Document de conception de moteur physique

Groupe C

Réalisé par : Antoine De Saint Martin

Victor Guiraud

Axel Guza

Table des matières

Contexte : Dans le cadre du cours « 8INF935 – Mathématique et Physique pour le jeu vidéo » nous devons réaliser un moteur physique.

La conception

Durant la conception nous avons pris beaucoup d'inspiration du livre de référence du cours (« I. Millington. Game physics engine development»), d'information et d'algorithmes de forums internet et du cours de notre professeur afin d'avoir les bases nécessaires en mathématique pour la bonne réalisation du projet.

Notre projet Visual Studio est composé de classes divisées en fichiers headers et cpp. Les fichiers .h stockent les prototypes des fonctions que nous allons développer dans les fichiers cpp qui contiendront les algorithmes qui supporteront le fonctionnement de notre engin.

Vecteur3D:

Dans la classe Vecteur3D nous retrouvons les déclarations de nos fonctions avec les différents axes x, y et z qui vont nous permettre de positionner nos vecteurs, les orienter et de les déplacer dans un repère orthonormé de l'espace. Cette classe va également nous permettre d'effectuer des calculs sur les vecteurs (normalisation, produit etc.) et d'être appliqué sur les particules et forces présentes dans la physique du moteur. Il s'agira du point initial de la physique de notre engin qui sera utilisé dans de nombreuses classes, un peu comme une boîte à outils.

Particules :

La classe particule va permettre de définir nos objets 3D qui vont naviguer dans l'espace avec des propriétés qui lui appartiennent (ses coordonnées, sa masse, sa vitesse, sa direction, son amortissement, ses forces, etc.).

Chaque image du rendu prend un certain temps à être calculée et nous nous basons sur cette dernière afin de préparer les prochaines positions et vitesses de notre particule (lastFrameDuration est le pas de temps qui sera multiplié à la vitesse pour

la position et à l'accélération pour la vitesse). Dans notre programme nous utilisons un système de frameRate afin que toutes les images aient la même durée.

Nous avons aussi implémenté le système de dumping afin de simuler les frottements appliqués sur notre particule (frottements du milieu par exemple, air, eau…).

Balistique :

Balistique est une classe regroupant tous les tirs de particule que nous pouvons retrouver dans notre simulation (Laser, Pistol, Artillery, Fire Ball et Arrow) qui ont toutes des caractéristiques différentes. Par exemple Artillery aura une trajectoire parabolique en raison de sa masse alors que le laser ayant une masse quasi nulle (valeur très faible car c'est un rayon lumineux mais non négligée car les calculs seraient impossible) va juste parcourir une trajectoire rectiligne.

Les différents tirs ont des valeurs par défaut qui sont modifiables dans l'interface IMGUI par l'utilisateur une fois le projet lancé (voir le README pour plus de précisions).

Chaque tire parcourt une distance, si elle dépasse la taille du terrain qui est attribuée (ici 300m), si elle touche le sol (y = 0) ou si elle est instanciée puis trop longtemps, on lui change son type en UNUSED qui la remet dans la pile des particules prêtent à être lancées. Ses attributs seront réinitialisés lorsqu'un nouveau tir l'utilisant sera projeté.

Timing :

Cette classe contient toutes les informations de gestion du temps (horloge ,temps de dernière frame du dernier objet…). Elle sera très utilisée dans la gestion du temps de la fonction update() de Main.cpp lorsque l'engin sera en cours de fonctionnement. Ce script a été récupéré sur un répertoire git l'utilisant.

Main :

Le Main.cpp est la classe principale qui permet de faire tourner tout le programme. Dans cette classe nous allons trouver différents algorithmes et blocs de tests. Dans un premier temps nous avons dû tester les méthodes implémentées dans le classe Vecteur3D, ce qui correspond au bloc 1. Dans un second temps, via l'affichage sur console, le deuxième bloc de test est utilisé afin de vérifier le bon fonctionnement de la classe Particule, la bonne intégration des positions et des vitesses ect…Ce test était réalisé avec un rafraîchissement à temps variable (*lastFrameRateDuration*), ce temps correspond au temps pris par le processus pour faire évoluer les valeurs des Particules ainsi que pour les afficher dans la console. Dès qu'un passage dans la boucle *While{}* est terminé, il recommence. Le troisième bloc correspond également au test de la classe Particule, mais cette fois-ci avec un rafraîchissement à temps fixe, nous avons fixé ici le rafraîchissement à 30 images par seconde. Si il prend moins de temps à calculer et à afficher, il doit attendre le temps restant avant de repasser dans la boucle (bien sûr ici, avec un programme simple, la boucle s'effectue en 3ms en moyenne, il doit donc attendre 30ms avant de continuer son exécution).

