SYNTHÈSE D'IMAGE II
PROGRAMMATION C++
ARCHITECTURE LOGICIEL

PROJET IMAC 2
ImacDream - ImacNightmare

AUBERT ALOÏS - HUVIER CLOTHILDE - ROZAIRE PAUL

JANVIER 2021


ECOLOGEU

TABLE DES MATIÈRES

SYNOPSIS 4

MÉTHODE DE TRAVAIL 6

ORGANISATION 8

FONCTIONNALITÉS C++ 9

FONCTIONNALITÉS BONUS 10

AFFICHAGE DU FOND 14

DIFFICULTÉS 15

AMÉLIORATIONS 17

NOTICE 19

ANNEXE : DIAGRAMME UML

SYNOPSIS DU JEU

Notre jeu est un simple **jeu contemplatif de passage de monde en monde**. Au démarrage, le joueur apparaît dans un monde aux aspects féériques, légèrement éclairé qui donne une ambiance magique. Il doit trouver des **indices cachés** à certains endroits afin de **débloquer la plateforme de téléportation**, le portail, qui l'emmène vers le deuxième monde du jeu.
Le deuxième monde est une petite île naturelle sur laquelle le joueur doit également récupérer trois indices afin de débloquer la plateforme de téléportation qui l'emmène vers le troisième monde.
Le troisième monde est un espace rocheux, naturel encore une fois, dans lequel se cache un espace non-visible à l'arrivée. Comme lors des mondes précédents, le joueur doit trouver les indices qui lui permettront de terminer le jeu.

SPOILER ALERT

Les **indices** à trouver dans ces mondes sont sous forme de **poubelles de recyclage**, de **déchets** à récupérer et **d'outils pour ramasser ces déchets**.
Nous avons voulu faire une **énigme simple**, basique, qui permet au joueur de **se balader de monde en monde**, mais avec un message précis.
Les mondes sont naturels, non modifiés visiblement par l'action de l'homme. Ce sont de beaux paysages dans lesquels se cachent des détritus. Nous avons voulu transmettre un **message écologique**, écoresponsable à travers un univers simple et léger.
Les mondes représentent ce à quoi ressemblerait notre belle planète sans l'action néfaste de Hommes sur son écosystème. L'utilisateur est donc "sensibilisé" en quelque sorte à la protection de l'environnement.
C'est la raison pour laquelle il aurait été intéressant de pouvoir importer nos propres mondes (Cf. partie Améliorations possibles).

MÉTHODE DE TRAVAIL

Nous avons dans un premier temps mis en place un repo GIT sur lequel nous avons tous les trois travaillé. Nous avons passé une soirée tous ensemble pour nous familiariser avec GIT et plus particulièrement avec **Git Kraken** (comment créer une branche, faire un commit, un merge, comment push/pull des modifications...).

