Rapport technique

Projet Tuteuré

Générateur matriciel de heightmap

Hugo Talbot - Jonathan Jobkel - Vincent Delannoy

Sous la direction de Mme Ferreira Da Silva

IUT Lyon 1, département Informatique

Remerciements

Nous adressons nos remerciements à Mr Peytavie et à Mme Ferreira Da Silva pour leur suivi tout au long du projet. Nous tenons également à remercier Julien Delannoy, le frère de Vincent, pour son aide à deux reprises lorsque nous étions bloqués.

Introduction

Dans le cadre du projet tuteuré de fin d'études se déroulant sur plusieurs mois, nous avons été amenés à choisir un sujet dans le but de mettre en pratique nos connaissances en développement d'applications informatiques. Nous avons décidé de proposer notre propre sujet : une application de génération aléatoire de carte 2D. Nous allons vous présenter ce programme, que nous avons baptisé MapGen, selon le plan suivant :

1. Présentation du programme

1.1. Motivations

Nous avons proposé ce sujet de projet tuteuré pour des motifs à la fois personnels et d'ordre technique.
Nous avons tout d'abord été attirés par l'aspect créatif du logiciel. Ce sujet nous a offert la possibilité d'utiliser nos connaissances en programmation pour produire des éléments visuels. Cet aspect était d'autant plus attirant que les données produites appartiennent à un processus de création plus vaste, en s'ouvrant sur la possibilité de concevoir des univers virtuels.
Par ailleurs, nous étions particulièrement attirés par les langages de programmation objet et le développement d'applications bureau. Ce sujet nous a dans cette optique permis d'utiliser et d'approfondir nos connaissances en Java.
Nous avons ainsi pu concevoir la structure de notre programme, et manipuler nos propres classes et méthodes dans une logique objet.

1.2. Objectifs du programme

Notre projet est de générer une image 2D représentant la carte d'un terrain fictif, cohérent en particulier du point de vue des reliefs. Le résultat d'une génération sera ainsi une image permettant de distinguer le relief du terrain.

Nous souhaitons également permettre l'export de cette image sous différents formats d'image et selon deux versions : couleur ou niveau de gris. Cette seconde version est en particulier destinée à un usage pour un logiciel tiers, capable de l'utiliser en tant que heightmap pour reconstituer le terrain en 3D.

Les objectifs du cahier des charges sont donc :

- Génération d'une carte crédible en termes de relief.

Nous devons être en mesure de générer des terrains, aux reliefs uniquement montagneux dans un premier temps, qui pourraient correspondre à un terrain réel du point de vue des altitudes.

- Gestion des altitudes, converties en couleurs pour l'image finale.

Le produit de la génération sera en premier lieu une large matrice d'entiers. Cette matrice devra être convertie en image puis être affichée. Plus exactement, il y aura deux images par matrice générée : une version en nuances de gris, adaptée à l'utilisation de l'image en tant que heighmap, et une version en couleur, adaptée à une lecture visuelle et qui pourra permettre d'afficher des informations supplémentaires telles que les étendues d'eau.

- Dimensions de la carte définie par l'utilisateur (longueur et largeur).

Au moins une entrée utilisateur est prévue dans un premier temps, à savoir le choix des dimensions du terrain à générer (dimensions en pixel). Il ne s'agit pas de déterminer le niveau

de détails du terrain, mais bien son étendue. Une option de zoom / de-zoom devra accompagner l'affichage des terrains.

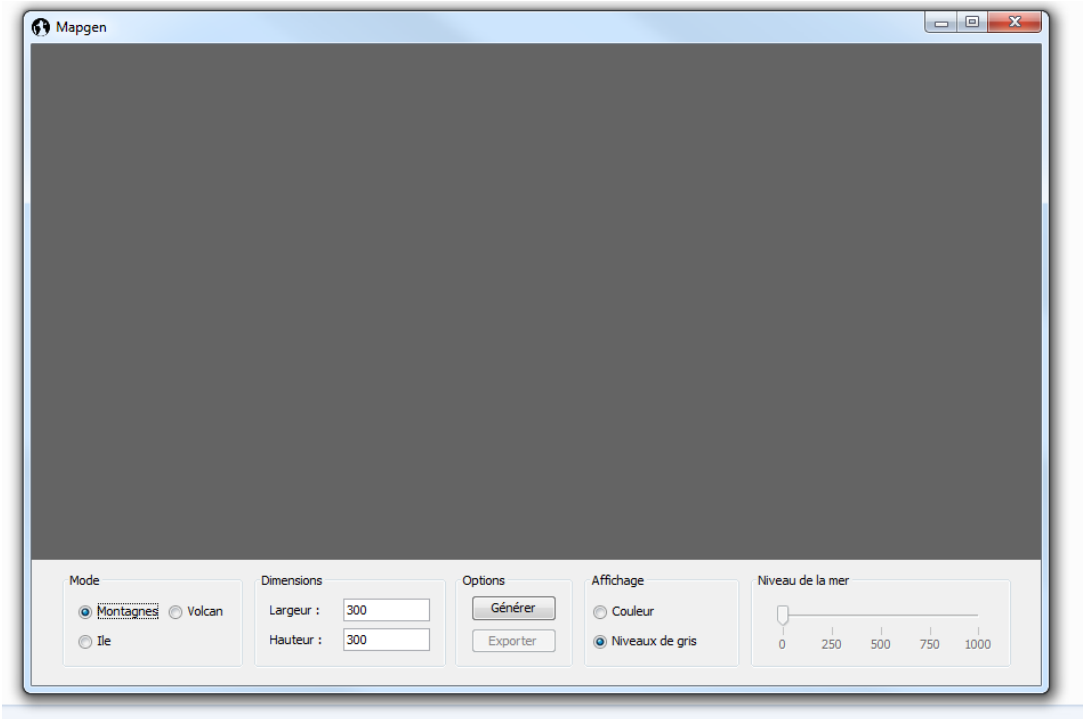
- Options d'export

Tout terrain généré devra pouvoir être exporté en tant qu'image, en niveaux de gris ou couleurs et aux trois formats suivants : .png .jpg .gif

