COMPTE-RENDU TP4

HMIN317 – Moteur de jeux

Résumé

Création d'un gestionnaire de ressources et affichage d'objet 3D.

David Lonni

Master 2 IMAGINA

1) Fonctionnalités

- Création d'un gestionnaire de ressources FileManager gérant l'enregistrement et le chargement d'un fichier binaire unique contenant toutes les données du jeu à un instant T,
- Création d'une classe Terrain permettant de recueillir toutes les informations importantes liées à l'état des terrains des fenêtres ainsi que des arbres qu'ils contiennent,
- Possibilité de charger et afficher des modèles 3D de type PLY, STL et OBJ dans une classe GameObject,
- Parallélisation des affichages des modèles 3D grâce à la directive #pragma omp
- Ajout d'une lumière dans la scène permettant de voir les objets PLY affichés avec leurs normales,
- Enregistrement de l'état de la caméra dans le fichier binaire, ainsi que l'état des 4 terrains incluant la position, rotation, échelle et chemin de tous les arbres pour toutes les saisons,
- Chargement du fichier binaire restituant l'intégralité de l'état des fenêtres, position caméra, état des terrains, saison courante et position des arbres sur ces terrains,
- Lorsque le fichier de sauvegarde est vide, génération des arbres pour les différentes saisons à des positions aléatoires sur le terrain (calculées en fonction de la position des vertex de celui-ci).

2) Démarche de développement

Classe MyTcpServer

C'est cette classe qui va déclencher le processus d'enregistrement de l'état du jeu avec la méthode MyTcpServer::saveGame().

En effet, cette classe contient un timer qui va, à intervalle régulier, enregistrer l'état des fenêtres.

Elle va tout d'abord faire une copie de l'ancien fichier, si celui-ci existe, pour éviter toute perte de données, puis vider l'ancien fichier.

Ensuite, l'état et la position de la caméra seront enregistrés en premier dans le fichier avant de demander aux fenêtres connectées de sauvegarder leur état.

Classe FileManager

Création d'une classe gestionnaire de ressources utilisant le pattern « singleton » et ne pouvant dont être instancié qu'une unique fois.

Cette classe permet de sauvegarder au format binaire les différents terrains passé en paramètre via la méthode FileManager::saveCustomMap(Terrain* T). Le gestionnaire de ressource va sauvegarder dans le fichier, tous les vertex du terrain, ainsi que sa couleur et sa saison courante. Il va également inscrire la position et chemin des objets 3D concernant les différents arbres pour toutes les saisons du terrain.

Ensuite, une méthode FileManager::loadCustomMap(QString _localPath) permettra, à chaque ouverture du programme, de charger le fichier binaire s'il existe, et de générer les terrains en fonction de celui-ci.

Les différentes fenêtres peuvent alors, lors de leur création, récupérer au gestionnaire de ressources un terrain afin de pouvoir l'afficher. La caméra peut également lors de sa création, récupérer les informations enregistrées dans le fichier afin d'être dans le même état, la même rotation et échelle que lors de la sauvegarde.

Classe GameWindow

La classe GameWindow va, lors de son initialisation, demander au gestionnaire de ressources s'il existe un terrain à charger. Si c'est le cas, elle va récupérer ces informations et initialiser les arbres sur le terrain en fonction des données. Sinon, elle va charger une heightmap et initialiser des arbres pour chaque saison à des positions aléatoires correspondants aux positions des vertex du terrain.

Le nombre d'arbres générés pour chaque saison est donné par l'attribut GameWindow::NB_ARBRES.

Lorsque le serveur envoi l'information de sauvegarde, la méthode GameWindow::save() est appelée et va ainsi créer le terrain correspondant pour l'envoyer à la classe FileManager afin de l'enregistrer dans le fichier binaire.

Classe GameObject

Pour le chargement des modèles 3D il m'a fallu tout d'abord regénérer les arbres fournit au format PLY sur Blender car leur faces n'étaient pas toutes triangulées et leur origine n'était pas au centre de l'objet selon x et y. Cela m'a permis de traiter plus simplement ces modèles par la suite.

Cette classe permet le chargement et l'affichage des objets 3D au format PLY, STL et OBJ. Lorsque les arbres (qui sont des GameObjects) sont initialisés, en fonction des données du fichier ou aléatoirement, un chemin de fichier est passé en paramètre du constructeur de GameObject.

La méthode GameObject::open() va alors déterminer de quel type d'objet il s'agit en examinant l'entête du fichier. Ensuite, elle pourra appeler la méthode d'ouverture approprié en fonction du type de l'objet 3D.

```
void openPLY();
void openSTL();
void openOBJ();
```

La méthode GameObject::display() fonctionne sur le même principe. Elle se chargera de placer selon la bonne position, rotation et échelle les arbres, ainsi que d'appeler la méthode d'affichage correspondante au type de l'objet.

Certain type d'objet liste tous les vertex de l'objet puis relisent ces vertex sous forme de faces indexées, que l'on pourra stocker dans un tableau d'indices, d'autres possèdent des normales par sommet ou par face, et certain rien de tout cela.

Une méthode d'affichage spécifique a donc été développée en fonction de l'objet que l'on veut afficher qui est caractérisé par un enum.

```
enum Format { PLY, STL, OBJ };

void displayPLY();
void displaySTL();
void displayOBJ();
```

Ces méthodes étant définies comme `private` la classe GameWindow affiche les arbres en appelant uniquement la méthode display() qui se charge de rediriger l'appel vers la méthode appropriée :

```
for(unsigned int i = 0 ; i < NB_ARBRES ; i++){
        tree[saison][i]->display();
}
```

3) Structure de données

Le fichier binaire est structuré comme ceci :

Etat caméra – rotation caméra – échelle caméra

Pour chaque terrain

| *Saison terrain – couleur terrain – nombre vertex longueur – nombre vertex largeur*

| *Tous les vertex (x,y,z) du terrain*

| *Nombre de saison du terrain – Nombre d'arbres par saison*

| **Pour chaque arbre (de chaque saison)**

| | *Chemin de l'objet 3D – position de l'arbre – rotation de l'arbre – échelle de l'arbre*

4) Parties bonus

Pour les parties bonus j'ai réalisé l'affichage de plusieurs arbres par terrain et par saison ainsi que leur enregistrement dans le fichier binaire.

J'ai ajouté l'arbre island_tree.ply, téléchargé sur le site blendswap afin de différencier les arbres de la saison hiver et printemps.

J'ai également permis le chargement de fichier 3D sous les formats STL et OBJ en plus du format PLY.