LAURINE CAPDEVILLE DAPHNÉ CHAMOT-ROOKE LÉA ROSTOKER

IMACITY

PROJET D'OPENGL ET C++ D'IMAC 2

I. Scénario du jeu et inspiration graphique :

Le joueur commence la partie sur une île sur laquelle trône un petit village et son château, le tout bardé d'une forêt. Notre jeu consiste en une sorte de chasse au trésor dont le but est d'entrer dans le château. Pour cela, le joueur doit trouver toute les clés dans le petit univers que nous avons créé. La première clé se cache sur l'île et peut ouvrir une des maisons. Les autres clés, elles, se cachent dans les différentes habitations. Une fois que le joueur aura en sa possession les quatre clés, il pourra entrer dans le château et il aura gagné la partie !

Pour l'aspect graphique du jeu, nous nous sommes inspirées d'une maquette en papier réalisée par Laurine (première photo ci-dessous et au milieu, peinture d'inspiration), elle même inspirée le style de la maquette en papier ci dessous (troisième photo). Nous avons donc voulu recréer cet aspect "papier", lisse dans notre projet.

III. Détails techniques :

1) Méthodes de travail :

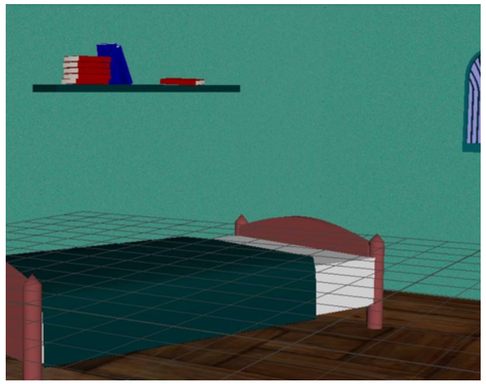
Pour nous répartir les tâches, nous avons fait des points réguliers au moins une fois par semaine, pour partager notre avancée dans le projet et répartir les nouvelles tâches en fonction des accomplissements de chacun. L'utilisation de git simplifie beaucoup le processus, même s' il y a eu quelques difficultés pour Laurine et Léa à prendre en mains cet outil au début.

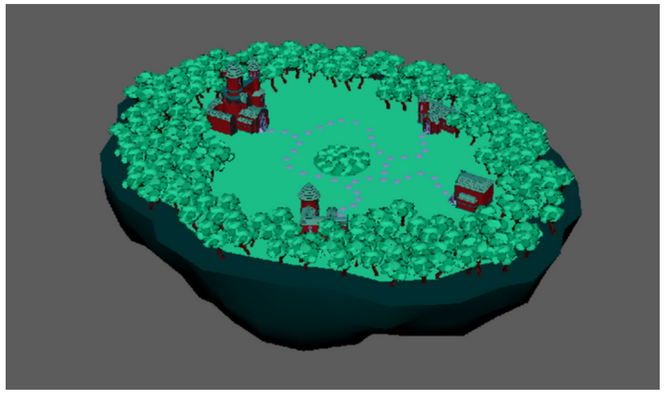
2) Bibliothèques utilisées:

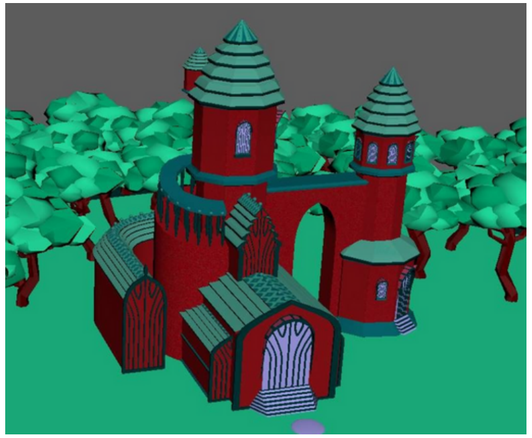
Pour notre projet, nous avons décidé d'employer la bibliothèque assimp en suivant le tutoriel de learnOpenGL. Pour la gestion de la fenêtre et des événements, nous avons choisi la bibliothèque glfw3, et avont utilisé glew pour la gestion d'OpenGL.Pour le parsing des fichiers de description de scène, nous avons trouvé une librairie très pratique appelée confini, qui a grandement simplifié cet aspect du projet.

3) Modèles 3D :

Nous avons créé nos éléments de décors sur le logiciel de 3D Maya. Pour être en accord avec nos inspirations nous avons réalisé des modèles 3D géométriques pour les maisons et nous avons fait les arbres et les buissons en "low poly" pour renforcer l'impression que les éléments du décors sont des maquettes en papier.