C'est dans cette classe que nous allons retrouver les programmes d'affichages de menu dearIMGUI et de graphisme 3D à l'aide d'OpenGL. Différentes méthodes sont utilisées, une gère l'affichage de notre repère X,Y,Z et fixe le point de vue de la caméra (*LookAt(..)*), ce point de vue est d'ailleurs réglable afin de se déplacer dans l'espace autour du repère. Dans la partie principale de notre fonction Main (la partie gérant les tests finaux), un *While{}* gère la boucle de jeu, la boucle d'exécution, interruptions clavier et souris, calculs des attributs de nos particules et affichage de celles-ci. Un ensemble de méthodes IMGUI permettent à certaines valeurs renseignées d'être modifiées dans la fenêtre utilisateur, ces valeurs sont ensuite passées aux méthodes respectives. L'affichage des particules s'effectue dans une boucle *For{}*, chacune des particules est passée en paramètre de cette fonction et elle les affiche une à une.

Pour ce qui est du retour utilisateur sur le comportement des projectiles, le retour graphique ne suffit pas, nous avons donc un retour console. Lorsqu'une particule ne respecte plus les conditions d'évolution (limites de terrain, temps depuis initialisation…), l'utilisateur en est informé. Il saura qu'elle particule vient de disparaître, pour quelle raison, et le temps passé depuis son instanciation (par exemple : La particule de type ARTILLERIE a touché le sol en N secondes).

Nos outils

Pour la bonne réalisation du projet nous utilisons un « Trello » qui est un outil de gestion de projet en ligne et c'est ce qui nous permet d'avoir une bonne répartition des tâches.

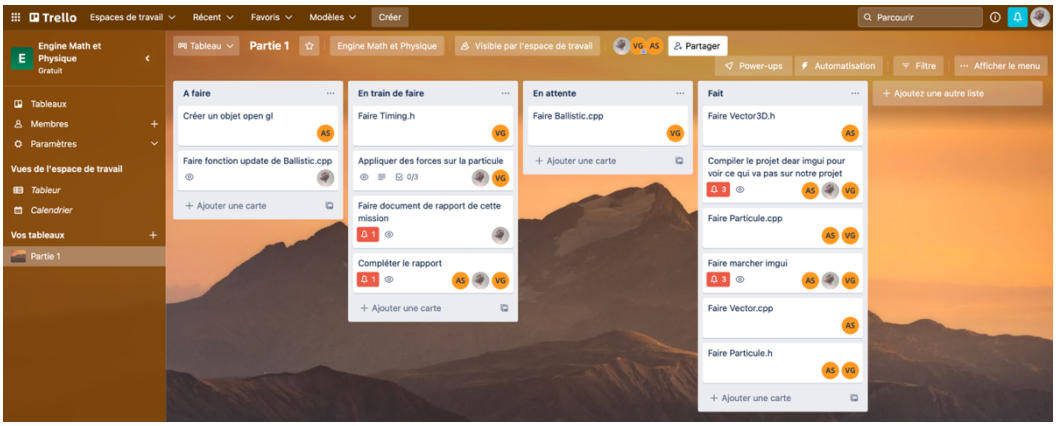

Figure 1 - Gestion de Projet - Trello - @Axel GUZA

Nous utilisons plusieurs librairies indispensables à notre projet comme OpenGL3 glfw3 et slfw qui fournit une API simple pour créer des fenêtres, des rendus d'écran, des dessins objets, etc.