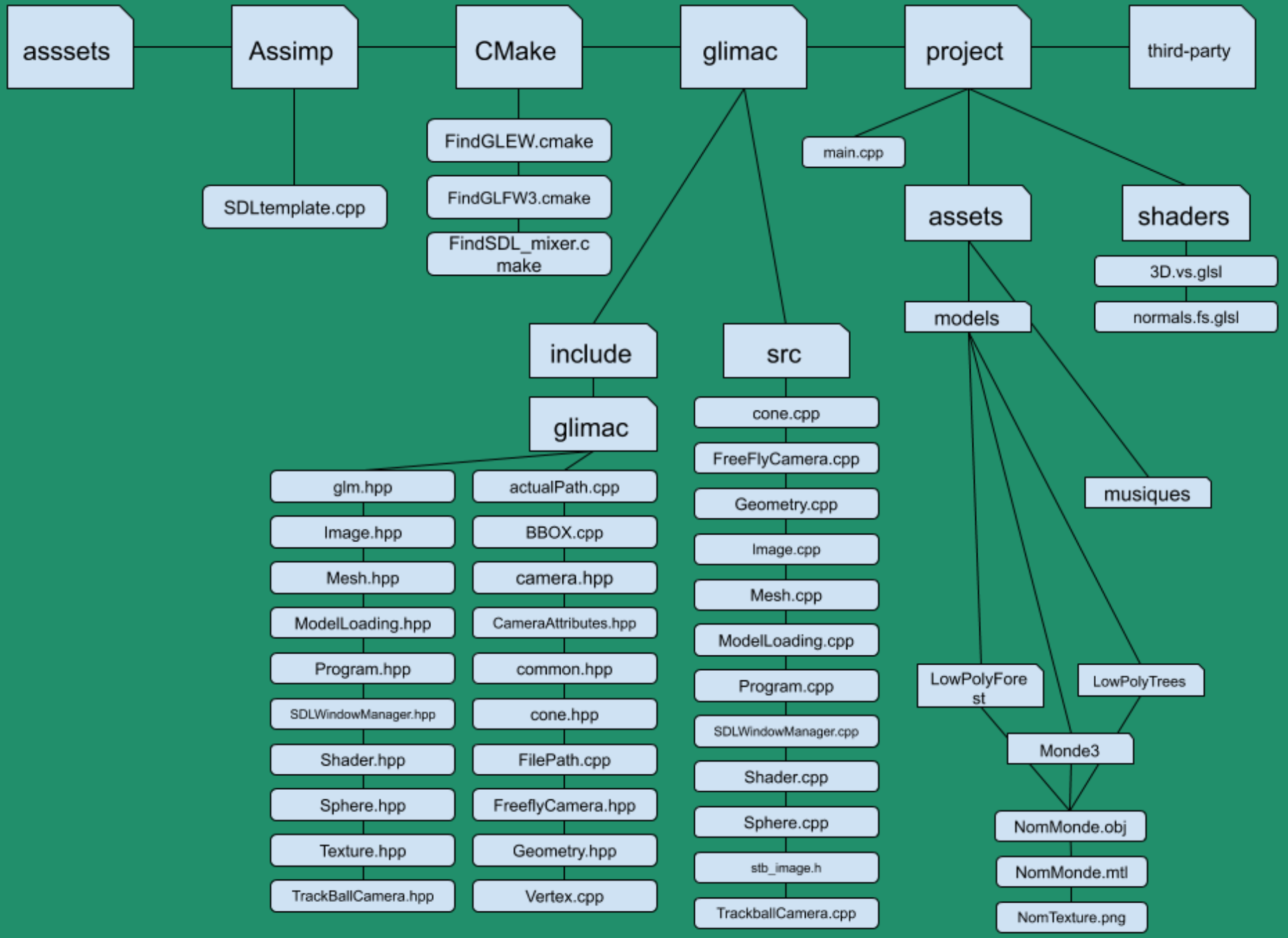
Une fois que nous étions tous à l'aise avec le fonctionnement de GIT, nous avons réfléchi à un environnement que nous souhaitions représenter. Malheureusement, à cause du manque de temps à accorder au projet, nous avons fait le choix de ne pas modéliser les mondes. Notre but était donc de trouver des **mondes déjà modélisés** et de les exporter en **.obj** pour gagner du temps et nous concentrer sur la partie programmation du projet.

À la suite de cela, nous avons commencé à réfléchir à un **diagramme de classe** pour nous aider à visualiser la portée du projet et l'**organisation des fichiers/dossiers**.
Nous avons repris l'organisation des dossiers dans lesquels nous avons travaillé pour les TPs. Cependant, au démarrage du projet, nous n'avions pas encore une idée très précise des classes que nous allions créer. Notre diagramme de classe a donc évolué au fur et à mesure que nous avancions dans la réalisation du programme. Nous nous sommes rapidement rendus compte des nombreux avantages que nous apportait la réalisation d'un diagramme de classe : le développement était plus rapide, plus simple, moins coûteux en termes de réflexion puisque tout avait été défini sur papier au préalable.

Une fois ces premières étapes réalisées, nous avons pu commencer le développement du jeu. **Paul s'est occupé de faire la base du code avec l'installation d'assimp, et le chargement du premier monde**. L'idée était d'**avoir une base propre, claire, commentée, sur laquelle nous allions tous pouvoir travailler proprement.**

Une fois cette base réalisée, il l'a push sur le Git et nous avons fait une réunion pour nous répartir les tâches. **Une fois la tâche réalisée, la personne pushait sur le git et tout le monde pouvait aller récupérer et éventuellement déboguer** (notamment au niveau des liens vers les modèles, les musiques, les textures). Plus tard dans le développement, nous avons trouvé une solution pour résoudre ce problème, que nous vous expliquerons plus précisément lors de la soutenance.