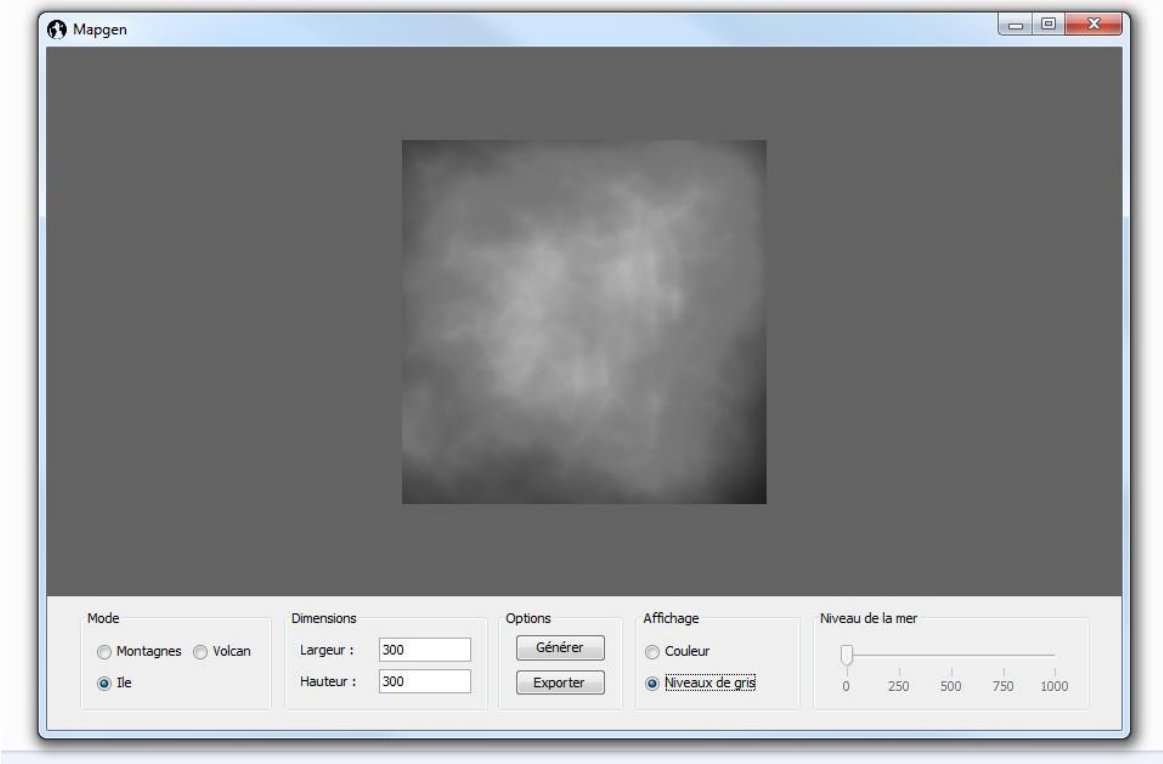
1.3. Fonctionnalités

L'interface de notre application est séparée en 2 parties. Sur la partie supérieure : l'espace réservé à l'affichage de la carte générée, sur la partie inférieure : les contrôles que l'utilisateur peut exercer sur l'image. Le bouton exporter fait apparaître une fenêtre pour enregistrer l'image en choisissant son format. Lorsque le bouton générer est actionné, une nouvelle image apparaît au centre de la fenêtre, avec la taille renseignée par l'utilisateur et en niveau de gris par défaut. On peut ensuite zoomer et dézoomer l'image avec la molette de la souris et la déplacer en maintenant le clic gauche enfoncé. Une fois l'image générée, on peut basculer entre un affichage en couleur ou en niveaux de gris, et faire varier le niveau de l'eau lorsque l'image est affichée en couleur.
La partie mode permet de choisir le type de génération souhaitée, on renseigne la longueur et la largeur de l'image en pixels dans la partie dimensions.
Lors d'un export, si l'image actuellement affichée est en couleur, le fichier enregistré sera en couleur, de même pour le niveau de gris.

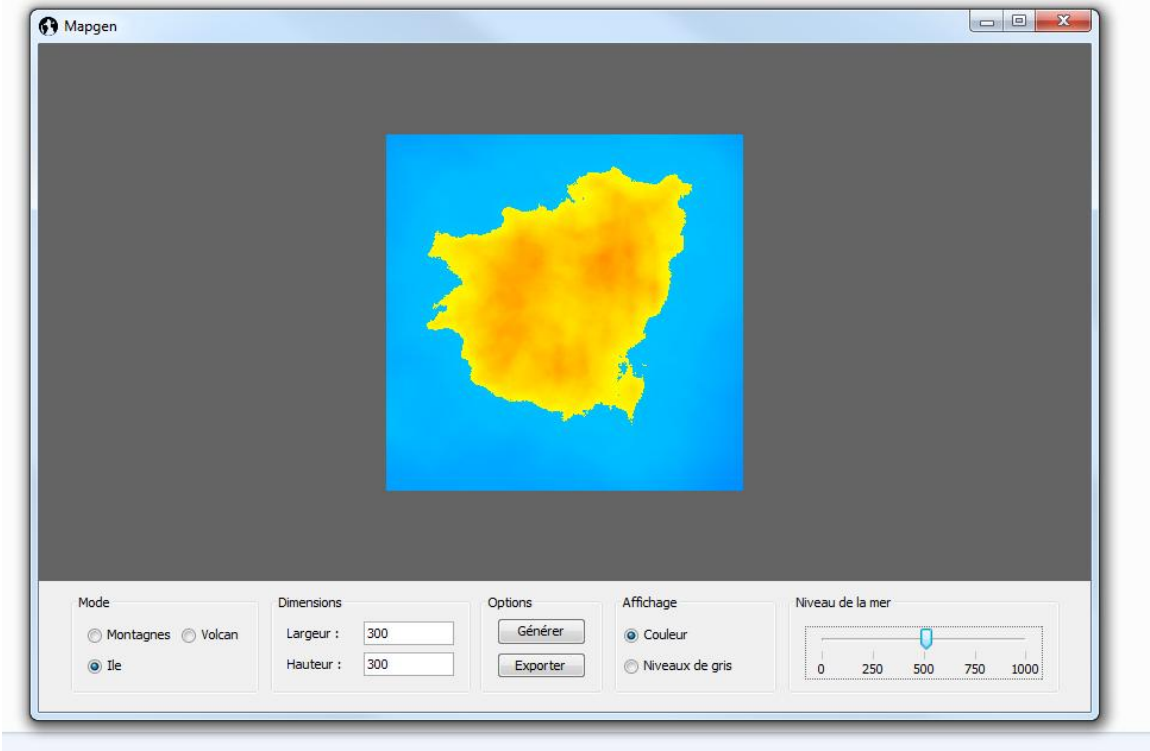
La fenêtre du programme au lancement :



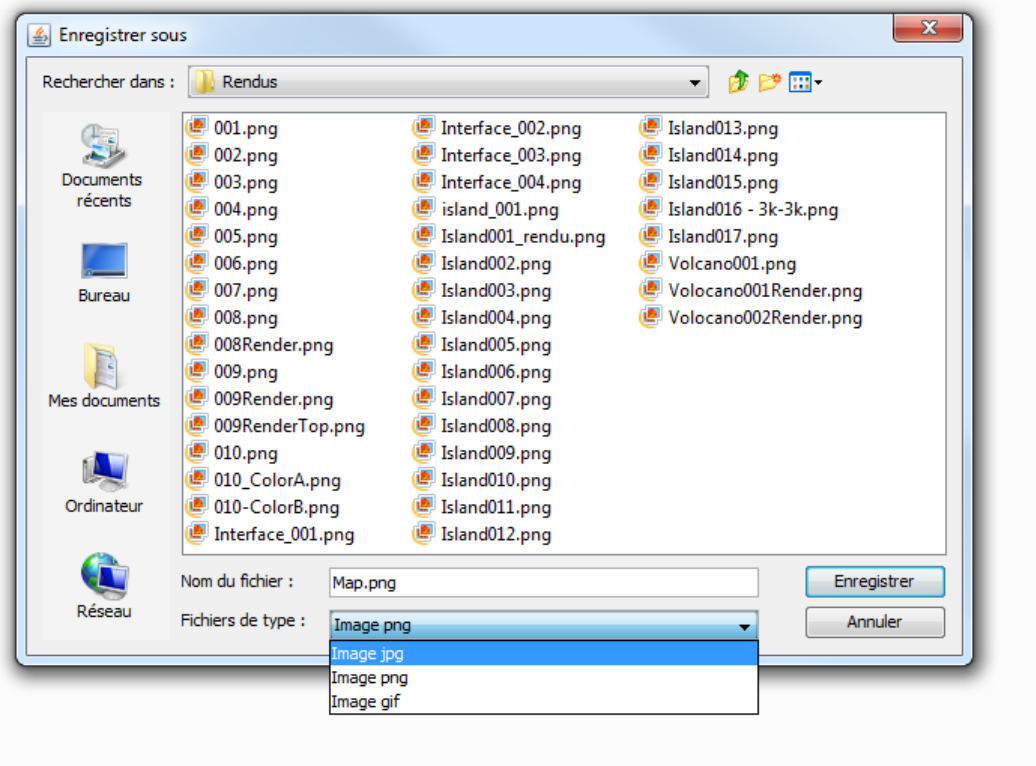
Après génération :
Version Niveaux de gris :



