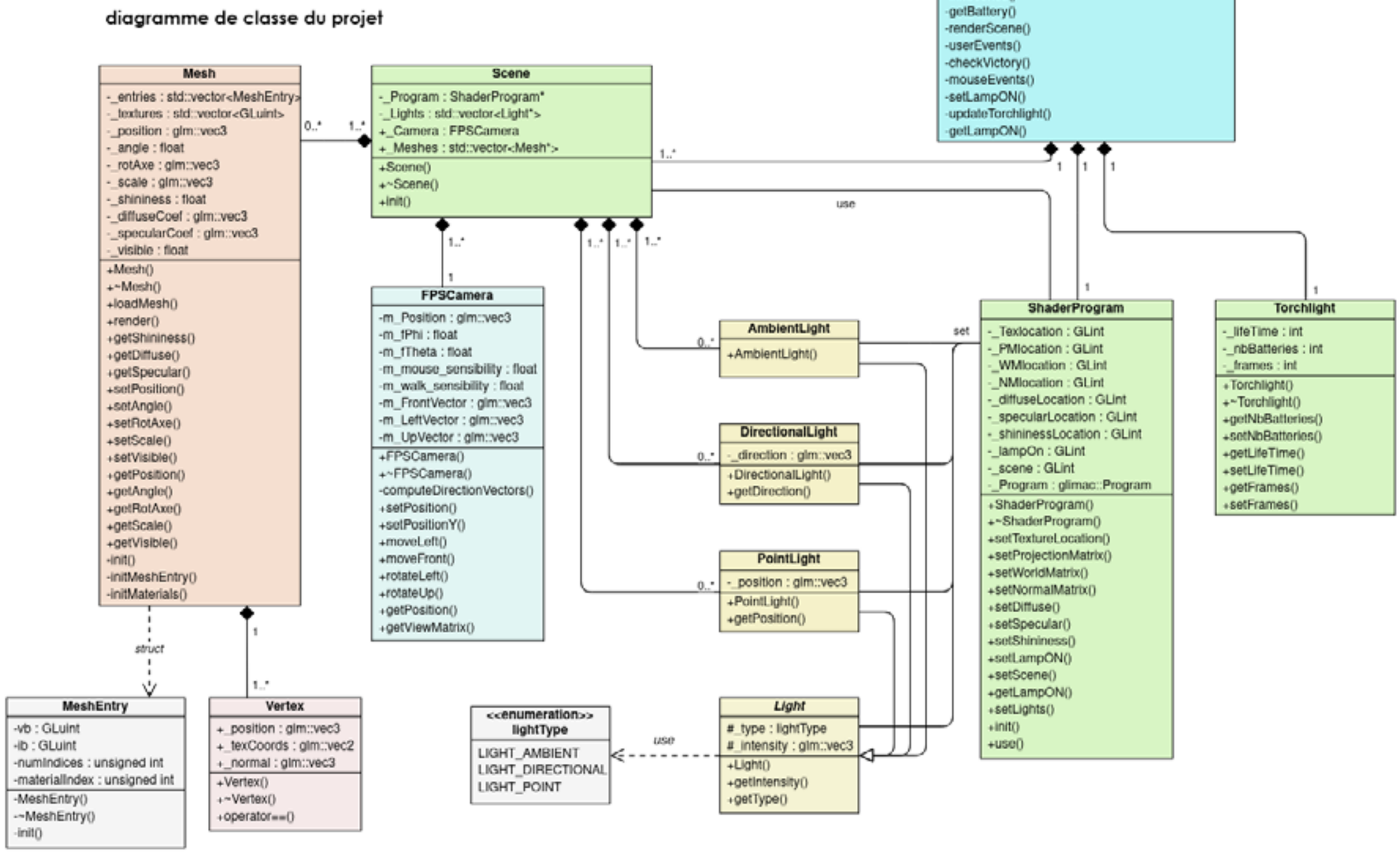
IMAC Nightmare

Fabian ADAM Benjamin BRIERE Théo GAUTIER

SOMMAIRE

ARCHI LOG

diagramme de classe du projet



```
namespace game
{

    class GameEngine
    {
    public:
        GameEngine(const char* appPath)
        ~GameEngine();

        bool init(const char* vsFilenam
        void run();

    private:
        bool initSdlOpenGl();
        bool initScenes(const char* fil

        // test récupération des piles
        void getBattery();
```

```
        void renderScene();
        bool userEvents();
        void checkVictory();
        void mouseEvents(SDL_Event e);
        void setLampON(int v);
        void updateTorchlight();

        int getLampON();

        glimac::SDLWindowManager* _Wind
        ShaderProgram* _Program;
        std::vector<Scene*> _Scenes;
        Torchlight* _Torchlight;

        int _SceneId;
        int _lampON;
        int _NbStatue;
```

MOTEUR DE JEU

Notre Game Engine représente le coeur de notre projet. Il est la source de presque toutes les composantes du jeu (entre autre car nous voulions éviter d'utiliser des variables globales). L'objectif était de charger des modèles depuis des fichiers externes. Ainsi, tant que le Game Engine n'était pas fonctionnel nous ne pouvions pas travailler sur les mécaniques de Gameplay. Cette méthode ne permettait pas une réelle productivité mais le Game Engine nous a monopolysé de précieuses d'heures avant de pouvoir le rendre véritablement fonctionnel.

Schématiquement, le moteur de jeu fonctionne de la manière suivante :

1 - Initialisation : tests de sécurités pour voir si les librairies sont correctement chargées.

2 - Exécution : boucle de rendu, traitement des événements côté utilisateur et update de la scène 3D + test de la condition d'arrêt du jeu (Win/Lose).

Le Game Engine est pensé pour ajouter un HUD mais nous n'avons pas eu le temps de le développer. Il faudrait encore ajouter la classe HUD en 2D, les shaders associés et les méthodes de calcul pour garder un affichage dynamiquement mis à jour et immobile.

