traitement : En effet, le stockage d'un tableau de plusieurs milliers d'entiers n'est en aucun cas négligeable. Après test, il s'avère que, pour le fichier original déjà de taille conséquente (~80 Mo), fait atteindre des sommets à la consommation mémoire après chargement et parsing et stockage en mémoire sous forme d'entiers (+500 Mo minimum voire 1 Go pour l'ensemble de la JVM hébergeant le programme).

Une solution d'abord envisagée était de ne pas du tout stocker ces valeurs, et de constituer la buffered image directement depuis le fichier. Or, il s'est avéré avec le temps qu'il était nécessaire de stocker ces valeurs, ne serait-ce pour faire du traitement sur les valeurs, car il est obligatoire de passer toutes les valeurs pour connaître la valeur d'altitude maximum du relief chargé et ainsi pouvoir dimensionner ces valeurs pour les afficher correctement pour le niveau de profondeur de couleur choisi (voir chapitre 4.2.1).

Afin de diminuer l'impact sur les performances en mémoire, le stockage des informations d'altitude sont stockées en *char* car stockées sur 2 octets[6] (contre 4 pour les entiers) elles peuvent accueillir des

[6] *Testé pour Java 1.6 sur architecture 32 bits*

valeurs jusqu'à 2^{16} (soit 65536) en non-signé et un peu moins de la moitié en signé, ce qui est un bon compromis entre espace consommé et altitude maximum stockable (aucun mont sur terre ne mesure 32 km d'altitude).

Après test de la solution énoncée ci-dessus, il a été remarqué que la consommation mémoire est largement revue à la baisse et atteint un seuil acceptable pour ce type d'application (350 Mo en charge maximum).

4.2.5. Déploiement sous MacOSX

Comme la mandante possède un Mac, l'étudiant s'est intéressé aux techniques de déploiement dans un environnement Mac. Il existe plusieurs types qui ont été testés

- Java web start
- Jar bundler (développé par Apple)
- Simple *.jar
- Déploiement via Eclipse pour Mac

Premièrement, Java Web Start nécessite un serveur pour tourner, dans le sens ou l'application est stockée puis copiée/exécutée sur la machine cliente[7]. Cependant, il semblerait que cette solution ne soit pas adaptée car elle nécessite un accès constant à internet, ce qui s'avère compromis en raison de la nature itinérante d'une exposition telle que celle où sera présentée l'œuvre.

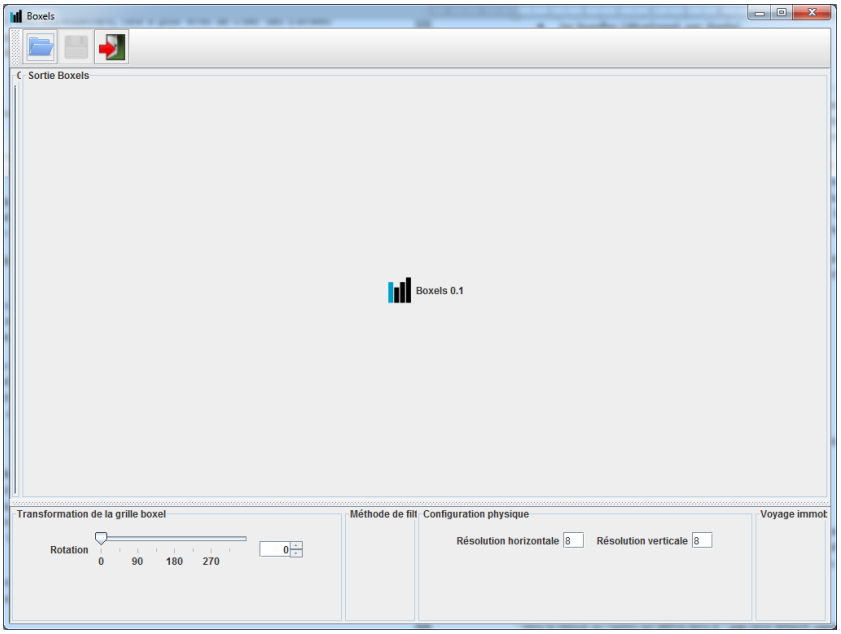
Jar Builder est un programme maintenu par Apple dans le but de compacter les *.jar dans une structure *.app représentant une application dans un environnement MacOS. Cette méthode, bien que très propre est esthétique[8], a été rejetée car l'étudiant travaille sur Windows et que cette méthode n'est pas faisable sur un ordinateur pourvu de Windows. Les mêmes conclusions ont été tirées pour le déploiement via Eclipse, qui est réalisable uniquement via une version Mac de l'IDE.

