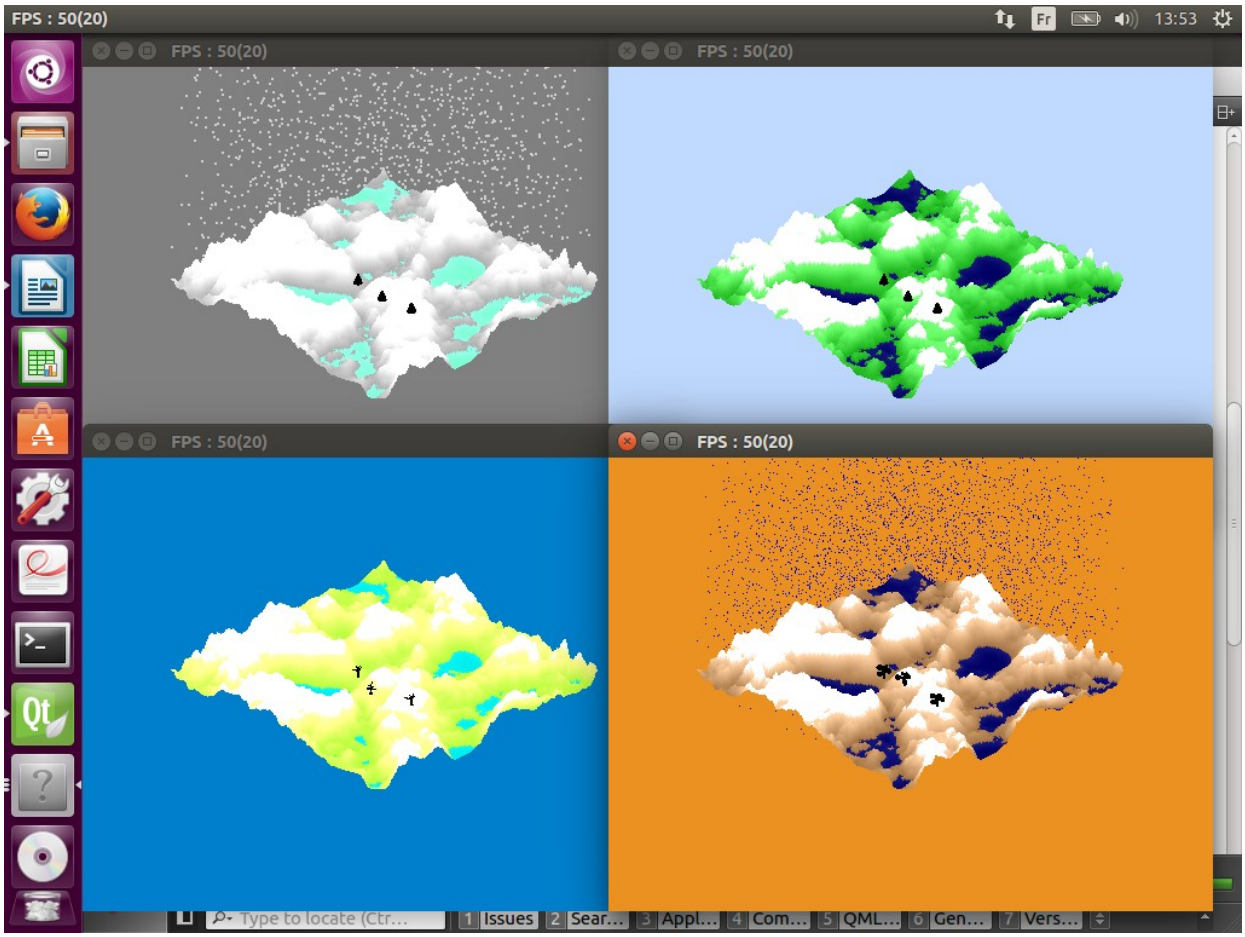
Compte rendu : TP4 – MOTEUR DE JEUX

Vincent de Rousiers



Résumé :