Ensuite, nous nous sommes rendu compte qu'il fallait ajouter de nouvelles classes différentes (ex : lampe torche) pour gérer certains aspects du jeu et alléger la charge de travail du Game Engine.

Dans la scène, on génère une lumière directionnelle et on simule un éclairage de lampe torche. Nous avons créé une nouvelle variable uniforme qui gère le mode "Lampe" et son allumage. Si la lampe est alllumée, on utilise un modèle Blinn-Phong associé à des paramètres qui gèrent la taille du cône d'éclairage ainsi que la distance d'affichage (la lumière est plus puissante au centre de la lampe que sur les extérieurs).

TECHNIQUES

Technique	Oui	Non	Commentaires
Classes	x		Ex : Classe Lampe torche
Classes et fonctions templates		x	Nous n'en avons pas eu besoin au vu de notre projet.
Héritage	x		Surtout utilisé pour les lumières
Polymorphisme	x		Polymorphisme ad hoc. Exemple : torchlight.cpp (fonctions setFrames, setNbBatteries, setLifeTime).
Outils de la STL	x		Utilisation surtout des vecteurs (ex : GameEngine getbattery() : for auto Mesh...
Exceptions	x		Ex : GameEngine init()
Messages d'erreur	x		Beaucoup utilisé entre autre lors des chargements des textures.
Asserts	x		Utilisé très ponctuellement pour éviter les divisions par 0 ou autres erreurs critiques (ex : Cone.cpp)
Espaces de nommage	x		Nouvel espace de nommage : Game
Fonctions lambdas		x	Nous n'en avons pas eu besoin au vu de notre projet.

MUSIQUE

Dans notre démarche créative, l'ambiance sonore est primordiale pour que l'utilisateur se plaise dans le jeu.

Pour l'ambiance sonore, nous avons choisi de créer notre propre musique et de l'implémenter dans le jeu sous forme de boucle sonore. Pour cela, la SDL propose un ensemble de fonctions liées à SDL_Mixer. Cette librairie est très pratique car elle permet de gérer les volumes, les nombre de répétitions que l'on souhaite pour notre fichier sonore.

