# Journal de Développement :

Développement d’un Moteur Physique en C++

## LES CHOIX

**Outils de développement :** Visual Studio, OpenGL (avec librairies Glew et Freeglut)

L’IDE de développement sélectionnée est Visual Studio car nous l’avions déjà installé chacun de notre côté. Concernant la librairie graphique, nous utilisons OpenGL car Martin et Clémence l’avaient déjà utilisée l’année précédente dans d’autres projets sous QtCreator.

Nous avons toutefois dû adapter son utilisation à Visual Studio, via les bibliothèques *glew* et *freeglut*.En effet, QtCreator encapsulait OpenGL de telle manière que son utilisation ne requérait rien de plus que l’IDE de développement.

## LES DIFFICULTÉS

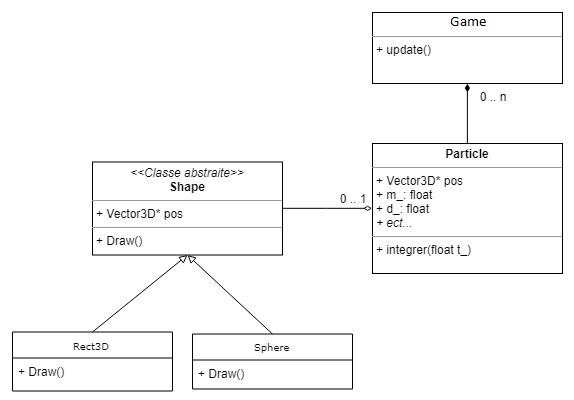
Nous avons rencontré diverses difficultés durant le développement :

* L’utilisation d’une librairie graphique de bas niveau requiert plus d’effort pour l’affichage, et avons donc passé du temps à programmer l’affichage et rajouter une couche d’abstraction à notre code.
* Comme *freeglut* ne supporte pas bien l’encapsulation de ses fonctions de callback (la librairie étant en C, elle émet des erreurs si on appel des fonctions appartenant à des instances d’objets), nous avons dû mettre en place un hotfix avec une instance statique de la classe gérant l’affichage graphique. Ce n’est pas une solution élégante mais nous avons préféré ne pas perdre trop de temps sur l’affichage, car ce n’est pas l’objectif premier de notre travail.
* Le C++ étant un langage nouveau pour Elie, il lui a fallu apprendre la syntaxe et autres spécificités du langage au fil du projet.
* Nous avons également rencontré des certains problèmes liés aux pointeurs, notamment liés à deux classes se partageant une instance de Vector3D, problèmes assez récurrents lors du développement d’un programme en C++.

## CHOIX D’IMPLEMENTATION

La séparation a été faite entre la logique et l’affichage, une classe **Game** gère la majeure partie de la simulation, et possède une liste d’objets **Particle** qui contient les données importantes à la physique, qui peuvent chacune avoir un objet **Shape** gérant leur affichage.

Un UML **simplifié** du système :



Rect3D et Sphere sont des classes filles de Shape implémentant une fonction Draw() qui trace via OpenGL leur forme à la position de la particule.

La classe Jeu fait le lien entre les différentes classes, fait les appels de l’intégrateur sur chacune des particules puis dessine la scène.