Project-MFB By URSI

1) Résumé du projet :

Le **Projet-MFB**, est une démonstration technique ayant comme inspiration ***Wolfenstein 3D*** **(1992)** et ***DOOM*** **(1993)**.

Le projet repose principalement sur la mécanique du raycasting, qui consiste à l'envoi de rayon dans de multiples direction afin de détecter l'intersection avec le mur le plus proche dans le but de générer un rendu 3D.

Les objectifs sont :

- Un raycasting fonctionnel
    - Car sans lui, nous n'avons simplement pas de projet.
- Un code qui compile
    - Si le code ne compile pas, il n'y a pas de projet non plus.
- Un système d'arme fonctionnel
    - Ce système devra intégrer au moins deux armes avec des caractéristiques différentes.
- Des NPCs capables de se déplacer
- Un moteur de son
    - Le son est une variable importante pour l'immersion du jeu.
- Une map à taille variable
    - Map qui sera stockée dans un fichier séparé
- Des textures transparentes
    - Si elles ne le sont pas, le rendu ne sera pas propre.

Le projet nous apporte une connaissance approfondie de la librairie graphique GFX, ainsi que l'expérience d'une période de « crunch », pour la réalisation d'un projet ambitieux avec un timing serré. Également, nous avons des connaissances concernant le multithreading.

Pour l'interaction homme-machine, les contrôles sont ZSQD pour le déplacement, et le clic gauche de la souris pour le tir.

2) Tâches :

1° Mise en place du raycasting :

Pour mettre en place le raycasting, il faudra faire un ensemble de fonction capable d'envoyer des rayons dans une multitude de directions, et qui, en fonction de leur réaction (rencontre d'un mur ou non, à quelle distance se trouve le mur…), sera capable de générer un affichage donnant une illusion de 3D.

2° Collisions :

Pour mettre en place les collisions, il faudra, en fonction du raycasting, créer une zone infranchissable pour le joueur là où se trouvent les murs.

3° Map :

Pour générer la map, nous l'avons stockée dans un fichier séparé et nous avons utilisé malloc pour stocker la valeur des cases dans un tableau.

4° NPCs :

Pour les NPCs, nous allons faire une fonction de déplacement aléatoire en direction du joueur, puis une fonction de tir, par rapport à leur arme. Ces personnages seront agressifs.

5° Moteur de son :

L'objectif du moteur de son est de jouer des bruits d'ambiance, tels que des bruits de pas, de tirs, ainsi qu'une musique, sans bloquer le fonctionnement du jeu.

6° Armes :

Avec les armes, le but est d'en assigner trois au joueur. Un revolver classique, un fusil ainsi qu'un dubstep gun, une arme qui réagit à la musique en cours, et qui joue du dubstep. Il faudra aussi utiliser un raycasting pour gérer la distance par rapport aux NPC lors du tir.

7° Structures :

Ayant découvert l'héritage de classes du C++ lors du cours de POO en fin de semestre, nous allons tenter d'implémenter un système similaire en C afin de simplifier la lecture du code.

8° HUD :

Pour le HUD, l'objectif de notre équipe est d'afficher un HUD avec un nombre de points de vie, de munitions et de notre arme courante, un peu dans le style de celui de Wolfenstein 3D.

9° Implantation des textures :

Ici, le défi sera d'arriver à plaquer les textures sur les murs générés par le raycasting. De même, il faudra arriver à réduire la taille des ennemis lorsqu'ils sont loins.

10° Opacité des textures :

Pour obtenir un bon rendu sur le HUD et sur les NPCs, il faudra gérer la transparences des textures. Pour cela, le fond des images en BMP sera d'une couleur que nous n'utilisons pas (le fuchsia), et avec un algorithme de transformation du BMP en ARVB, les pixels fuchsia seront d'une opacité égale à 0.

3) Attribution des tâches :

Quentin : Raycasting, collisions

Kylian B. : Map, moteur de sons

Kylian C. : NPCs, Structures, HUD, logique des armes

Clément : Opacité des textures, implémentation des textures

4° Chronologie :

Raycasting -> ± 2 semaines

Collisions -> ± 6 heures

Structures : ± 3 heures

Map -> ± 3 heures

NPCs -> ± 5 heures

Moteur de son -> ± 2 jours

Armes -> ± 5 heures

Opacité des textures -> ± 9 heures

HUD -> ± 4 heures

5° Exigences spécifiques :

Le système de sons nous as obligé à utiliser le multithreading, car les sons bloquaient le fonctionnement du programme ; c'est une mécanique que nous avons utilisée régulièrement.

Github du projet :