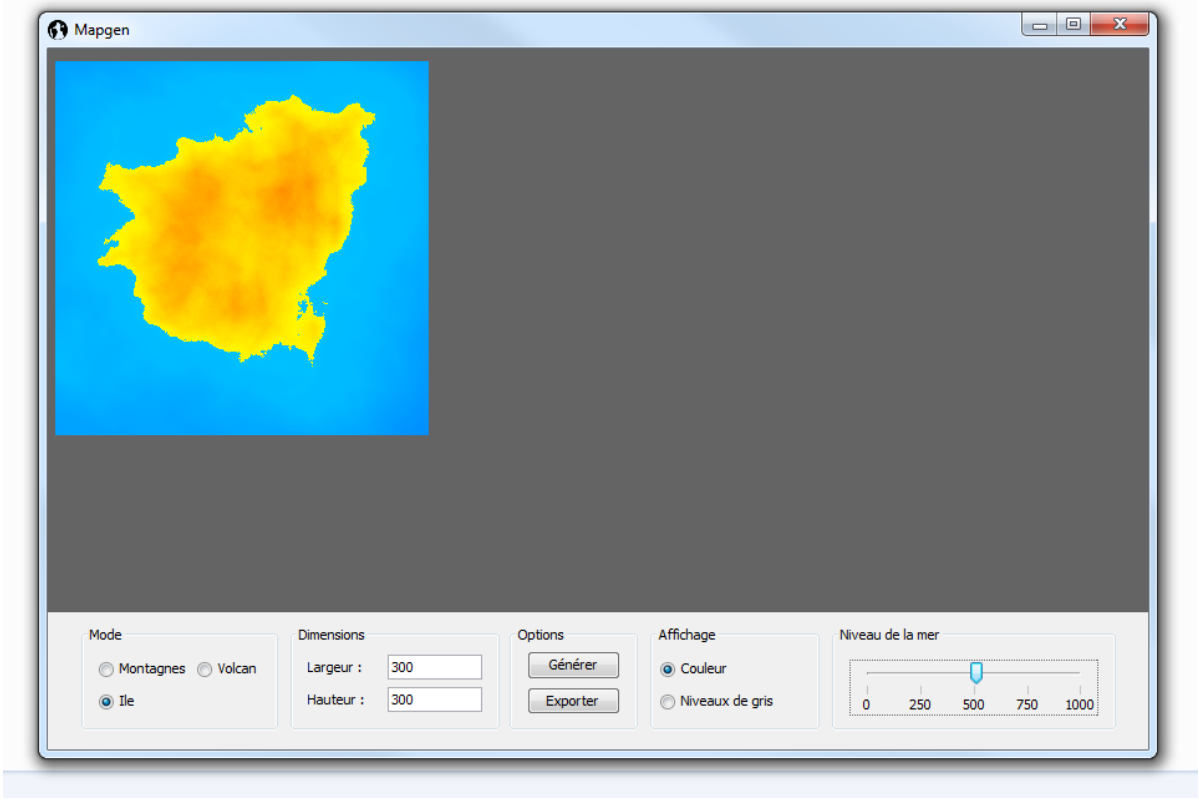
Version couleur :



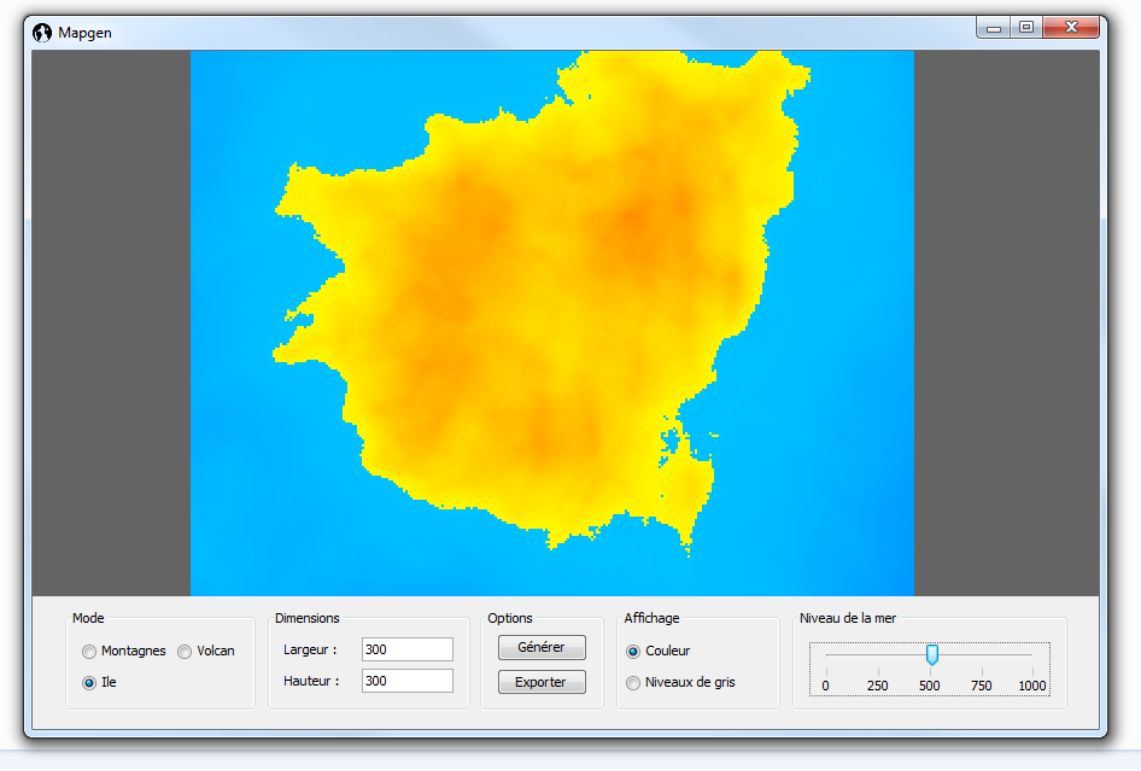
La fenêtre d’enregistrement :