IV. Description du projet :

1) Structure du répertoire :

\-- assets/modes/ ->tous les models à charger et leurs textures
\-- doc/ -> sujet et rapport
\-- include/ -> fichiers.hpp
\-- lib/ -> glm et SOIL2
\-- src/ ->fichiers.cpp et shaders
\-- ini-files/ -> tous les fichiers .conf de description des scènes

Notre projet est, comme il était demandé dans le sujet, réparti en classes. Les classes Models, Mesh, et Shader permettent le rendu des modèles 3D avec la bibliothèque assimp ainsi que la gestion des shaders. Les classes Event, Gameplay, et FreeflyCamera permettent l'interaction et le mouvement dans les scènes. Enfin les classes LoadIniFile et Scene permettent la lecture de nos fichiers de description des scènes et la mise en place de celles-ci.

2) Instructions du clavier :

Pour changer l'angle de vue, il suffit de bouger la souris.
Les commandes clavier sont les suivantes :

- **Echap** : quitter
- **Z** : avancer
- **S** : reculer
- **Q** : aller à gauche
- **D** : aller à droite
- **C** : entrer dans la pièce ou en sortir.
- **H** : ramasser la clé

IV. Description du projet :

3) Fonctionnalités

Le joueur peut se déplacer dans toute la scène. S'il en possède la clé, il peut entrer dans la maison correspondante en se plaçant devant une porte. Il peut alors se déplacer dans une nouvelle scène (et ne peut pas aller au-delà des murs). Il peut directement sortir, sans avoir besoin de se mettre en face de la porte, en cliquant sur C, ce qui le ramène dans le village. Pour ramasser la clé dans le village, le joueur doit s'en approcher suffisamment. Dans les maisons, ce n'est pas nécessaire : il peut la ramasser directement en cliquant sur H, et elles disparaîtront de la scène.

4) Fichier de description des modèles :

Dans chaque fichier, le nombre de modèles et le nombre de shaders à utiliser est d'abord précisé. Une description de chaque modèle est ensuite faite, avec son fichier .obj et ses valeurs de transformations.

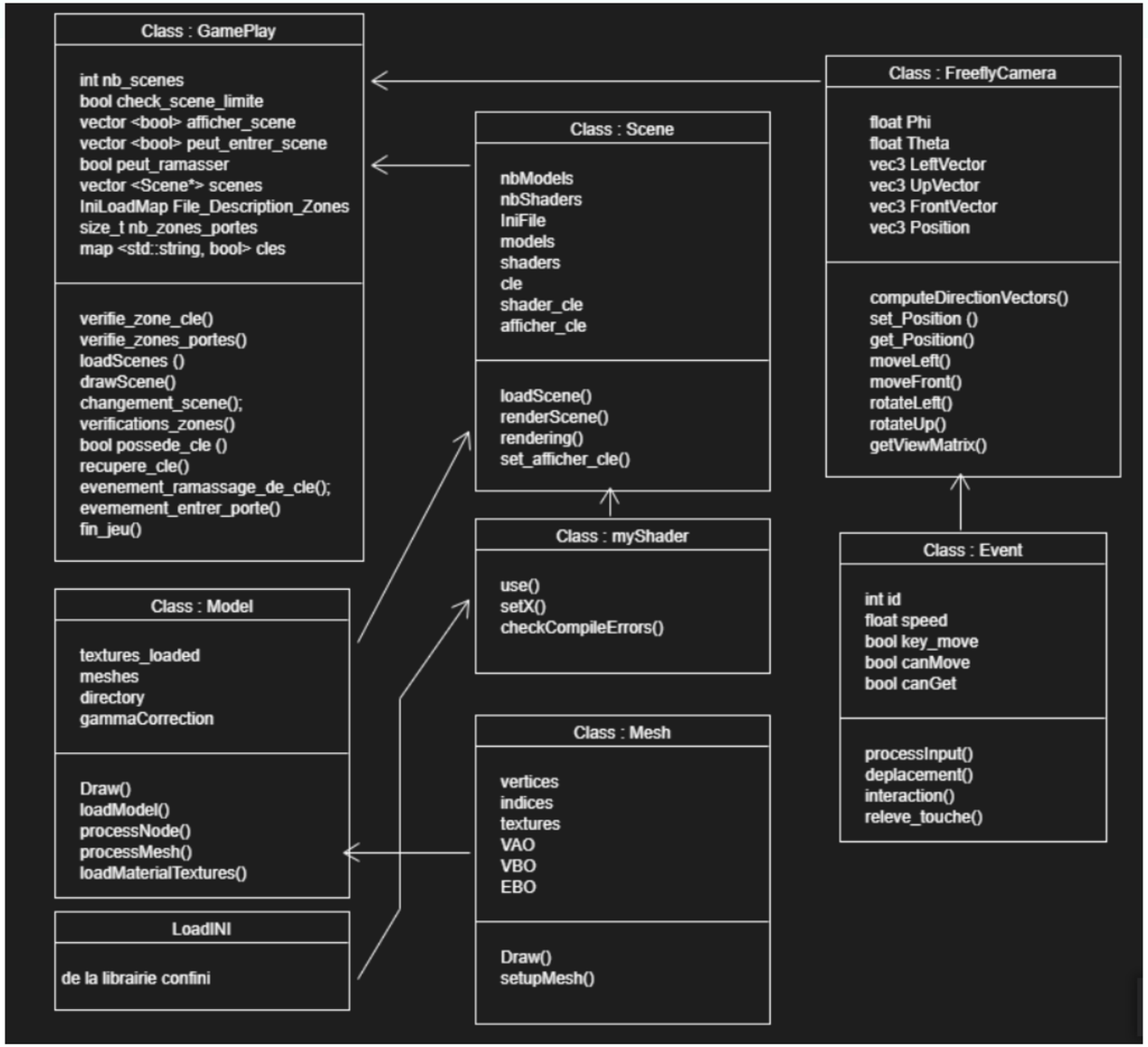
4) Lumières :

