[HAI809I - Projet] Compte Rendu N°3

Yahnis Saint-Val (yahnis.saint-val@etu.umontpellier.fr)

Léa Serrano (lea.serrano@etu.umontpellier.fr)

10 mars 2023

Génération procédurale de cartes d'environnement

1 Data Manager

Avant de commencer à implémenter la génération des différentes cartes d'environnement, il nous faut un moyen de centraliser et organiser les différents paramètres de génération.
C'est le rôle du singleton "DataManager" :

```
class DataManager {
public:
    map<string,double> values;
...
```

DataManager utilise un dictionnaire qui associe des identifiants (string) à des valeurs (double).
Une méthode "requestValue(id)" permet de récupérer une valeur via son identifiant, et si elle n'est pas encore définie, de demander à l'utilisateur de la saisir :

```
double requestValue(string name){
    if(values.count(name) <= 0){
      cout << ">> Enter <" << name << "> : ";
      double v; scanf("%lf",&v);
      setValue(name,v);
    } else {
      cout << ...
    }
    return getValue(name);
}
```

Du côté de ProjectManager, une méthode "saveData()" permet de sauvegarder les données de DataManager dans le fichier "project.txt" du projet actuellement ouvert.
Lors du chargement d'un projet existant, une méthode "loadData()" permet de récupérer les données et de les charger dans DataManager.

DataManager va donc nous permettre de centraliser les paramètres de génération, d'y accéder de manière simple tout en s'assurant que l'utilisateur puisse les choisir, et de les enregistrer pour ne pas les perdre lorsqu'un projet est fermé.

2 Requête de carte

De la même manière, nous avons ajouté une méthode "requestMap(id)" à la classe MapManager qui permet de récupérer la carte correspondant à l'identifiant donné.
Cette méthode vérifie si la carte demandée existe déjà : si oui, elle la retourne, si non, elle appelle la méthode de génération correspondante afin de générer la carte, puis la retourne :

```
ImageBase* requestMap(string id){
    if(maps.count(id) > 0){
        return maps[id];
    } else {
        return generateMap(id);
    }
}
```

La méthode "generateMap(id)" utilise un switch pour rediriger vers la méthode de génération correspondante à l'identifiant donné :

```
ImageBase* MapManager::generateMap(string id){
    switch(hash_djb2a(id)){
        case "HEIGHT_BASE"_sh: maps[id] = HeighMap::generateHeightMap(); break;
        case "HEIGHT_SEA"_sh: maps[id] = HeighMap::seuilHeightMap(requestMap("HEIGHT_BASE")); break;
    }
    return maps[id] ;
}
```

Le C++ ne permet pas directement d'utiliser un switch sur des string ; nous avons utiliser un système de hashage (que nous avons trouvé ICI).

On peut ainsi instaurer un système de dépendance entre les cartes : par exemple, dans la code ci-dessus, lorsque l'on cherche à générer "HEIGHT_SEA", le programme lance une nouvelle requête pour obtenir "HEIGHT_BASE", qui va alors être générée si besoin.

3 Génération d'une première carte d'altitude

Pour gérer et centraliser tout le code relatif aux cartes d'altitude, nous avons créé une nouvelle classe statique "HeightMap".
Pour le moment, elle comprend deux méthodes : "baseMap()", qui génère la carte d'altitude de base, et "seaMap()", qui applique un traitement sur la carte de base afin d'ajouter des "plages" et ainsi séparer la mer de la terre de façon plus douce.
La méthode "baseMap()" est paramétrable : c'est à ce moment que l'utilisateur doit choisir la taille en pixel des cartes, ainsi que la taille en kilomètres :

FIGURE 1 – Exemple d'une carte d'altitude de base, 2048*2048 pixel, et de 100*100 km

Ensuite, pour générer la carte avec les plages et la mer, la méthode "seaMap()" va demander trois paramètres à l'utilisateur :

- "sea_level", qui indique à quel niveau se trouve la mer (en terme de niveau de gris, entre 0 et 1).
- "sea_slope", qui indique la courbure du sol de la mer.
- "shore_width", qui indique l'épaisseur des "rives", c'est à dire la partie de la plage qui est immergée.