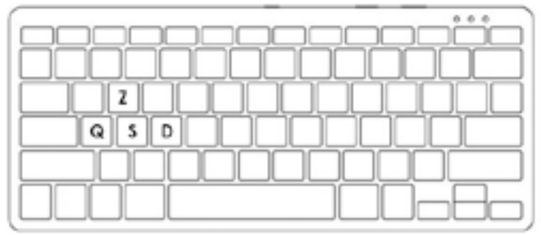
De même, nous avons choisi d'ajouter des bruitages lorsque on allume/éteint la lampe torche. Ici encore, nous nous sommes servis de SDL_Mixer qui permet, de charger des bruitages via les canaux audios préalablement ouverts.

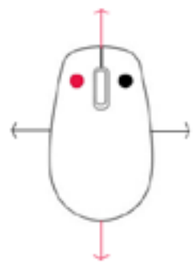
On peut ainsi gérer l'ensemble des pistes audio individuellement. Les musiques d'un côté et les bruitages séparés en canaux audio dont on peut ajuster le volume séparément.

INTERACTIONS

Mais comment jouer ?

Nous avons gardé un système de jeu aussi classique que possible pour plusieurs raisons. Tout d'abord, pour ne pas perdre l'utilisateur, mais aussi pour nous permettre de nous concentrer au maximum sur le rendu, plus que sur des fonctionnalités qui nous auraient pris du temps de développement inutile. Par exemple, faire sauter le personnage n'a aucun intérêt dans notre jeu. Ainsi, vous pouvez vous déplacer avec ZQSD et tourner la caméra avec un clic gauche et déplacement de la souris. Pour ramasser les objets (batteries et statues), il vous suffit de passer dessus ou d'être à proximité de l'objet. Enfin, vous pouvez, si vous le désirez, éteindre votre lampe torche et la rallumer avec clic droit (petite astuce pour mettre le jeu en pause !).





Z : Avancer
Q : Gauche
S : Reculer
D : Droite

Activer - Désactiver la lampe
Rotation de la caméra (maintenir)
↔ Tourner la caméra (droite-gauche)
↔ Tourner la caméra (haut-bas)

Resident Evil PS1

Alone in the dark PS1

IMAC Nightmare - Scène

Batterie

Statue #1

Statue #2

MODELISATION

Pour modéliser notre environnement, nous avions tout d'abord entrepris de créer le manoir "from scratch" sur Blender. Cela aurait été possible mais nous souhaitions obtenir un résultat proche du réalisme et plonger l'utilisateur dans un monde angoissant. Nous avons donc pensé à récupérer un capture 3D de manoir (libre de droit). Lors de l'implémentation de cette capture dans le jeu, nous nous sommes rendu compte que cela laissait apparaître des artefacts avec un rendu un peu "sali" sur l'image. Cela nous a fait penser aux jeux d'horreur de notre enfance, nos premiers jeux en 3D (fin des années 90', début 2000') comme Resident Evil, Alone in the dark et bien d'autres illustrés sur la page de gauche. Nous avons donc décidé de garder cette base 3D pour notre jeu.

En plus de cela, nous avons ajouté quelques props (petits objets) qui servent à l'utilisateur. Par exemple, nous avons ajouté des plots qui servent à lui indiquer que l'accès aux escaliers est interdit. De même, nous avons ajouté des batteries qui seront collectées par le joueur pour alimenter sa lampe torche. Elles sont disposées un peu partout dans le manoir mais n'apparaissent qu'une par une. Elles ont une durée d'utilisation de 25 secondes.

Enfin, nous avons ajouté des statues qui sont les objets à collecter pour s'échapper du manoir. Pour les trouver, il faut explorer l'ensemble des pièces de la scène.

DIFFICULTÉS

Nous avons perdu un temps extrêmement précieux pour prendre la main sur le moteur de jeu et l'importation des objets 3D. Nous n'avons donc eu que très peu de temps pour développer les autres aspects du jeu.

De plus, dans de nombreux cas, la documentation de méthodes nous paraissait obscure. Nous avions donc besoin de temps pour assimiler des connaissances cruciales pour le développement du projet. Ceci est couplé au fait que l'utilisation de glimac permet d'un côté de gagner du temps mais peut être source de destabilisation pour rechercher des tutos liés à ses méthodes. Nous avons souvent perdu du temps pour savoir si telle ou telle composante était native ou faisait partie de glimac.

