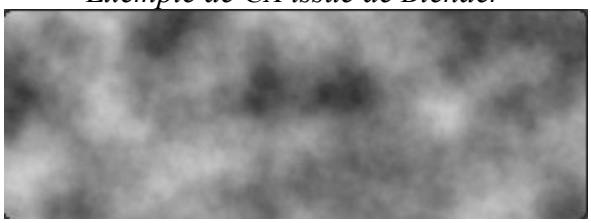
CREATION D'IMAGES 2D "HEIGH MAP" AVEC L'ALGORITHME DIAMOND-SQUARE

De plus en plus de logiciel de modélisation d'images 3D existent et sont de bonne qualité (Blender, Maya, Terragen, Bryce, Catia, SolidWorks, etc...), il est possible maintenant avec certains logiciel très pointu de créer des paysages 3D très réaliste à partir de données 'topographiques' complètement issue d'algorithme de simulation de terrains ou de ciel nuageux (Terragen par exemple), les jeux ne sont pas en reste et utilisent également ces simulateurs de terrains (Jmonkey, Unity), ces éléments à la base de paysages intéressant se nomment les "Heigh Map" (HM) ou carte d'altitudes (CA), ce sont des matrices de points qui permettent quand ces points sont uniformément et intelligemment réparties, de simuler des hauteurs et des plats (respectivement des creux, collines et montagnes).

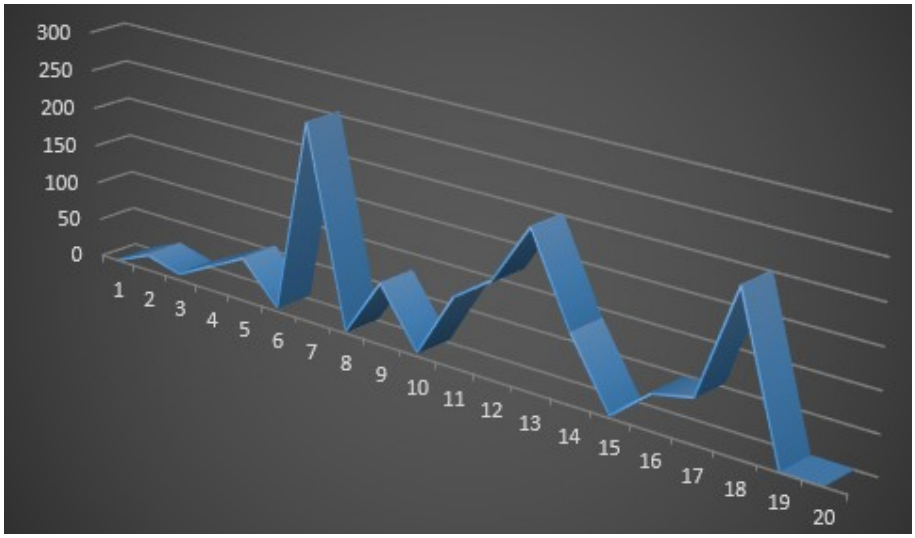
Nous nommerons ces éléments des cartes d'altitudes (CA), dont on représentera le point le plus bas par une valeur équivalente à 0 (en général la couleur noire) et le point le plus haut par un valeur de 1 au maximum (la couleur blanche) on peut également en faire une représentation binaire et coder ces hauteurs sur 8 (0 à 255), 16 (0 à 65535) ou 32 (0 à 4294967295) bits, ceci dépendra de la précision que l'on désire obtenir sur l'axe Z(hauteur) et de la représentation et des conventions de notre monde.