Nous avons créer deux shaders spéciaux pour les lumières (vertex et shader) en faisant une combinaison de lumières diffuse, ambiant et specular.
Nous avons, en outre, choisis de faire varier la lumière au cours du temps (en modifiant les couleurs vertes)

V. C++:

Classes	✓	Partout ex : Event.cpp	
Fonctions templates	✗		Nous n'avons pas trouvé de contexte où l'utilisation d'une fonction template aurait été judicieuse. Nous sommes certainement passé à côté, mais toutes nos fonctions devaient soit prendre un seul type, soit avoir un comportement différent pour chaque type, nous avons donc laissé tomber cette requête.
Héritage et polymorphisme	✗		Nous pensons que l'héritage et le polymorphisme n'auraient pas été judicieux pour notre projet.
STL	✓	très utilisée ex : Gameplay.hpp	
Exceptions	✓	myShader.cpp main.cpp, Scene.cpp	
Message d'erreur	✓	myShader.cpp main.cpp Gameplay.cpp...	
Assertions	✓	Gameplay.cpp par exemple	
Espace de nommage	✓	myShader.cpp FreeflyCamera.cpp	Ces deux fichiers sont ceux qui pouvaient potentiellement être en conflit avec d'autres fichiers de librairies au moment du développement (notamment avec glimac que nous avons fini par supprimer mais qui avait des classes similaires)
Fonctions lambdas	✗		Nous n'avons pas de contexte où nous en servir non plus. Dans le cas des fonctions lambda, elles auraient certainement pu nous être utiles, mais le manque de temps et de compréhension profonde de cet outil a malheureusement fait passer les fonctions lambda à la trappe.
Constexpr	✗		
Fonctions variadics	✗		

VI. Architecture logicielle :



VII. Difficultés rencontrées :

1) Travail en groupe à distance :

Nous avons parfois eu quelques difficultés dues au travail à distance au regard des conditions sanitaires actuelles. Cependant, nous nous sommes entre-aidés pour l'utilisation de github notamment, ce qui nous a grandement aidés au cours du projet.

2) Gestion des différentes librairies :

L'installation et la gestion de librairie a été la partie la plus difficile. Une fois le chargement des modèles en place, il n'y a pas eu de difficultés majeures, mais avant d'atteindre cet objectif, les choses n'ont pas été simples. D'abord, beaucoup de librairies avaient des modes de compilation et d'installation différents, ce qui est difficile à gérer avec notre niveau de compréhension de Cmake. Il y avait également des conflits entre glut et glew que nous avons mis du temps à comprendre. Mais une fois passé ce cape, les choses sont allées beaucoup plus vite.

3) Compréhension du sujet :

L'autre difficulté majeure que ce projet apportait était les nombreuses consignes, dont nous ne comprenions pas la moitié au début, ce qui faisait assez peur. Il était difficile au début d'avoir une vision globale du projet quand nous ne savions pas ce qu'était le parsing ou le chargement de model. Mais ça a l'avantage de nous forcer à nous adapter et à apprendre vite.

VIII. Améliorations possibles :

1) Ajout de musique :

Pour rendre l'expérience de jeu plus agréable nous pourrions ajouter une musique de fond (nous pensions à quelque chose de calme) et des bruitages lors des interactions avec le jeu (lorsque l'on ramasse une clé ou que l'on ouvre une porte).

2) Gestion des collisions :

Pour éviter que le joueur ne puisse passer au travers des modèles, il faudrait que l'on gère les collisions.

3) Interface graphique :

Pour rendre le jeu plus pratique d'utilisation nous avions pensé à ajouter plusieurs choses : une fenêtre de texte par dessus la scène pour expliquer le but du jeu quelque chose permettant d'indiquer que l'on peut ramasser une clé ou ouvrir une porte quand les conditions sont vérifiées (par exemple l'objet changerait de couleur ou une fênetre de texte s'ouvrirait pour indiquer qu'une action est possible) une fonctionnalités permettant de voir ce que l'on possède pour savoir combien de clé on a ramassé et/ou combien il en reste

4) Gestion du chargement :

Enfin, il aurait probablement été possible de mieux gérer nos chargement de scènes, de models et de shader avec un peu plus de connaissances. Le chargement des scènes supplémentaire aurait par exemple pu se faire au moment des changements de scènes au lieu que tout se fasse au début du programme. L'utilisation des shaders aurait pu être optimisée, car nous avons créé un programme pouvant gérer un shader par modèle alors que tous les modèles utilisent le même.