Ensuite, plusieurs d'entre nous ne connaissaient que peu le système GIT. Nous avons longtemps transféré nos fichiers par des méthodes peu recommendables par tout développeur (WeTransfer, mail...) pour éviter, entre autre, d'écraser les données mises en ligne sur GIT. Cela explique que notre utilisation de GIT est arrivée dans les derniers jours de développement.

Comme de nombreux groupes au sein de la promo, nous vivons une période extrêmement rude. Le S3 représente le semestre charnière de notre formation. Il est donc normal que notre charge de travail soit décuplée. Pour autant, celui-ci arrive dans une période où bon nombre d'entre nous ont décroché de certains cours, entre autre à cause de l'incapacité que nous avons à nous voir pour nous entraider en présentiel. Bien que ce projet soit un développement d'application, donc du code, il était nécessaire pour nous de nous rencontrer, d'échanger pour éviter certaines incompréhension (qui nuisent au développement) et pour garder tout le monde à flot. En ce sens, le projet de développement du jeu a réellement été difficile pour notre groupe.

Enfin, travailler sur des VM nous a rendu la tâche compliquée puisque le jeu tournait plus ou moins bien selon les capacités en RAM de nos machines virtuelles. Il aurait fallu accéder aux machines de l'IMAC pour avoir une référence en termes d'exécution et d'affichage de notre programme.

AMÉLIORATIONS

Nous avons identifié un certain nombre de points que nous aurions aimé améliorer si le temps et nos compétences nous l'avaient permis. Tout d'abord, il serait intéressant de voir évoluer notre personnage au travers de différents niveaux. Actuellement, notre workflow nous permet d'ajouter facilement de nouvelles pièces en fonctions d'évènements. Il suffit de créer un nouveau niveau et de le lier au précédent en cas de victoire.

De plus, nous pensons qu'il serait préférable d'améliorer la fluidité des mouvements de la caméra (avoir une démarche vacillante, améliorer la rotation de la caméra).

Afin de rendre meilleure l'expérience utilisateur, il faudrait aujouter un écran de chargement au début du jeu lorsque les textures se chargent et un menu.

Comme énoncé précédemment dans le rapport, nous aurions souhaité ajouter un HUD permettant d'afficher les piles restantes, la durée de vie de la pile en cours d'utilisation, le nombre de statuettes manquantes ou encore le temps de jeu écoulé.

Lors du chargement des textures, les objets sont chargés autant de fois qu'ils sont instanciés dans le jeu. Cette méthode est fonctionnelle mais augmente le temps de chargement.

Concernant la lumière, une lumière directionnelle éclaire la scène selon un axe. La lampe torche fait donc un peu l'illusion car elle sert davantage à restreindre l'affichage qu'à véritablement éclairer la scène (excepté sur l'axe de la fameuse lumière directionnelle). Nous aurions souhaité ajouter de l'aléatoire sur l'éclairage en fonction de la durée de vie de la batterie. Nous avions développé cette méthode et l'avions rendu fonctionnelle, mais nous avions remarqué que cela nuisait grandement aux performances du jeu. Ainsi, nous avons décidé de conserver les performances en dépit de cette feature.

Un élément important manque au jeu : les collisions. Au vu du temps de développement qui nous restait avant la date de rendu, il nous était impossible de développer l'entièreté des hitboxes.

Enfin, il serait préférable de créer nous-même un modèle 3D de la scène qui soit optimisé pour le jeu au lieu d'utiliser un MégaScan, bien que celui-ci nous plaise et rende une ambiance qui correspond à nos attentes.

RESSOURCES

Batterie (Plutonium.software) : http://bit.ly/IMACNightmare-battery

Plots (늑대머리) : http://bit.ly/IMACNightmare-Plot

Manoir (Thomas Flynn) : http://bit.ly/IMACNightmare-Manoir

Statue 1 (grafi) : http://bit.ly/IMACNightmare-Statue1

Statue 2 (grafi) : http://bit.ly/IMACNightmare-Statue2

Musique créée par le groupe.

Les ressources 3D sont libres de droit et sous licence Creative Commons.