Exemple de CA issue de Blender

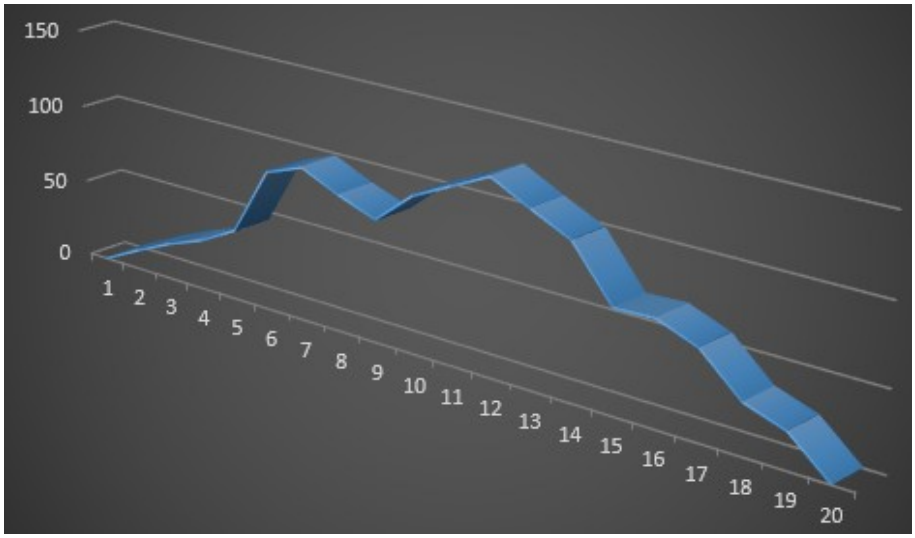


Pour générer ce genre de matrice de points et avoir une répartition des niveaux de gris la plus équilibrée possible(*) on peut utiliser plusieurs types d'algorithmes, l'un des plus connu étant celui du "Bruit de Perlin", il existe bien d'autres algorithmes basé sur la notion de bruit.

(*)Ci dessous une vue qui montre une mauvaise répartition des hauteurs, cela donne un terrain chaotique



Avec une meilleure répartition, on obtient une représentation plus réaliste du monde.



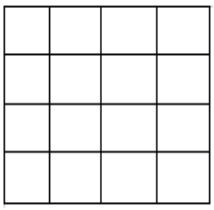
Il existe également des algorithmes basés sur le placement des points, c'est l'un d'entre eux que l'on va se mettre au défi d'utiliser pour générer nos CA. L'algorithme "Diamond-Square", est simple de compréhension et simple a mettre en œuvre et donne d'assez bon résultats.

L'algorithme Diamond-Square :
L'algorithme « DiamondSquare », comme son nom l'indique, se base sur 2 phases : la phase du carré et la phase du diamant. Il s'agit d'itérer ces 2 phases jusqu'à calculer tous les points de la matrice représentant le terrain 3D.

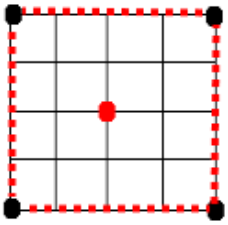
En effet, la largeur de la matrice représente les abscisses(X), la longueur, les ordonnées(Y) et enfin chaque "case" de la matrice représente la hauteur(Z) relative du point :

pour accéder à une case de la matrice on utilsera tab[x][y] = z;

Prenons un exemple simple pour expliquer l'algorithme. Soit une matrice 5X5 :

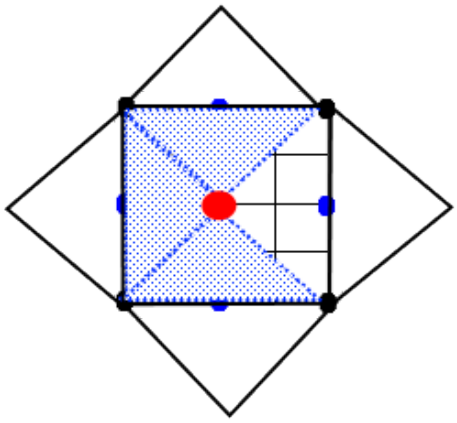


La phase du carré consiste à élever le centre du carré d'une valeur aléatoire (le point en rouge) :

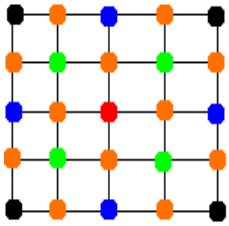


La phase du diamant consiste à élever le centre de chaque losange formé par le centre et les coins

du carré précédent d'une valeur aléatoire (les points en bleu). On peut constater déjà qu'on se retrouve confronté à un problème : il n'y a pas quatre points complet pour former le diamant. Il faut donc tenir compte du cas particulier des bords :



On réitère avec les carrés obtenus(vert),puis les diamants(orange) et ainsi de suite jusqu'à parcourir l'ensemble de la matrice:



On obtient donc l'algorithme suivant :

```
espace = Largeur;
tant que espace > 1 faire
{
  espace = espace / 2;

  pour chaque carre de largeur espace faire
  {
    etape du carre;
  }

  pour chaque diamant de largeur espace faire
  {
    etape du diamant;
  }
}
```

Prérequis à cela :
1) Il faut obligatoirement que la matrice ait pour largeur 2^n+1, sinon le calcul du centre serait faux, et l'algorithme ne marcherait pas.