L'algorithme applique 3 transformations différentes sur la carte d'altitude (voir fonction "seaCurve()") :

- Si le terrain est au dessus du niveau de la mer, on ne fait rien ;
- Si le terrain est en dessous du niveau de la mer, mais au dessus du niveau de rive (valeur de "shore_width"), on applique une racine (valeur à la puissance "1 / sea_slope"), ce qui va courber "vers le haut" la pente du terrain, et ainsi simuler le relief des rives de la mer qui est généralement assez plat ;
- Si le terrain est en dessous du niveau de la mer et en dessous du niveau de rive, on applique une puissance (égale à sea_slope), ce qui va courber "vers le bas" la pente, et ainsi simuler le relief en haute-mer, qui a tendance à être très pentu avant de s'aplatir.

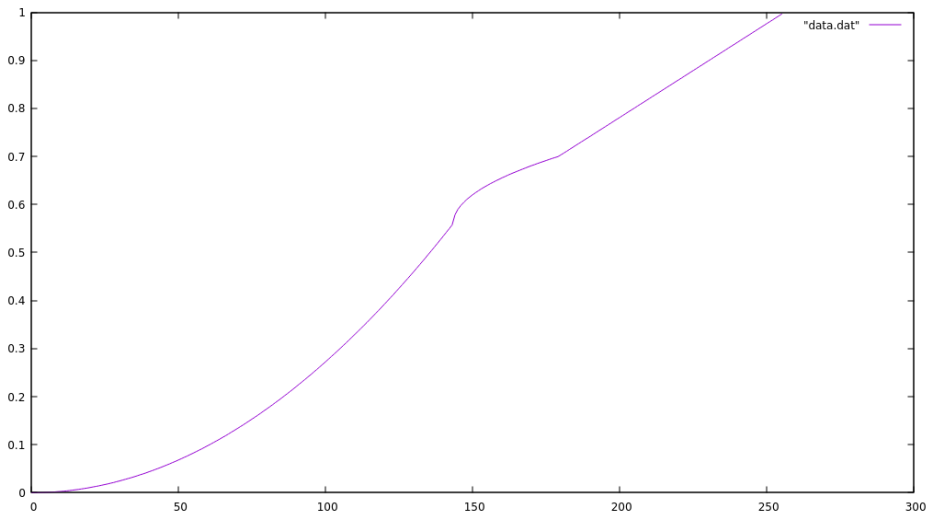

FIGURE 2 – Courbure du terrain avec pour paramètre : "sea_level" = 0.65 , "sea_slope" = 2 et "shore_width" = 0.15

On peut voir ici que le terrain reste inchangé au dessus de 0.65 (niveau de la mer), qu'il est "aplati" au niveau des rives, puis est très pentu en dessous du niveau des rives.

FIGURE 3 – Carte d'altitude avec "sea_level" = 0.5 , "sea_slope" = 2 et "shore_width" = 0.1

4 Objectifs pour la semaine prochaine

Nous allons terminer d'implémenter le traitement de la carte d'altitude : on va appliquer un traitement à la partie émergée du terrain, de la même manière que pour la partie immergée, de sorte à obtenir des reliefs plus intéressants (donc une carte moins "lisse"), par exemple des "plages" plus plates, des plaines et des falaises, ect...

Ensuite, on pourra générer une carte de "reliefs" qui indique quels types de relief se trouve à quels endrois (plaines, montagnes, falaises, ...) (en utilisant entre autres une carte de gradient), ce qui pourra ensuite permettre d'ajouter différents bruits sur la carte d'altitude (par exemple des "bosses" sur certaines plaines, des "ravins" aux niveau des falaises, ect...).

On pourra aussi éventuellement commencer à ajouter de la couleur sur nos cartes. Pour cela on pourra faire un système de dictionnaire qui, pour chaque type de relief (mer, plage, montagne), va associer une couleur.