- Ajout d'une classe FileManager afin de gérer la sauvegarde et le chargement des fenêtres et de leurs éléments
- Ajout d'une classe Ply pour représenter les modèles 3D provenant des fichiers .ply
- Amélioration du systéme de particules pour permettre leur sauvegarde
- Amélioration de l'affichage des saisons, représentation de lac et de pic de neige, ajout de trois arbres correspondant à la saison

Question 1 :

```
class FileManager{
public:
    FileManager();

    void saveCustomMap(QString,TriangleWindow**,int);
    TriangleWindow** loadCustomMap(QString);

    int getNbFenetres();
private:
    int nbFenetres;
};
```

Cette classe va enregistrer les nbFenetres fenêtres représentant notre jeu grâce à la fonction void saveCustomMap(QString,TriangleWindow**,int).
Cette fonction va créer un fichier binaire dans lequel elle enregistrera l'ensemble de données de chaque fenetre. Ces données comprennent principalement les différents attributs de la classe `TriangleWindow`, dont notamment les particules en activité (on récupére donc bien la fenêtre dans l'état dans lequel on l'enregistre, rien n'est recréé) ainsi que la liste des Ply présent dans la scéne.

Pour l'instant, toute la sauvegarde est gérée dans cette unique fonction, mais je pense découper cette sauvegarde et munir chaque classe de sa propre fonction de sauvegarde. Ainsi, void saveCustomMap(QString,TriangleWindow**,int) ne ferait plus qu'agréger ces différents fonctions de sauvegardes.

Le chargement se fait donc via TriangleWindow** loadCustomMap(QString) qui renvoie une liste de fenêtres.

En résumé, les éléments sont stockés les uns à la suite des autres en mode binaire et lu de la même façon. Toutes modifications de la sauvegarde impliquent donc obligatoirement une modification du chargement.

Question 2 :

```
class Ply{
public:
    Ply();
    Ply(QString);

    …
    getters/setters
    …
    void drawPoints();
    void drawFaces();
    void drawLines();

    void initMatrix();
    void delMatrix();
    void changeModel(QString);

private:
    QString nom;
    Point** points;
    int nbPoints;
    Point** normales;
    Face** faces;
    int nbFaces;

    float taille;
    float posX;
    float posY;
    float posZ;
    float rotX;
    float rotY;
    float rotZ;
};
```

Cette nouvelle classe permet de charger un modèle 3D depuis un fichier .ply et de l'afficher.
La gestion du chargement est laissé au constructeur Ply(QString) qui va s'occuper de lire le fichier .ply et d'y récupérer les principaux attributs.

L'affichage se fait via les différentes fonctions de draw et le placement, la taille ainsi que la rotation sont gérées par les 7 derniers attributs et la fonction void initMatrix() qui va permettre d'initialiser la matrice (eh oui…). Ces 7 attributs ne sont pas présent dans le .ply et sont donc entièrement gérés par l'utilisateur (le placement des arbres dans mon TP n'est d'ailleurs pas extraordinaire). Ils changent par contre de modèle à chaque nouvelle saison.

Les différents .ply étant stockés dans un tableau propre à une fenêtre, leur sauvegarde et chargement (la lecture du fichier binaire, pas celle du .ply) sont gérés par la classe FileManager. Celle -ci va parcourir le tableau de Ply et les stocker au format binaire.

Test :

Pour tester mon programme, la sauvegarde est automatiquement faite au lancement, pour utiliser le chargement, il faut modifier la variable bool chargement et la mettre à true.

On peut aussi rajouter d'autre modéles .ply dans le main à l'aide de la fonction void addPolygone(Ply*); sur n'importe quelle fenêtre.

Bonus :

- Les différents types d'arbres fournis sont déjà implémentés et d'autres fichiers .ply peuvent être rajoutés sans problèmes.
- Pour gérer d'autres types de modéles 3D, il serait possible de créer une classe Mesh dont hériterait les différentes classes représentant les modéles 3D (.ply, .stl, .obj…).

  La classe Mesh récupérant les ensembles communs aux fichiers (liste de points, faces…) et les sous classes s'occupant principalement des fonctions de sauvegarde/chargement propre aux différents types de fichiers et gérant aussi les attributs qui leur seraient propres (normales, textures...)