2) Comme le terrain est aléatoire, la hauteur d'un point doit être choisit au hasard. Néanmoins, pour ne pas avoir n'importe quoi, il faut quand même contrôler sa valeur. On parle alors de hauteur pseudo aléatoire car cela n'est pas totalement dû au hasard.

Tout d'abord, la hauteur d'un point est calculée en fonction de la moyenne de la hauteur des points que l'algorithme a utilisé pour l'obtenir.

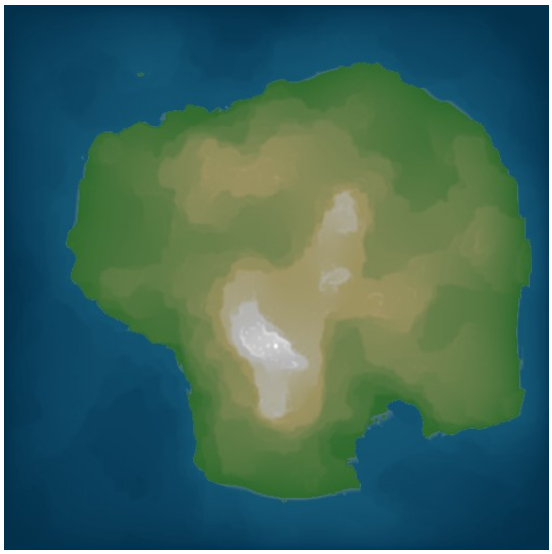
On y ajoute une valeur aléatoire (supposée pure), à laquelle on applique un coefficient de lissage

pour ne pas avoir un terrain en dents de scie. De plus, on multiplie à cette valeur aléatoire pure, l'espace entre le point cible et les points sources, afin de garder la cohérence du terrain. Voici donc une formule pour obtenir la hauteur :

```
hauteur = moyenne() + random() * espace * lissage
```

En complément du générateur :

On essayera suite à la génération de notre CA, d'afficher l'image de celui-ci en couleur en utilisant un gradient de couleurs de votre choix qui permettra d'obtenir un résultat de ce type (voir ci-dessous), on pourra jouer sur ce gradient pour placer plus ou moins haut le niveau de la mer.



Le travail à réaliser :

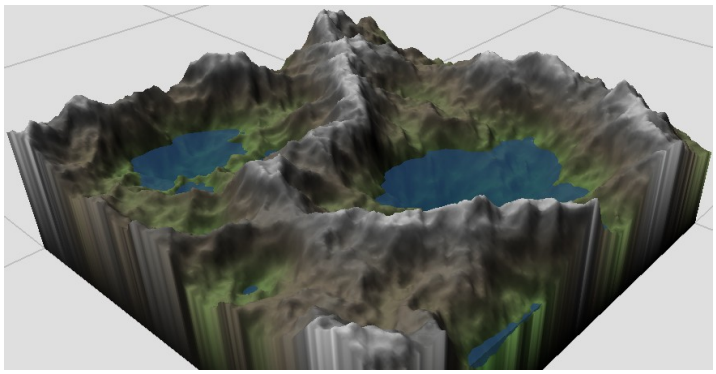
Est de développer une application Java avec le FrameWork JavaFX qui permettra de générer des CA dont on pourra choisir ses paramètres (la taille par exemple) selon l'algorithme "Diamond-Square".

Cette application permettra également de charger un gradient de référence et afficher notre HM en utilisant ce gradient de couleurs pour appliquer un gradient de couleurs.

Elle devra permettre également de sauvegarder le HM sous forme d'une image au format "PNG", et également sauvegarder notre rendu d'image couleurs sous un format "PNG" également.

Pour les plus téméraire, il est possible de donner du relief à notre image en plaçant un effet de lumière et en utilisant un algorithme de "Relief Shading", car en effet notre rendu couleur semblera un peu plat.

Ou aller plus loin et afficher notre CA en 3D...



Outils imposés :
Java 8 (http://www.oracle.com/technetwork/java/javase/downloads/index.html) **ET** l'Editeur NetBeans 8.1 pour Java (https://netbeans.org/downloads/)

Contact :

M. Tondeur Hervé : herve.tondeur@univ-valenciennes.fr

Répartition des charges :

Ce projet nécessite un ou deux étudiants maximum.

Motivations :

Découvrir l'informatique sous un angle algorithmique et ludique, comprendre et percevoir la richesse des algorithmes qui se trouvent masqués dans les outils de modélisations et les jeux 3D.

Approfondir ses notions de Java dans l'utilisation du langage Java et des API JavaFX.