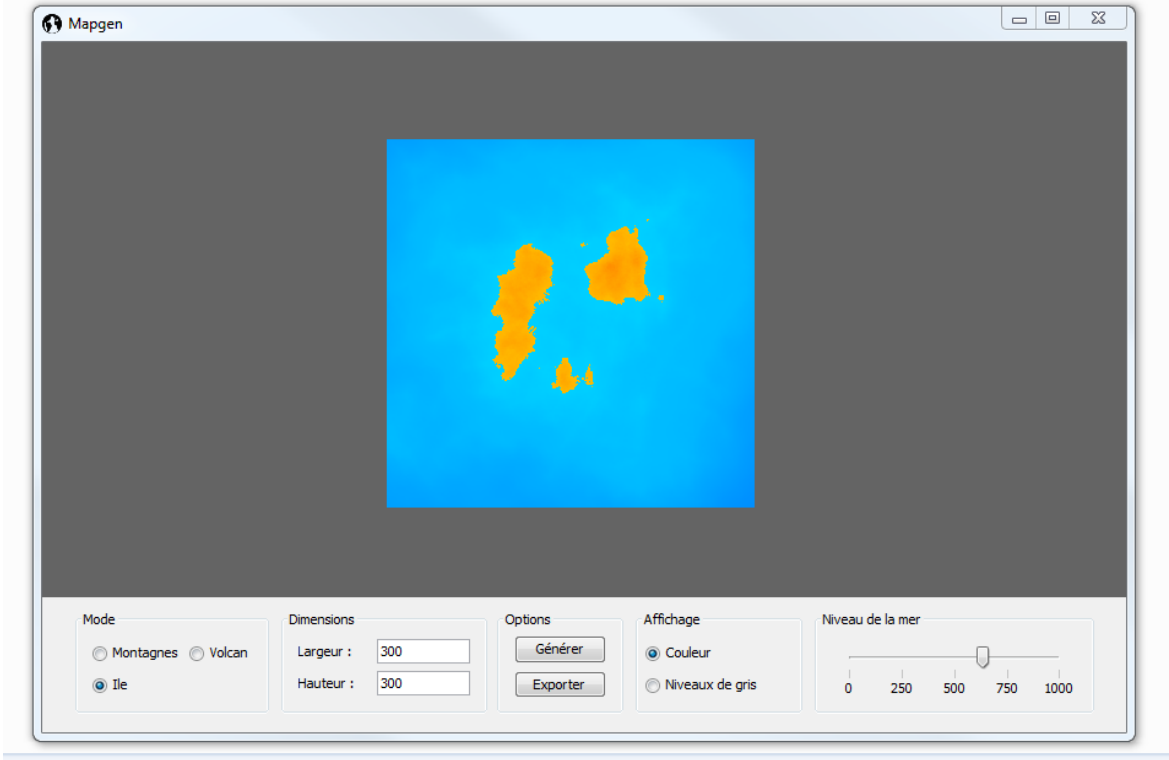
L’image déplacée avec la souris :



Zoom avec la molette :



Variation du niveau de la mer :



2. Présentation technique

2.1. Outils utilisés

Nous avons fait le choix d'utiliser Java comme langage de programmation parce que nous sommes susceptibles de l'utiliser souvent dans nos vies professionnelles. C'est de plus un langage qui nous permet de faire beaucoup de choses, avec une documentation riche, et compatible avec de nombreuses plateformes.

Nous avons aussi eu besoin d'un système de gestion de versions pour notre code et nous avons choisi Subversion qui est intégré à Netbeans, accompagné de Tortoise SVN pour restaurer d'anciens commits. Notre repository est hébergé sur le site Xp-Dev.com.

Le logiciel Terragen3 nous a permis d'obtenir des rendus 3D de nos heightmaps.
Tous nos échanges se sont déroulés via Skype.

2.2. Logique technique

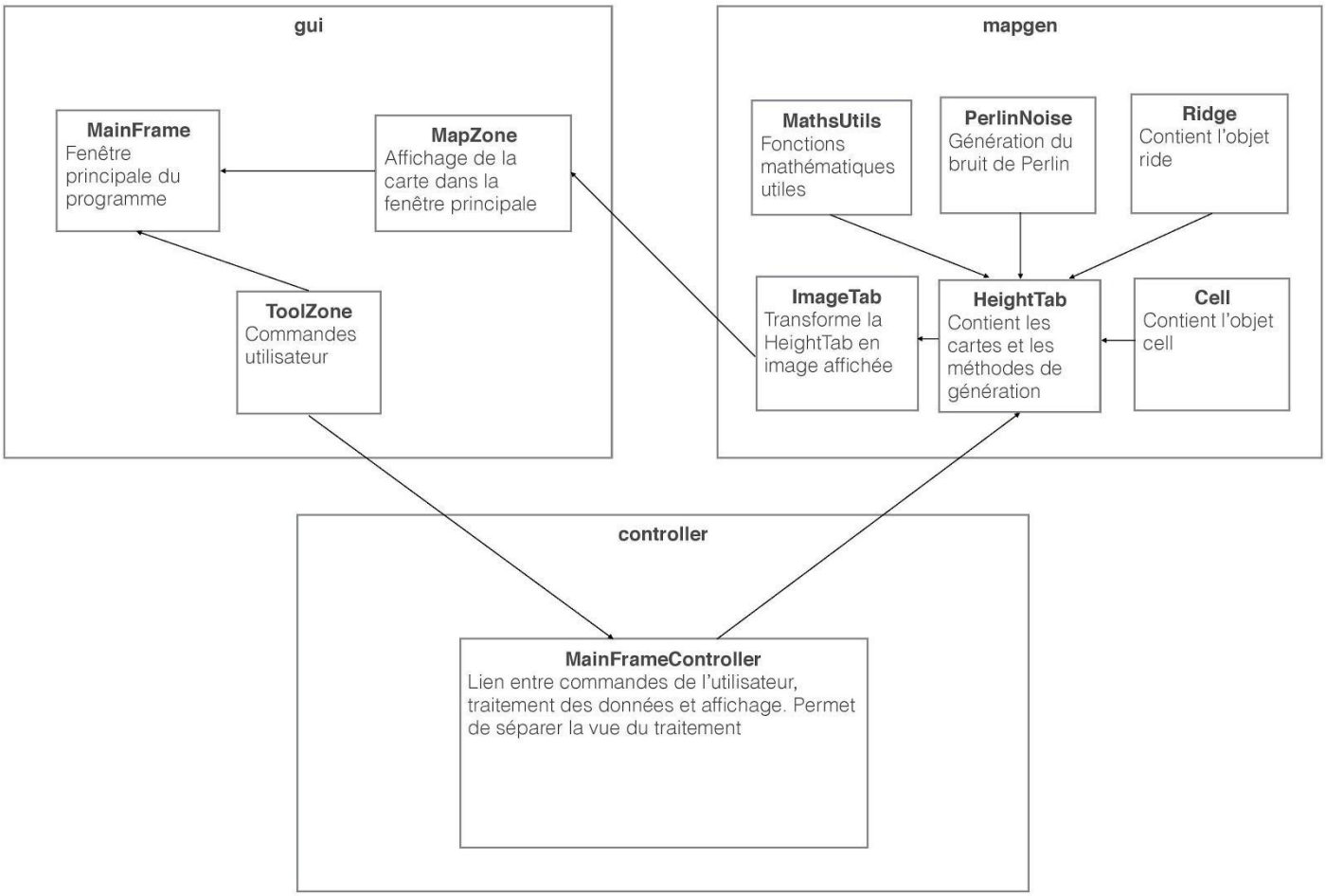
Lors de la construction de notre code nous avons adopté une logique MVC (Modèle Vue Contrôleur), en répartissant nos classes dans 3 packages :

- "mapgen" qui contient les objets métiers et leurs méthodes de traitement,
- "gui" qui contient les classes d'affichage,
- "controller" qui assure le lien entre la partie affichage et la partie traitement.