C'est finalement le simple *.jar qui sera utilisé, car il n'a pas posé de problèmes à la mandante lors des premiers essais, ce qui avait été le cas sur le MacBook testé durant les phases de test de déploiement.

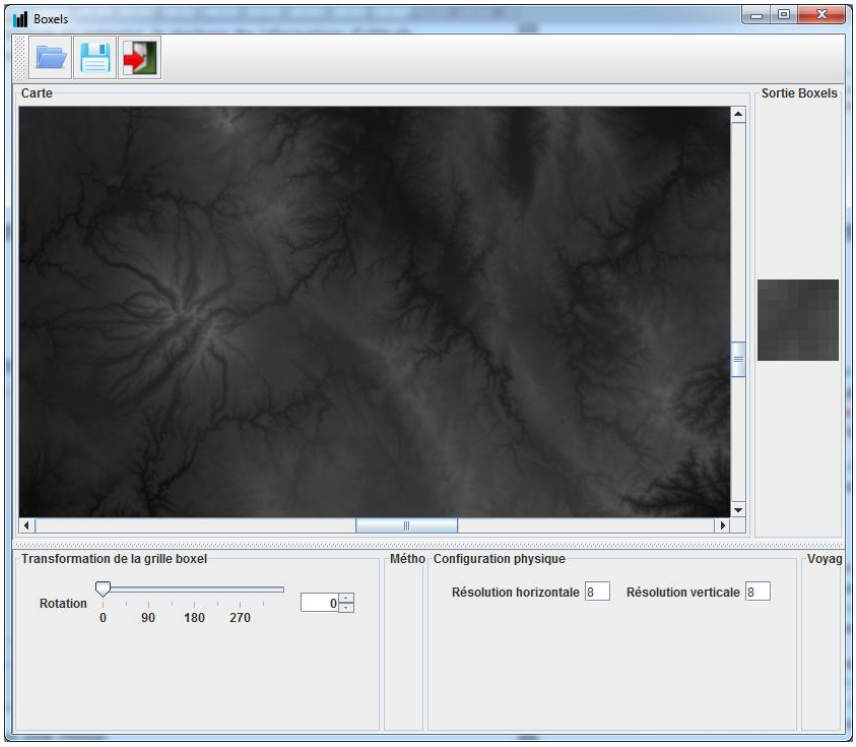
4.3. Interface résultante

Une fois l'interface codée, voici à quoi ressemble Boxels :

[7] *Dans la mesure où l'option est définie dans le *.jnlp (Java Network Launching Protocol).*
[8] *Ajout d'icônes, disk images d'installation, installation sur l'ordinateur, etc.*



Une fois un fichier d'altitude ouvert et une zone choisie:



4.4. Bugs connus

4.4.1. Rafraichissement des résolutions

Il semblerait que la prise en compte effective de la résolution de sortie après changement se fasse uniquement après perte de focus de la zone de texte.

4.5. Améliorations possibles

4.5.1. Le couplage fort des classes GUI

Afin de pouvoir se transmettre de l'information, les classes GUI se connaissent entre elles via des références membres en interne de chaque classe nécessitant une communication avec une autre classe (notamment activation/désactivation des icônes de la toolbar lors du chargement du fichier *.asc).