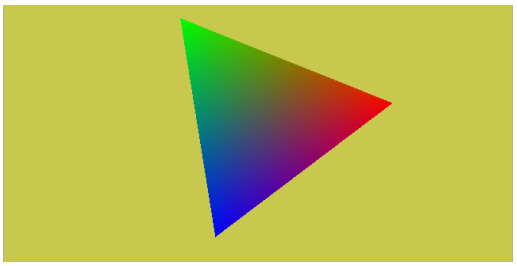

Figure 2 - Triangle fade - OpenGL - @Sdaau

Nous utilisons aussi IMGUI qui est une interface utilisateur en mode immédiat qui nous permettant de gérer des données dynamiquement dans notre projet. Ceci va permettre d'influencer les composants du monde 3D du logiciel via une interface.Nous pouvons changer la couleur de la forme, sa position, son angle etc. Il permet d'influencer virtuellement tous les aspects de notre code et d'obtenir des résultats visuels en temps réel.

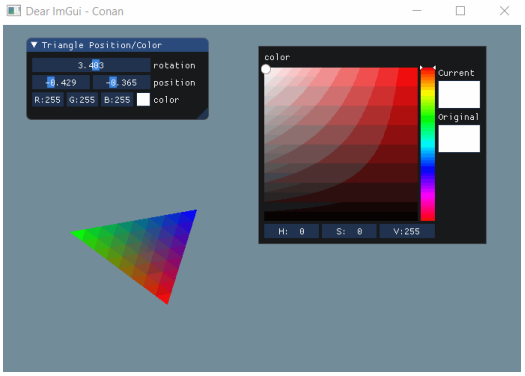

Figure 3 - utilisation de l'interface de IMGUI - OpenGL - @Conan.io

Les erreurs rencontrées

Nous avons tous rencontré des erreurs ou des problèmes à différents niveaux. Les débuts du projet ont été difficiles pour plusieurs d'entre nous pour la répartition des tâches, le fonctionnement des librairies et les connaissances sur les outils utilisés.

Axel a rencontré pas mal de soucis avec l'utilisation de son mac pour les programmes du projet. Afin d'essayer de le faire fonctionner il a essayé plusieurs méthodes :

- Il a d'abord tenté de lancer le programme sur Visual Studio 2019. Cependant l'outil lui permettait uniquement de débuggé la solution mais pas de la lancer.
- Il a ensuite implémenté OpenGL sur Xcode afin de lancer les programmes de base mais en raison de problèmes de compatibilité entre Xcode et Visual Studio nous ne pouvions pas développer ensemble.
- Il a ensuite essayé d'utiliser une machine virtuelle pour avoir un système d'exploitation Windows qui malheureusement fonctionne très mal sur sa machine.

Ayant pas de machine pour développer, il à contacter un de ses professeurs de projet afin d'avoir une salle avec poste de travail Windows rapidement pour pouvoir travailler sur le moteur et d'autres projets dans le cadre d'autre cours. Nous avons désormais accès à cette salle. Mais dans le cadre de cette première étape
il a travaillé en pair programming avec son collègue Victor.

Pour Victor et Antoine ils ont rencontré des problèmes avec l'installation des librairies OpenGl et IMGUI. Tout d'abord ils ont essayé d'intégrer les librairies par les « nuggets » contenus dans Visual Studio 2022 dont l'installation n'a pas fonctionné. Pour contourner le problèmes ils ont ensuite télécharger les paquets depuis internet. Nous avons d'abord installé IMGUI (https://github.com/ocornut/imgui) , puis ensuite OpenGL (https://www.opengl.org/). Nous avons rencontré des soucis dans l'intégration de ses modules qui nous a demandé de changer des paramètres du projet afin de les accueillir. Pour au final ouvrir des fenêtres et faire afficher des objets OpenGL et interfaces IMGUI nous avons dû rechercher plusieurs échantillons de codes fonctionnels sur internet afin de s'en inspirer.