Un autre aspect auquel nous nous sommes attaché est celui de la documentation : nous avons accompagné la plupart de nos méthodes de commentaires javadoc afin d'en facilité l'utilisation. Nous avons particulièrement travaillé à la bonne documentation de la classe HeightTab, dont les méthodes sont fréquemment utilisées et comportent parfois de nombreux paramètres.

2.3. Architecture du code

Notre code est organisé de la manière suivante :



2.4. Présentation détaillée des classes

2.4.1. Package mapgen

2.4.1.1. Cell

```
public Cell(int height)
public void addHeight(int height)
```

Chaque case d'une HeightTab contient exactement un Cell. Un Cell est composé d'un attribut unique, int height, qui correspond à l'altitude du point.
Nous aurions pu directement utiliser une matrice d'entiers pour gérer nos altitudes dans HeightTab, cependant nous avons décidé de passer par cette classe dans une logique de maintenabilité et surtout d'évolutivité de notre application. Il serait ainsi très aisé de rajouter des indications pour chaque Cell concernant par exemple le type terrain, ou tout autre donnée utile à implémenter pour de nouvelles fonctionnalités.

2.4.1.2. HeightTab

```
public HeightTab(int sizeX, int sizeY)

public void generateIsland()
public void generateMountains()
public void generateVolcanos()

private void addHeightTab(HeightTab heightTab)
private void adjustHeightDelta(int newMin, int newMax)
private void alterateBeaches(int level, int range, float factor)
private void changeContrast(int percent)
private HeightTab cloneHeightTab()
private ArrayList<Ridge> findMountainRidges(int minRidges, int maxRidges, int pasMountains,
int pasHollows, int distanceBetweenTops)
private void findNextRidge(Point currentPoint, ArrayList<Ridge> ridges, ArrayList<Point>
allPoints, ArrayList<Point> highPoints)
private void generateBump(int minHeight, int maxHeight, int minRadius, int maxRadius, int
posX, int posY)
private void generateBump(int minHeight, int maxHeight, int minRadius, int maxRadius, Point
point)
private void generateBump(int minHeight, int maxHeight, int minRadius, int maxRadius, int
delta)
private void generateBumps(int number, int minHeight, int maxHeight, int minRadius, int
maxRadius, int delta)
private void generateBumps(int numberMin, int numberMax, int minHeight, int maxHeight, int
minRadius, int maxRadius, int delta)
private void generateNoise(int min, int max)
private void generateNoise()
private void normalizeHeightTab()
private void paintMountainRidges(ArrayList<Ridge> ridges, float heightFactor)
private void printHeightTab()
private int searchAverageHeight()
private ArrayList<Point> searchHighLowPoints(int pas, boolean searchHighPoints)
```

```
private Point searchHighestPoint()
private ArrayList<Point> searchHollow(int pas)
private ArrayList<Point> searchMountains(int pas)
private void substractHeightTab(HeightTab heightTab)
```

HeightTab est la classe phare de notre processus de génération. Il s'agit d'une matrice 2D de Cells.
HeightTab comprend deux grands types de méthodes :

- Les méthodes outils, qui permettent essentiellement de modifier les données de la HeightTab (génération de bruit, de bosses, addition, etc). Ces méthodes sont de portée privée, car uniquement utilisées au sein même de la classe par les méthodes de génération.
- Les méthodes de génération, elles, combinent les méthodes outils afin d'obtenir un terrain final. Ce sont ces méthodes, publiques, qui sont appelées depuis l'extérieur pour chaque nouvelle génération.
  Les méthodes de génération suivent schématiquement la logique suivante :
  - Génération de bosses (generateBumps)
  - Génération de bruit de Perlin (generatePerlinNoise)
  - Détection et tracé des arêtes montagneuses (findMountainRidges, paintMountainRidges)

Il s'agit des méthodes outils les plus importantes, mais d'autres méthodes existent. Par ailleurs, l'ordre d'appel à ces fonctions peut largement varier. Ainsi, il est fréquent de devoir générer des bosses à plusieurs étapes de la génération, et non pas une unique fois au début. Voici en illustration la méthode de génération d'île :

```
	public void generateIsland() {
//On ne passe pas les dimensions en paramètre, la methode s'applique sur la heightTab
courante
	int borderMin = sizeX;
	int borderMax = sizeY;
	if (sizeY < sizeX) {
		borderMin = sizeY;
		borderMax = sizeX;
	}
	//On génère des bosses selon la taille de la map choisie
	generateBumps((int) (borderMax * 0.065f), // nombre de bumps min
			(int) (borderMax * 0.1f),   // nombre de bumps max
			15,                         // Hauteur min d'un bump
			34,                         // Hauteur max d'un bump
			(int) (borderMin * 0.1f),   // Radius min d'un bump
			(int) (borderMin * 0.2f),   // Radius max d'un bump
			(int) (-borderMin * 0.30f));// Empêche les bumps d'apparaitre trop
                                            près des bords

	//On ajoute un Perlin faible/moyen
	HeightTab tempHeightTab = new HeightTab(sizeX, sizeY);
	tempHeightTab.generatePerlinNoise(7, 0.8); // 7 octaves, persitance à 0.8
```

```
        // On crée une HeightTab clone aux reliefs plus prononcés, qui servira pour le tracé
des arêtes
        HeightTab cloneHeightTab = cloneHeightTab();
        cloneHeightTab.addHeightTab(tempHeightTab);

        tempHeightTab.generatePerlinNoise(4, 0.8); // On écrase tempHeightTab avec un
                                                      nouveau bruit de Perlin
        tempHeightTab.adjustHeightDelta(0, 220);   // On veut des altitudes comprises
                                                        entre 0 et 220
        addHeightTab(tempHeightTab);

        //On se sert de cloneHeightTab, aux reliefs plus prononcés, pour trouver où
          tracer nos arêtes de montagnes.
        ArrayList<Ridge> ridges = cloneHeightTab.findMountainRidges(1, 3, -100, 600, 0);
        paintMountainRidges(ridges, 0.42f); // On trace les arêtes sur l'objet courant
        normalizeHeightTab();

        // On aplatie deux intervalles au pied de l'île pour former des plages
        alterateBeaches(525, 150, 0.25f);
        alterateBeaches(475, 50, 0.6f);
        }
```

Les méthodes outils ont elles nécessité un travail souvent direct sur la matrice d'altitude.
Toutes les méthodes outils ne sont pas nécessairement utilisées par notre application, nous avons cependant choisi de les conserver : ces méthodes constituent la boite à outils pour générer nos terrains, et une méthode non utilisée pour un type de terrain pourrait très bien nous être utile pour de nouvelles formes de génération. C'est par exemple le cas pour generateNoise (génération de bruit pur), changeContrast, ou encore printHeightTab (affichage en console des valeurs numériques des altitudes : méthode de débuggage).