ORGANISATION DES DOSSIERS



FONCTIONNALITÉS C++

Noms	Présent dans le projet (oui/non)	Si oui, dans quel(s) fichier(s)	Commentaires/ indications
Classes	oui	camera, CameraAttributes, Cone, enigme, FilePath, FreeflyCamera, font, Geometry, Image, Mesh, Model, modelLoading, Shader, TrackballCamera, SDLWindowManager, Sphere	
Classes+fonctions template	oui	enigme.hpp FilePath.hpp (template<>) Model, Sphere, Cone Font.hpp (map) Geometry, Mesh.hpp modelLoading tiny_obj_loader (map et vector)	Vector
Héritage	oui	FilePath.hpp (provient de glumac)	Outre le "friend" qui provient de glimac, on aurait pu utiliser l'héritage pour les caméras avec une classe amie.
Polymorphisme	non		Pas naturel pour nous, de ce fait, nous ne voulions pas forcer le polymorphisme.

Noms	Présent dans le projet (oui/non)	Si oui, dans quel(s) fichier(s)	Commentaires/ indications
Outils STL	oui	Cone, enigme.hpp, Geometry, Mesh.hpp, Model, modelLoading.hpp, Image, tiny_objet_loader, Sphere	vector, unordered_map
Exceptions erreurs systèmes	oui	ActualPath.cpp, main.cpp, Program.cpp, shader.cpp	throw runtime error(“”), “ifdef_WIN32”, “IFDEF_APPLE_”,
Exceptions erreurs utilisateurs	oui	main.cpp, Model.cpp, Image.cpp, font.cpp, SDLWindowManager.cpp	std::cerr, “ERROR::FREETYPE : “
Asserts (détecter et prévenir erreurs de programmation	oui	tiny_objet_loader.cpp	
Espaces de nommages	oui	BBox.hpp, Cone, FilePath.hpp, GeometryImage, main.cpp, Mesh.cpp, Model.cpp, Program, SDLWindowManager, Shader, Sphere.cpp	glimac, std, tinyobj
Fonctions lambdas	non		Manque de compréhension de la notion.

FONCTIONNALITÉS BONUS

UTILISATION D'ASSIMP

Très utile pour **charger les scènes 3D**. Nous avons dans un premier temps essayé de faire autrement et de ne pas l'utiliser mais nous nous sommes finalement rabattus sur cette solution car la **documentation sur Internet était large et facilitait son utilisation.**

CHARGEMENT DE MUSIQUES

Pour cela, nous avons utilisé la **librairie SDL_mixer** (les commandes à faire pour l'installer sont dans le README). Il était important à notre sens d'ajouter un univers sonore à notre jeu, car c'est ce qui permet au joueur de s'imprégner de l'univers. Notre message est plus intéressant et passe mieux si le joueur est plongé dans le monde.
Il a fallu trouver une musique libre de droit, la télécharger, puis l'importer dans un **dossier "musiques"**.
Ensuite, il a fallu l'importer dans le code en utilisant encore une fois (comme pour les textures et les modèles) le **actualPath**.
À la suite de cela, il a fallu déclarer un **nouveau VBO pour la musique et la lancer dans le main.cpp de l'application**.
Le plus gros problème que nous avons rencontré avec l'utilisation de SDL_mixer est l'installation sur les machines, car en fonction du matériel, l'installation ne se faisait pas de la même manière. De plus, nous rencontrions souvent des problèmes de chemin vers les musiques utilisées.

CHARGEMENT DU TEXTE

Le but de notre jeu est de trouver des indices pour naviguer de monde en monde. Comme pour beaucoup de jeux, l'affichage de texte est non négligeable puisqu'il permet de comprendre où nous en sommes, que ce soit en terme d'indice ou encore de score. Nous avons donc utilisé la **librairie Freetype** pour afficher du texte. **Le principe consiste à afficher du texte, caractère par caractère pris dans un fichier .ttf et l'afficher en 2D à l'écran**. Il y a d'autres librairies disponibles, mais celle-ci était la plus simple, ne nécessitant que 2 / 3 fonctions et un shader. Une classe était nécessaire pour simplifier l'affichage du texte, **on ne stocke pas d'image ni de texte, celui-ci est remplacé à chaque fois que du nouveau texte intervient.**

CHEMIN RELATIF

Pour que le jeu puisse se compiler sur d'autres ordinateurs sans avoir à modifier le code, il faut utiliser des **chemins relatifs**. C++17 fournit Filesystem qui possède des fonctions pour extraire les chemins relatifs vers des fichiers, dossiers. Cependant, cette librairie nous a posé problème. Ainsi, **nous avons créé la fonction "actualPath" qui permet de renvoyer le chemin vers le dossier du projet.** De ce fait, si un fichier se trouve dans "assets/models" il faudra utiliser la fonction actualPath pour avoir le chemin vers le dossier du projet, que nous avons appelé "fullpath" dans le projet et effectuer fullpath + "assets/models". Ce n'est pas la meilleure méthode pour lire des fichiers, mais au moins le projet compilait sur toutes nos machines sans avoir à utiliser de fonctions supplémentaires.

