DOCUMENTATION TACHE TP_CMAKE/DIRECTX

COMMENT LANCER LE PROJET

- 1er Solution (a la main) :

il faut se mettre dans un terminal a la racine du projet

Créer un dossier « Build » : mkdir Build

Puis se mettre dedans : cd Build

A la suite de cela exécuter les 2 commandes CMake suivante :

- cmake ../App
- cmake --build .

Enfin il ne vous reste plus cas lancer le projet .sln, soit par le raccourci a la racine ou bien dans le dossier Build

- 2e Solution (exécutable) :

A la racine du projet il y a un exécutable (LaunchBuild.bat) il suffit de l'exécuter pour créer un dossier Build avec le projet Cmake dedans.

Enfin il ne vous reste plus cas lancer le projet .sln, soit par le raccourci a la racine ou bien dans le dossier Build

RECAPITULATIF TACHE FAITE

CMake :

- Créer un projet CMake qui génère une solution Visual Studio 2022 séparer en 2 projets : App et Tools. **FAIT**

- Le projet Visual Studio démarre avec comme target le projet App. - Les fichiers .h et .cpp sont inclus individuellements dans le CMakeLists.txt. **FAIT**

- Le projet utilise le standard CPP 20, une vérification sur la version CMake nécessaire est faite. - Sont ajouter en dépendance de App le projet Tools et les librairies windows "d3d12.lib" "dxgi.lib" "dxguid.lib". **FAIT**

- Un fichier .h est généré à partir d'un fichier de configuration CMake, celui-ci contient en variable les "VERSION_MAJOR" et "VERSION_MINOR" du projet. **FAIT**

- Bonus, ainsi que :

    - Le path précalculé par CMake du dossier de projet. **FAIT**

    - Un Define pour activer ou désactiver les logs. **FAIT**

C++ :

- L'application définit une classe Windows à utiliser pour son programme. **FAIT**

- Le nom de la fenêtre inclus "Rendu CMake 2024 + le numéro "VERSION_MINOR" récupérer de CMake. **FAIT**

- La fonction main défini une boucle d'exécution du programme, celle-ci ne s'arrête que lorsque la fenêtre windows à été fermée. **FAIT**

- Lors de l'appuit sur le bouton fermé, la boucle main fini et la fenêtre est détruite ensuite. **FAIT**

- Une class static de log est présente, elle écrit dans la console des messages. **FAIT**

- Bonus :

    - La classe de log utilise le define bonus CMake pour log ou non les messages **FAIT**

    - Un DebugLayer D3D12 et DXGI track les erreurs DirectX12 et les instances DXGI présentes dans le programme **FAIT**

- Chaque fois que la fonction main repasse dans la boucle principale d'execution, elle calcule un deltaTime double du temps passé depuis le dernier passage, cette valeur est logger avec la classe de log - Si la touche, z, q, s ou d est enfoncé, alors un log est écrit dans la console précisant quel touche a été appuyée. **FAIT**

RECAPITULATIF TACHE FAITE

Les taches principales étant des mises en place déjà faite cela fut plutôt rapide. Également pour la création de la fenêtre et du Debugger vue qu'il fut travaillé pendant les cours.

Les difficultés ont été rencontré dans la réalisation des taches bonus. La mise en place du path du dossier du projet en Variable dans le fichier AppConfig.h.in fut rapide cependant j'ai eu quelque porbleme avec la création d'une option utilisé ou non le Debug. J'ai omis de mettre une certaine ligne dans le cmake faisant que je ne pouvais pas l'utiliser dans le code C++, j'y suis rester un certain temps avant de le voir.

Pour ce qui est du calcul et l'affichage du deltaTime, le calcul me pris un peu de temps, rien d'important, mais surtout la conversion en string pour pouvoir l'utiliser dans un log m'a pris un temps fou.

Enfin pour ce qui est du dernier bonus qui est le log quand nous appuyons sur Z/Q/S/D, je n'ai pas réussi a utilisé les define mise à disposition dans l'énoncé (je n'arrivé pas à rentrer dans la case), je me suis renseigné sur comment faire pour une window et c'est là que j'ai vue une autre technique pour la détection des touches qui m'a permis de l'implémenter facilement. Ducoup avec le log du DeltaTime on ne vois pas bien les logs des touches donc je conseille de le commenté juste pour voir la différence.

Github si besoin : https://github.com/senswann/TP_DirectX_M1