Les méthodes de génération de collines sont parmi les méthodes outils les plus importantes. Elles sont utilisées dans de nombreuses étapes des générations. Au départ, l'idée était de créer des bosses au sens propre, avec donc un arrondi de la forme. Cependant, et contre nos attentes, une forme en cône s'est révélée plus efficace pour nos générations. La ligne de code permettant de donner une forme arrondie au bump est ainsi commentée.
Il existe plusieurs surcharges faisant appel à cette méthode.

```
private void generateBump(int minHeight, int maxHeight, int minRadius, int maxRadius, int posX, int posY) {
        Random random = new Random();

        int centralHeight, radius;
        float distance;
        int addedHeight;
        int i, j;

        if (minHeight != maxHeight) {
                centralHeight = random.nextInt(maxHeight - minHeight) + minHeight;
        } else {
                centralHeight = minHeight;
        }
```

```
        if (minRadius != maxRadius) {
                radius = random.nextInt(maxRadius - minRadius) + minRadius;
        } else {
                radius = minRadius;
        }
        for (i = posX - radius; i < posX + radius + 1; i++) {
                for (j = posY - radius; j < posY + radius + 1; j++) {
                        if ((i >= 0) && (j >= 0) && (i < this.sizeX) && (j < this.sizeY)) {
                        distance = MathUtils.calculDistance(i, j, posX, posY);
                                if (distance == 0) {
                                        addedHeight = centralHeight;
                                } else {
                                        //On détermine la hauteur à ajouter
                                        addedHeight = (int) ((1 - (distance / radius)) * centralHeight);
                                        //On transforme la hauteur à ajouter, pour un aspect bosse plutôt que
                                        cône
                                /*if (addedHeight > 0) addedHeight = (int)(addedHeight +
                                   ((centralHeight-addedHeight)*(1-(distance/radius))));*/
                                }
                                if (addedHeight >= 0) {
                                        getCell(i, j).addHeight(addedHeight);
                                }
                        }
                }
        }
}
```

2.4.1.3. ImageTab

```
public ImageTab(HeightTab heightTab)
public void convertToColorBufferedImage()
public void convertToGreyBufferedImage()
public void exportWindow(String typeMap)
```

ImageTab est la classe de conversion d'une HeightTab en BufferedImage. Lorsqu'un terrain est généré, une instance de ImageTab à laquelle on passe en argument ce HeightTab est créée.
Le constructeur de ImageTab transforme alors immédiatement la HeightTab en bufferedImage, version couleur et niveaux de gris.
Nous procédons ainsi car il est fréquent de vouloir visualiser les deux versions d'un terrain : ainsi, une fois le terrain généré, les deux versions sont immédiatement prêtes à la visualisation.
ImageTab contient également la méthode d'export des rendus (.png, .jpg, .gif)

Voici la méthode convertToColorBufferedImage :

```
public void convertToColorBufferedImage() {
        BufferedImage bufferedImage = new BufferedImage(heightTab.getSizeX(),
                                        heightTab.getSizeY(), BufferedImage.TYPE_INT_RGB);
        Color color;
        int rgbValue;
        int cellHeight;
```

```
        int red = 0, green = 0, blue = 0;

        for (int x = 0; x < bufferedImage.getWidth(); x++) {
                for (int y = 0; y < bufferedImage.getHeight(); y++) {
                        cellHeight = heightTab.getCell(x, y).getHeight();
                        cellHeight *= 0.255; //On recentre cellHeight entre 0 et 255 pour
                                             transformer facilement l'altitude en couleurs

                        if (cellHeight <= 127) {
                                red = 255 - (int)((127-cellHeight)*1.5);
                                green = 255;
                                blue = 0;
                        } else {
                                red = 255;
                                green = (int)((255-cellHeight)*2);
                                blue = 0;
                        }
                        //Affichage de l'eau
                        if ((seaLevel > 0) && (cellHeight <= seaLevel * 0.255)) {
                                red = 0;
                                green = 128 + cellHeight / 2;
                                blue = 255;
                        }
                        color = new Color(red, green, blue);
                        rgbValue = color.getRGB();
                        bufferedImage.setRGB(x, y, rgbValue);
                }
        }
        colorBufferedImage = bufferedImage;
}
```

2.4.1.4. MathUtils

```
public static float calculDistance(int X1, int X2, int Y1, int Y2)
public static float calculDistance(Point posA, Point posB)
public static boolean isPointUnderLimits(Point point, int xMin, int xMax, int yMin, int yMax)
```

MathUtils est une classe utilitaire qui regroupe des méthodes, statiques, utilisées fréquemment. L'appel à ces méthodes permet une plus grande clarté du code ainsi que d'éviter des duplications de code.

2.4.1.5. PerlinNoise

```
public static HeightTab generatePerlinHeightTab(int largeur, int hauteur, int nombreOctaves,
double persistance)

private static double bruit2D(int i, int j)
private static double bruit_coherent2D(double x, double y, double persistance)
private static double fonction_bruit2D(double x, double y)
private static void initBruit2D(int l, int h, int nombreOctaves)
private static double interpolation_cos1D(double a, double b, double x)
private static double interpolation_cos2D(double a, double b, double c, double d, double x,
double y)
```

Le bruit de Perlin est complexe à générer, et nécessite de nombreuses méthodes propres. Nous avons donc décidé de créer une classe spécialement pour la génération de ce bruit pour ne pas surcharger inutilement la classe HeightTab. La méthode generatePerlinNoise de HeightTab appelle ainsi la méthode statique generatePerlinHeightTab de PerlinNoise, qui elle même utilise les autres méthodes de la classe pour renvoyer une HeightTab de bruit de Perlin.

2.4.1.6. Ridge

```
public Ridge(Point startCell, Point endCell, int nbPoints)
public Ridge(Point startCell, Point endCell)
public final void findRidgePoints(int nbPoints)
```

Ridge est une classe représentant un segment d'un point A vers un point B.
Elle contient également la distance entre ces deux points, et un ArrayList<Point> qui permet de stocker des points correspondant à des emplacements sur ce segment, également espacés entre eux.
Un premier constructeur permet donc de définir simplement un point A et un point B, tandis qu'un deuxième fait appel à findRidgePoints pour trouver nbPoints intermédiaires entre A et B. Demander un nombre de points relatif à la longueur du Ridge permet un espacement des points identique entre plusieurs Ridges, quelque soit leur longueur.
Les Ridges servent ainsi à positionner des Bumps sur notre terrain le long de segments.
Voici la méthode findRidgePoints, qui peut être appelée dans le constructeur ou ultérieurement.

```
public final void findRidgePoints(int nbPoints){
        ridgePoints.clear();
        float distanceX = (endCell.x-startCell.x);
        float distanceY = (endCell.y-startCell.y);
        float gapX = distanceX/(nbPoints+1);
        float gapY = distanceY/(nbPoints+1);

        float deltaX, deltaY;

        for (int i = 1; i <= nbPoints; i++){
                deltaX = gapX*i;
                deltaY = gapY*i;

                Point point = new Point();
                point.x = (int)(startCell.x + deltaX);
                point.y = (int)(startCell.y + deltaY);
                ridgePoints.add(point);
        }

}
```

2.4.2. Package gui

2.4.2.1. MainFrame

```
public void choiceMap(ImageTab imageTab , String render)
```

MainFrame est utilisée pour afficher la fenêtre principale du programme. On utilise la classe ToolZone pour afficher la barre d'outils au bas de la fenêtre.