Antoine a ensuite rencontré des problèmes avec l'utilisation de la librairie GLUT, fonctionnant de paire avec OpenGL, des erreurs mémoire apparaissaient lors de l'utilisation des fonctions de la librairie. Après plusieurs vérifications sur les pointeurs utilisés (aucun n'était *null* lors du passage en paramètre, les .dll et .lib étaient tous dans le projet et dans les fichiers du système), il s'est résolu à utiliser une différente librairie. Les fonctions pour récupérer les interruptions clavier, souris n'étaient pas utilisables, la boucle d'exécution de GLUT non plus. SFML est la librairie adéquate (https://www.sfml-dev.org/) permettant la récupération des interruptions, pour la boucle d'exécution, la fonction de GLUT est simplement remplacée par un *While{}* ect…
D'autres problèmes mineures sont présents, lorsque l'on se déplace dans l'espace avec les réglettes X, Y, Z, nous pouvons faire disparaître le texte indiquant la projectile sélectionné. Dans la structure d'un projectile, une couleur d'objet était appliquée à chaque type, la couleur ne fonctionne pas à l'affichage. Et lorsque nous lançons l'application, dans la console, nous passons dans une boucle faisant affichant quelques fois une phrase *std::cout* << " ", le passage dans cette boucle n'est pas nécessaire et n'a pas été enlevé.

Source :

- https://www.youtube.com/watch?v=Du--cH01ZWI
- https://i.stack.imgur.com/LVHBo.png
- https://github.com/ocornut/imgui
- https://www.opengl.org/
- https://moodle.uqac.ca/mod/url/view.php?id=653865
-

Les librairies :

-https://www.sfml-dev.org/ : SFML librairie actuellement utilisée.

-https://www.opengl.org/resources/libraries/glut/glut_downloads.php : librairie GLUT appairée à OpenGL.

-https://www.just.edu.jo/~yaser/courses/cs480/opengl/Using%20OpenGL%20and%20GLUT%20in%20Visual%20Studio.htm : utilisation de GLUT et OpenGL sur Visual Studio.

-https://www.dll-files.com/glut32.dll.html : récupération .DLL GLUT.

-https://web.eecs.umich.edu/~sugih/courses/eecs487/glut-howto/#win : placement des .DLL dans les fichiers systèmes.

-https://www.opengl.org/resources/libraries/glut/glut_downloads.php : librairie GLUT et documentation.

Les problèmes rencontrés :

-https://openclassrooms.com/forum/sujet/visual-studio-erreur-opengl-sfml : problème de violation de mémoire avec GLUT.

-https://support.microsoft.com/fr-fr/topic/vous-recevez-un-message-d-erreur-violation-d-acc%C3%A8s-0xc0000005-0b5b76a3-b15b-5d2b-221f-af2d28badf1b : problème de violation de mémoire avec GLUT.

Les ressources de répertoires GIT et livre :

-https://github.com/idmillington/cyclone-physics/blob/master/src/demos/main.cpp : répertoire GIT du livre associé au cours, Millington. Game physics engine development.

-https://ebookcentral.proquest.com/lib/uqac-ebooks/home.action : livre de référence du cours.

Les sites divers :

-https://learn.microsoft.com/fr-fr/troubleshoot/windows-client/deployment/dynamic-link-library : renseignements sur le DLL avec Visual Studio.

Aides sur les librairies :

-https://learnopengl.com/Getting-started/Hello-Triangle : commencement sur OpenGL.

-https://dokipen.com/modern-opengl-part-3-hello-triangle/ : commencement sur OpenGL.

-https://en.cppreference.com/w/cpp/header : librairies de bases c++.

-https://www.developpez.net/forums/d1108488/applications/developpement-2d-3d-jeux/api-graphiques/opengl/dessiner-repere-xyz/ : renseignements élaboration repère avec OpenGL.

-https://www.opengl.org/resources/libraries/glut/spec3/node14.html : renseignement sur GLUT.

-https://www.geeksforgeeks.org/dos-h-header-in-c-with-examples/ : gestion du temps en boucle.

-https://cpp.hotexamples.com/fr/examples/-/-/GET_XBUTTON_WPARAM/cpp-get_xb-utton_wparam-function-examples.html : récupération interraction souris.

-https://learn.microsoft.com/en-us/windows/win32/learnwin32/keyboard-input : récupération interraction clavier.

-https://learn.microsoft.com/en-us/windows/win32/learnwin32/mouse-input : récupération interraction souris.

-https://en.cppreference.com/w/cpp/chrono/system_clock : gestion du temps en c++.

-https://community.khronos.org/t/glcolor3f-not-working/69751/2 : coloration des objets avec OpenGL.