Il serait très indiqué dans une future version du programme de découpler ce comportement afin de garantir l'aspect open-close des versions ultérieures.

L'étudiant n'a, à l'heure actuelle, mené aucune recherche afin de palier à ce problème, principalement par manque de temps.

4.5.2. Eclaircissement de la carte

Comme la carte est relativement sombre même après avoir résolu les problèmes énoncés dans le chapitre 4.2, il faudrait encore intégrer une étape de filtrage supplémentaire pour rendre la carte bien visible sur tout le territoire. Cette modification ne toucherait que la visualisation de la carte et en aucun cas l'ensemble des données !

4.6. Génération de la documentation

Pour générer la documentation dynamiquement depuis le code source, il est possible de le faire via l'outil Doxygen. Voici la marche à suivre pour régénérer la documentation :

1. Télécharger et installer Graphwiz (contenant l'addon dot) : http://www.graphviz.org/
2. Télécharger et installer Doxygen : http://www.stack.nl/~dimitri/doxygen/
3. Se munir du fichier Doxygen issu du dépôt SVN (Doxyfile.doxy)
   ATTENTION : Ne pas le déplacer, les chemins spécifiés dans le fichier doxygen sont relatifs !
4. Lancer Doxywizard et lancer la génération de la documentation sur le fichier Doxygen.

Un nouveau dossier frère du fichier Doxygen sera créé contenant la documentation complète du projet issue directement des sources java.

5. Conclusion

Après avoir terminé le projet, force est de constater que les exigences et la difficulté de l'implémentation ainsi que toutes les implications sous-jacentes découvertes tout au long du développement ont été clairement sous-estimées. Bien que le projet ait abouti à un programme parfaitement utilisable et fonctionnel, il n'en demeure pas moins qu'une version obéissant au cahier des charges minimal.

Ce résultat est dû principalement à deux aspects externes au projet : premièrement, le semestre était extrêmement court (environ 8 semaines) et de plus il a été amputé de séances de projet par la préparation des portes ouvertes de la HE-Arc, et deuxièmement la masse de travail pour d'autres projets a passablement morcelé les phases de travail, ce qui a notablement retardé le projet.

Malgré tous ces contretemps et les problèmes rencontrés, l'étudiant estime avoir rempli le cahier des charges minimal et obtenu un résultat convenable.

6. Bibliographie

- **Arduino Team. 2012.** Site officiel arduino. *Site officiel arduino.* [En ligne] 2012. http://www.arduino.cc/.
- **Debrauwer, Laurent. 2009.** *Design Patterns pour Java.* s.l. : eni Editions, 2009. ISBN 978-2-7460-5057-0.
- **Heesch, Dimitri van. 2012.** Site web officiel Doxygen. [En ligne] 2012. http://www.stack.nl/~dimitri/doxygen/.
- **ImageJ. 2012.** Site officiel de ImageJ. [En ligne] 2012. http://rsbweb.nih.gov/ij/.
- **Muñoz, Danilo. 2012.** Site officiel du projet Marvin Image Processing Framework. [En ligne] 2012. http://marvinproject.sourceforge.net.
- **Oracle Corporation. 2010.** Desktop Java: Java 2D API. *Site officiel Java.* [En ligne] 2010. http://java.sun.com/products/java-media/2D.
- **The Buzz Media. 2011.** *Forge officielle imgsclr.* [En ligne] 2011. https://github.com/thebuzzmedia/imgscalr/.

7. Annexes

- Planning initial
- Forge publique He-Arc (http://projets-labinfo.he-arc.ch/projects/12inf-tb200)
- Wiki de la forge
- Documentation générée (fin du projet de printemps, au 6 mai 2012)
- Fichier doxygen pour la génération automatique.

8. Index