2.4.2.2. Mapzone

```
public MapZone()
public MapZone(BufferedImage bufferedImage)
public void translateCanvas(int valX, int valY)
```

C'est ici que l'on affiche la carte, en gérant la position et la taille à laquelle l'image doit être affichée selon l'état du zoom et du glisser-déposer.

2.4.2.3. ToolZone

Classe créée via l'outil Design de NetBeans.
La classe ToolZone gère la création et l'affichage de la barre d'outils au bas de la fenêtre principale. On crée dans ce panel cinq autres panels correspondant aux parties Mode, Dimensions, Options, Affichage et Niveau de la mer. Ces panels comprennent les boutons, slider et champs de texte nécessaires aux contrôles de la carte générée.

2.4.3. Package controller

2.4.3.1. MainFrameControler

Cette classe permet de gérer les actions de l'utilisateur et de les répercuter sur l'interface. C'est ici que l'on prend en compte les mouvements de zoom et de glisser-déposer de la souris avec les sous-classes DragDropAdapter et Zoom, ainsi que les clics sur les boutons de la ToolZone.

DragDropAdapter prend en entrée la position du curseur de la souris et modifie ensuite la position de l'image.
Zoom prend en entrée le nombre de tours de molettes de la souris et répercute cette valeur (positive ou négative selon le sens de rotation) dans une variable steps. Nous passons ensuite steps à la classe MapZone pour connaître le niveau de zoom à appliquer lors du rendu de l'image.
MainFrameControler contient de nombreux ActionPerformed et on récupère à de nombreuses reprises l'état des boutons et champs de texte de la barre d'outils du programme (ToolZone).

3. Déroulement et évolution du projet

3.1. Organisation et répartition des taches

Ce projet étant le plus important que nous ayons eu à faire au cours de notre cursus en termes de durée et de travail à fournir. La question de l'organisation et de la répartition des tâches a été cruciale pour pouvoir progresser correctement.
Durant la phase de conception, nous avons travaillé en groupe sans réelle répartition du travail. Nous désirions en effet démarrer sur une compréhension complète du projet par chacun, et une conception née d'un effort collectif.

Cette première étape terminée, nous avons réparti le travail entre les membres du groupe, en essayant de respecter au maximum les préférences de chacun. Dans cette optique, et en suivant les conseils de Mme Ferreira Da Silva et de Mr Peytavie, nous avons mis en place un agenda avec la répartition des tâches entre les membres du groupe (voir Annexes). Cependant, si ceci nous a permis de nous organiser sur le moment et d'appréhender une organisation sur le long terme, nous avons du faire quelques écarts lorsque de nouvelles idées sont apparues au cours du développement. En effet, nos prévisions ont omises des méthodes, mais nous avons surtout d'une manière générale prévu des délais plus importants que le temps qu'il nous aura réellement fallu.

Concernant la répartition effective des tâches :

- Hugo Talbot est principalement intervenu sur l'interface graphique et sur la classe MainFrameController. Les points les plus importants de son travail ont notamment été le zoom, le déplacement de l'image, l'export des images (exportWindow() de la classe ImageTab) et le bon fonctionnement général des réponses aux entrées utilisateurs. Participation également à la méthode de génération d'îles.

- Jonathan Jobkel a travaillé sur les trois aspects du programme. Il a participé au développement de méthodes sur lesquelles travaillait Hugo (zoom, déplacemement, affichage et interaction) ainsi que certaines méthodes de génération (addition et soustraction de HeightTabs, bruit de Perlin), avec en particulier les méthodes de recherche de points hauts et bas (searchMountains() et searchHollows()).

- Vincent Delannoy a principalement travaillé sur les méthodes de génération (package mapgen). Les points les plus notables ont été les méthodes finales de génération (montagnes, île, volcan), la génération des collines (generateBump()), la conversion des HeightTabs en images, et la détection et tracé des arêtes des montagnes.

Cette répartition des tâches a ainsi permis à chacun de se concentrer sur les points qui l'intéressaient le plus. Néanmoins, les entraides ont été fréquentes et nous nous sommes assurés à ce que chacun ait une bonne compréhension de l'ensemble des aspects du code.

Le travail lui même s'est le plus souvent organisé en sessions collectives de travail d'environ 3 à 4 heures, en moyenne deux fois par semaines. Ceci nous a notamment permis d'avancer le projet de manière commune, et de respecter les principes d'entraide et de compréhension du code énoncé ci-dessus.
Par ailleurs, cette organisation nous a permis de maintenir un avancement et un contact régulier, Hugo Talbot en particulier étant en alternance et ayant donc un emploi du temps différent du reste du groupe.

3.2. Problèmes rencontrés et résolution

Nous avons rencontré plusieurs problèmes importants durant le développement de notre application :

- Implémentation du glisser-déposer

Nous ne savions pas comment implémenter le glisser-déposer de l'image et nous avons fait appel à Julien Delannoy qui nous appris à utiliser les mouseListener et à répercuter les mouvements de la souris sur la position de l'image. Nous n'arrivions pas à faire transiter les informations entre le MainFrameControler et MapZone, ce qui empêchait l'image d'être déplacée.

- Modification difficile de l'interface

Lors de la création de la barre d'outils nous n'avions pas une grande connaissance de NetBeans et de Java, nous l'avions donc implémenté manuellement dans une méthode de MainFrame. Ce n'est que très récemment que nous avons appris l'existence des classes Java héritant de JPanel et JFrame (JFrame Form et JPanel Form) qui nous permettent de modifier très rapidement et efficacement notre interface via l'outil "Design" de NetBeans.
Nous avons donc recréé notre barre d'outils (classe ToolZone) dans un JPanel Form, qui est ensuite ajouté à notre classe MainFrame.

- Le tracé des montagnes

Les montagnes ne sont pas uniquement des zones plus hautes que d'autres, et il nous a fallu leur tracer des arêtes pour qu'elles n'apparaissent pas comme de simples bosses. Cette étape a été l'une des plus délicates de la génération, et probablement celle que nous redoutions le plus.
Nous avons choisi de procéder de la manière suivante :

- Notre première étape a été de détecter des points 'hauts' et des points 'bas' d'une Heightab. Autrement dit, détecter des points qui pouvaient s'apparenter à des sommets de montagnes ou bien à des creux. Les méthodes searchMountains et searchHollow remplissent cette mission, en prenant en paramètre un pas nous permettant de moduler la quantité de points désirés.
- La deuxième étape a été de tracer, via les points hauts et bas trouvés précédemment, des Ridges, autrement dit des segments correspondant à des arêtes à tracer. Les ridges ainsi instanciés sont stockés et renvoyés dans une ArrayList<Ridge>. C'est la méthode findMountainRidges qui, en partant des points les plus hauts, va chercher à déterminer les Ridges des montagnes sur l'ensemble d'une HeightTab. Elle utilise la méthode récursive findNextRidge qui, depuis un point, va chercher à déterminer le point suivant pour y tracer un nouveau Ridge.
- La dernière étape consiste enfin à tracer effectivement les arêtes sur le HeightTab à partir des Ridges détectés. La méthode paintMountainRidges effectue ce travail.