AFFICHAGE DU FOND

Pour afficher le fond en Low Poly, nous avons dans un premier temps essayé d'utiliser le système de **Skybox**.
Nous avons donc créé une classe Skybox, avec deux fonctions : SkyboxInit() et LoadCube(), et un vecteur d'images. La fonction LoadCube permettait de faire correspondre les images aux côtés du cube à afficher.
Ce principe de skybox permet de créer une impression de profondeur chez le joueur.
Malheureusement, par **manque de temps pour le débug**, nous n'avons pas réussi à le faire fonctionner.
Nous avons donc remplacé cela par l'**affichage d'un modèle englobant** sur lequel nous avons posé une texture en fonction du monde affiché.

DIFFICULTÉS RENCONTRÉES

Les premières difficultés auxquelles nous nous sommes heurtées sont les **difficultés matérielles**. En effet, nous ne travaillons pas tous sur les mêmes machines (**Windows, MAC**) ce qui rendait le **débug assez difficile**. Si l'un d'entre nous rencontrait des problèmes de machine virtuelle par exemple, il était difficile pour les autres de l'aider.
De plus, Aloïs a dû réinstaller sa machine virtuelle quelques fois, et malgré cela, elle rencontrait toujours des problèmes lors de l'exécution du programme. En effet, elle pouvait difficilement manipuler les touches, et effectuer des actions dans les mondes, car l'exécutable était très lent, se figeait, et ainsi, avait du mal à "suivre". De ce fait, il lui était compliqué de tester efficacement ses bouts de codes.

DIFFICULTÉS RENCONTRÉES

Une autre difficulté que nous avons rencontrée est le **partage des fichiers sur Git Kraken.** En effet, une fois sur deux les mises à jour ne s'affichaient pas. Il fallait se reconnecter puis reload tout le repo. Certaines fois, il a même fallu supprimer le dossier avec le projet puis recloner le repo dedans. Cela s'est avéré contraignant, car il était du coup facile de mélanger des versions de fichiers.

Une autre difficulté qui a été assez handicapante à notre sens, même si commune à tous les groupes, a été le **temps, plutôt limité**, que nous avions à accorder au projet.
Nous avons beaucoup apprécié travailler sur le projet et avons énormément appris, malheureusement, nous aurions aimé avoir plus de temps pour nous concentrer sur certains détails sur lesquels nous n'avons pas eu le temps de nous pencher. Je pense notamment aux collisions ou au monde que nous aurions aimé modéliser nous-même.

AMÉLIORATIONS POSSIBLES

COLLISIONS

Pour améliorer notre projet, il serait important de nous concentrer sur le **système de collisions.** Par manque de temps, nous n'avons pas pu gérer cela dans le projet actuel, mais nous pensons qu'il serait intéressant que l'utilisateur ne puisse pas rentrer dans certains éléments du décor. Cependant, dans le jeu tel qu'il est conçu actuellement, ce manque de collisions n'est ni choquant ni dérangeant pour le gameplay.

SYSTÈME D'INDICES

Nous pourrions également essayer d'**améliorer le système d'indices** qui aujourd'hui est relativement basique, en ajoutant de la difficulté par exemple. Nous pourrions ajouter des indices, les cacher autrement, leur donner un ordre, faire **interagir le joueur** avec.

MODÈLES IMPORTÉS

Il serait également intéressant que nous essayons d'importer nos **propres modèle**s pour avoir un jeu un peu **plus personnel**, et qui ressemble à l'idée que nous nous en étions fait au démarrage du projet. Comme expliqué dans le synopsis, le jeu a un aspect "serious game" qu'il aurait été intéressant de développer, notamment au travers de **décors élaborés**. Nous aurions imaginé un monde marin, dans lequel le joueur se serait baladé dans l'océan et aurait été sensibilisé à la protection des environnements maritimes.

SCORE

Un **système de score** aurait été plus ludique aussi, en fonction du **temps mis pour finir le jeu**, ou pour finir chaque monde.

GÉNÉRATION ALÉATOIRE

Pour que le jeu soit **re jouable à l'infini**, nous aurions pu **générer un monde aléatoirement**, comme pour Minecraft, et **placer les objets aléatoirement** à une certaine distance les uns des autres. La re jouabilité est un aspect fondamental dans les jeux aujourd'hui. Il suffit de voir League of Legends, Overwatch, Dota 2. Ici générer un monde aléatoirement permettrait d'apprécier le jeu à chaque fois.