3.3. Chronologie

3.3.1. Conception et mise en route du projet

Nous étions au départ un groupe de quatre étudiants sur ce projet. Le quatrième membre, Mathieu Plat, a participé à la phase de conception, en recherchant avec nous les différentes manières de générer un terrain sous forme d'image 2D. Il a ensuite quitté l'IUT peu après.

Nous avions envisagé trois approches pour notre algorithme de génération :

- Approche par influenceurs

Cette approche consistait à envisager le terrain sous forme d'une grille d'altitudes. Sur cette grille auraient été placés de manière aléatoire des influenceurs, c'est à dire des points modelant les altitudes proches. Certains points auraient nivelés l'altitude vers le haut ou vers le bas de manière importante, afin de former ainsi des reliefs. D'autres influenceurs de force moindre auraient permis d'ajouter une granularité au terrain.
Cette approche nous a néanmoins parue complexe et risquée à mettre en œuvre. Cette conception restait en effet assez abstraite et présentait encore de nombreux aspects incertains.

- Approche vectorielle

Il s'agissait ici d'imaginer le terrain comme un ensemble de points qui, reliés entre eux, auraient formé des zones d'altitude. Ces zones d'altitudes, progressivement subdivisées, auraient permis un affinage progressif du décor avec des zones de plus en plus petites et cohérentes entre elles.

- Approche matricielle

Cette approche, que nous avons adoptée, consiste à travailler directement sur une matrice d'entiers, matrice ensuite convertie en image. Le travail sur la matrice se fait via des générations de bruit, des ajouts et des soustractions d'altitudes.

Nous avons retenu cette approche matricielle, puisqu'elle nous semblait la plus propice à nous garantir des résultats pour notre projet, en présentant moins de zones d'incertitudes que les autres approches. La correspondance avec l'image finale est par ailleurs directe, et le travail sur matrice ne nous était pas inconnu.
Enfin, M. Peytavie nous a indiqué que cette approche serait certainement la plus adaptée pour notre projet.

3.3.2. Développement de l'application

Nous avons commencé par réfléchir à l'interface que nous souhaitions obtenir, et nous avons dessiné de nombreux schémas pour chaque fenêtre de l'application.
Nous avons rapidement fait le choix de créer une barre d'outils fixe avec tous les boutons accessibles au bas de l'écran, plutôt qu'un menu d'options "classique" avec des menus déroulants alignés horizontalement. Nous voulions que l'utilisateur puisse avoir une vue pleine et entière des contrôles qu'il peut effectuer sur la génération.

Pour la partie génération, nous avons visualisé le fonctionnement de notre logique de génération à l'aide du logiciel d'édition d'image Gimp. Cela nous a permis d'appréhender un fonctionnement par étape de la génération, et nous avons ainsi été capables de créer les méthodes outils de HeightTab en conséquence.

Nous avons commencé le code par la partie graphique, d'abord très élémentaire. Nous avons rapidement ajouté une méthode très basique de génération, du bruit pur, et géré sa conversion en bufferedImage pour son affichage dans MapZone. Ceci nous a permis d'obtenir rapidement une interface et une génération opérationnelles, qui ont dès lors pu évoluer de manière séparée tout en laissant l'application fonctionnelle. Nous avons aussi pu commencer à nous répartir davantage le travail.
Nous avons été amenés, sur les conseils de Julien Delannoy, à adopter un modèle MVC pour l'organisation de nos classes. Cette organisation technique nous a permis de rendre le code plus clair et plus facilement maintenable. Nous avons ensuite travaillé en parallèle en codant simultanément la partie génération et la partie interface utilisateur.

3.4. Bilan vis-à-vis du cahier des charges

Notre application finale nous parait respecter les spécifications du cahier des charges :

- Génération d'un terrain cohérent de type montagneux,
- Conversion et affichage en image : en niveaux de gris ou en couleur
- Gestion d'une option de zoom/dézoom de l'image
- Image aux dimensions demandées par l'utilisateur
- Export aux formats d'image .png, .jpg et .gif en version couleur ou niveaux de gris.
- Utilisation de l'image en niveaux de gris comme heightmap par des logiciels tiers (dans notre cas, Terragen3).

Nous avons par ailleurs pu apporter certaines améliorations supplémentaires :

- Des modes de génération supplémentaires : génération d'île et génération de volcan.
- Choix et affichage interactif du niveau de la mer pour la version couleur
- Implémentation d'un glisser-déposer de l'image, complémentaire au zoom

3.5. Possibilités d'évolution

Des évolutions possibles du programme sont :

- Ajout de modes de génération supplémentaires
- Support du format d'image RAW
- Un nouveau moded'affichage, en plus de couleur et niveau de gris, avec des couleurs plus naturelles

Conclusion

Hugo : Ce projet m'a permis de développer ma maîtrise du Java, d'appréhender le modèle MVC que je ne connaissais pas, de créer pour la première fois une application informatique de A à Z. J'ai beaucoup aimé travailler sur l'interface utilisateur, en particulier du point de vue de l'ergonomie de l'application. J'ai beaucoup apprécié le fait de travailler en équipe, ce qui m'a beaucoup motivé, j'ai pu ainsi donner le meilleur de moi-même lors de ce projet, ce que je n'aurais peut-être pas été capable de faire si j'avais travaillé seul. A l'issue de ce projet, je constate que la création d'une application informatique nécessite une logique et une organisation importante afin d'obtenir une bonne cohésion au sein du code pour un bon fonctionnement du programme.

Jonathan : Le projet tuteuré m'a apporté de nouvelles méthodes de travail, en effet j'ai pu travailler en groupe en utilisant les méthodes agile vues en cours, ce qui est très pratique et aide à la cohésion de groupe. J'ai aussi pu me familiariser avec le Modèle Vue-Contrôleur (MVC) qui apporte une grand lisibilité du code de par sa structure en modules, les notions vues en cours au niveau algorithmique m'ont également beaucoup servies. Je me suis également beaucoup amélioré en Java au fil de l'avancement de notre projet tuteuré.
Finalement tout cela m'as permis de participer à l'élaboration d'un logiciel en passant par tout les aspects, de sa conception à sa réalisation.

Vincent : Je pense avoir beaucoup appris de ce projet, conséquent à notre échelle. Il a été l'occasion d'un travail long de plusieurs mois, et mené de sa conception à sa réalisation. Il m'a permis de mieux mesurer l'importance du travail de conception, qui peut-être n'avait pas été assez recherché dans un premier temps. Ce fut aussi l'occasion d'approfondir mes connaissances techniques en Java mais également, et plus particulièrement, en conception logicielle. Enfin c'est en termes de travail d'équipe et d'organisation sur une longue période que ce projet a été très instructif à mes yeux.
Participer à ce projet tuteuré aura ainsi été particulièrement enthousiasmant pour moi, et m'aura permis d'avoir une vision plus claire du travail et de l'organisation nécessaires à la conduite de tels projets de développement.