RAPIDITÉ

L'ajout de modèles avec Assimp est un ajout intéressant parce qu'il permet un niveau de détail assez élevé pour un jeu. Cependant à voir le temps que met le jeu à charger on aurait pu utiliser des techniques de programmation pour **accélérer ce temps de chargement**, comme le **multi-threading**. Mais nous manquons de connaissances à ce sujet.

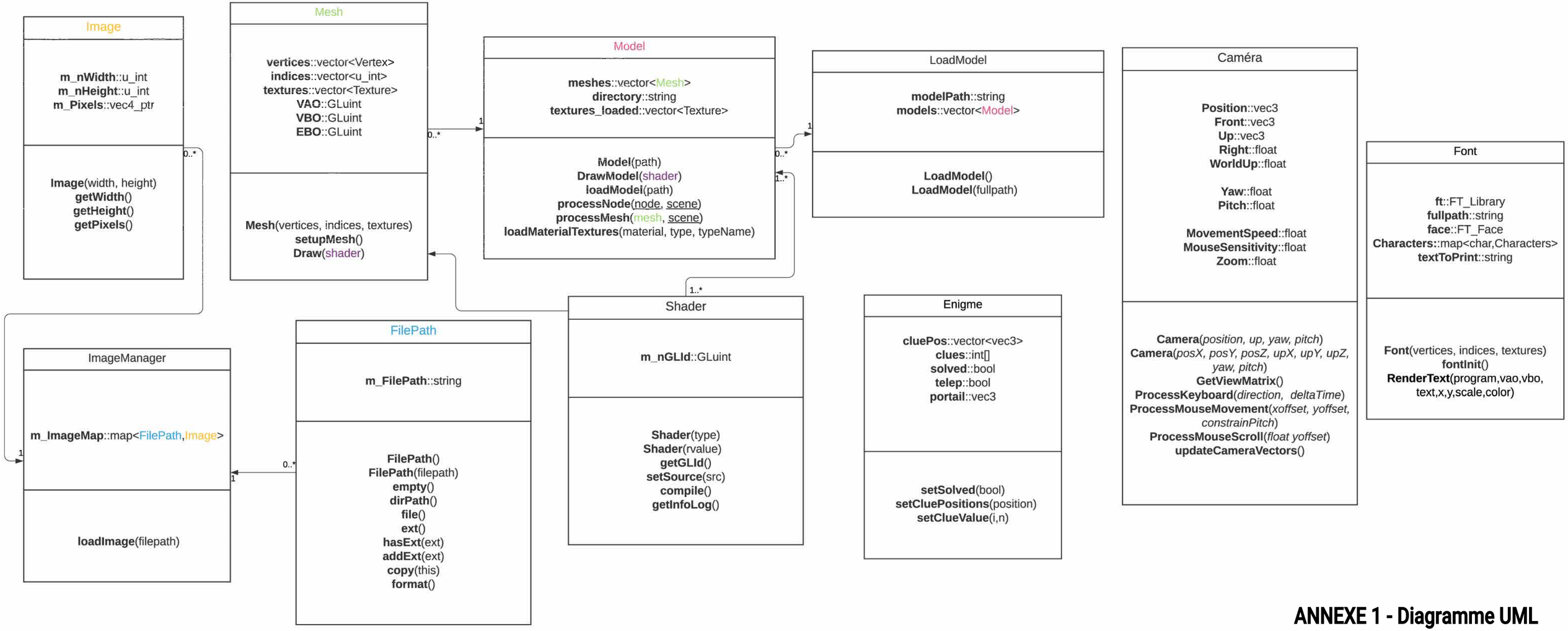
NOTICE

TOUCHES BASIQUES POUR SE DÉPLACER

“A” = devant
“Q” = gauche
“S” = derrière
“D” = droite

PREMIÈRES ÉTAPES

1 - Lors de l’éxécution, à **“A cessé de fonctionner”** --> ATTENDRE. (lenteur d’éxécution)
2 - METTRE la souris au milieu pour commencer, APPUYER sur “ESPACE” puis sur “P” --> SE BALADER
3 - Une fois à proximité des objets, APPUYER sur “E”, pour récupérer l’objet
4 - TRAVERSER les portails dès que les trois objets ont été trouvés



ANNEXE 1 - Diagramme UML