Projet de Fin d'Année

Version	Deadline
Alpha	23/05/2020 - 23h59
Beta	01/06/2020 - 23h59
Gold (Fin du projet)	12/06/2019 – 23h59
Présentation du projet	15/06/2019 - 14h00

Nombre d'étudiants par groupe : 4

Rendu pour la Gold :

- Fichiers :
  - toutes les sources du projet
  - Des CMakeLists.txt qui permettent de tout compiler sans problème ni warnings.
- Dossiers :
  - "**libgfx**" : le répertoire de votre lib graphique
  - "**app**" : le répertoire de l'app de votre jeu
  - "**vendor**" ou "**thirdparty**" : le répertoire contenant toutes les libs externes que vous utilisez
  - "**resources**" : le répertoire contenant tous vos assets (mesh 3D, textures, sons…)
  - "**doc**" : le répertoire de vos diagrammes UML et analyse prévisionnelle du projet.
  - tout autre dossier nécessaire à la compilation de votre projet

Rendu attendu pour la présentation :

- Dossiers :
  - "**Screens**" : le répertoire qui contiendra des screenshots de votre jeu
  - "**Videos**" : le répertoire qui contiendra des vidéos de gameplay (au moins 1 vidéo de 30 secondes minimum **sans montage**).
  - "**Presentation**" : le répertoire contenant la présentation finale montrée par les étudiants.

GIT

N'oubliez pas de tagger vos milestones ALPHA | BETA | GOLD sur la branche master !!

Sujet

Le but de ce PFA est de vous faire travailler toutes les compétences que vous avez acquises cette année en C++, 3D et détection de collision.

Cette année, le PFA est un FPS en C++/OpenGL sans utiliser de moteur 3D ou physique externe :

Le projet sera de développer une partie de la version classique du jeu Minecraft. On pourra se déplacer dans un monde 3D composés de blocs 3D texturés. Nous pourrons détruire des blocs et en placer d’autres.

Règles générales

Voici les contraintes à impérativement respecter pour ce projet :

Organisation :

- La durée du projet est de 5 semaines. Du **11 mai** jusqu'au **12 juin inclus**.
- Le groupe devra élire un **Lead Tech**. Il aura pour fonction de prendre la décision finale technique dans le groupe.
- Il y aura 3 milestones : **ALPHA**, **BETA** et **GOLD**.
- chaque Lundi, l'équipe effectue un point d'avancement avec le professeur.
- Le **15 juin**, les équipes présenteront un diaporama de leur projet aux enseignants.

Technologie :

- Le projet doit être développé sur Windows ou Linux avec C++ et OpenGL
- Le projet de l'équipe devra reprendre un des moteurs développé pour le Platformer. L'objectif est de ne pas repartir from scratch mais plutôt de reprendre et améliorer l'existant.
- **Vous devez programmer toutes les détections de collisions vous-même.**
- Vous n'avez le droit à **aucune autre librairie externe**, exceptée les suivantes :
    - **SDL**, **SFML** ou **GLFW** pour le fenêtrage
    - **Freetype** pour le chargement des fonts
    - **STB_image** ou **SOIL** pour charger des textures
    - **GLM** pour les mathématiques
    - **Assimp** pour charger des modèles 3D
    - **TinyXML** ou **RapidJSON** pour charger des fichiers de config / sauvegarde
    - **irrKlang** pour lire et jouer des fichiers sonores
    - **ImGUI** pour la GUI.
    - (bien sûr, vous pouvez aussi choisir d'utiliser votre propre implémentation.)

Features :

Votre jeu doit implémenter plusieurs fonctionnalités de Minecraft :

- **Génération aléatoire du terrain** : il faut que l'on puisse charger un terrain à taille fixe

- **5 types de blocs** : Dirt, Gravier, Bois, Lave, Eau.

- Le personnage joueur doit avoir des déplacements classiques d'un personnage de FPS avec ZQSD (ou WASD). Le joueur doit évidemment ne pas traverser les murs, subir la gravité, et se faire mal s'il tombe de trop haut. Il peut également sauter avec un peu d'air control.

- La possibilité de **casser** et **poser** des blocs (de manière logique, pas de blocs sur l'eau). L'objet couramment équipé (pelle, arc…) doit être visible à l'écran (pas besoin de l'animer).

- L'**interface utilisateur**
    - Main Menu : new game, load game, options.
    - In Game : HP, munitions...
    - Inventaire

- **Système d'inventaire** : implémenter une version simple de l'inventaire de Minecraft
    - (limiter la stack à 100 par case dans l'inventaire)

- Système de **sauvegarde** et de **chargement** de partie

- Les bases du **crafting** :
    - Coffre
    - une arme (épée, pelle…)
    - En bonus : l'arc et les flèches, et la torche qui agira comme une point light.

- Le jeu doit pouvoir spawn des ennemis dont le comportement est le suivant :
  - Ils se dirigent droit vers le joueur
  - Lorsqu'ils sont proches du joueur, celui-ci subit des dégâts (X dommages par seconde)
  - Tout comme le joueur ils doivent avoir des collisions pour ne pas rentrer dans les murs
  - Un seul type d'ennemi avec un mesh statique suffira (inutile de l'animer)

- Le joueur doit pouvoir équiper une arme et taper avec. Si il touche un ennemi, ce dernier doit perdre de la santé. Si l'ennemi n'a plus de santé, il meurt (disparaît simplement). Les armes se cassent après trop d'utilisations.

- Lorsque le joueur meurt, un message “You Died” s’affiche pendant quelques secondes et le joueur respawn à un endroit prédéfini.

- Une BGM (background music) calme est jouée en permanence. Des bruitages spatialisés en 3D peuvent être ajoutés (marche, attaque, rivière, etc..)

- Le jeu pourra implémenter d’autres éléments de votre choix (arbres, animaux...)

Jouabilité :

Vous développez un jeu et non une démo technique, il doit donc être jouable, le gameplay précis et agréable.

Performances :

Le jeu doit bien sûr ne pas souffrir de problèmes de performances (60 FPS constant).

Planning

1. Documentation : 14 mai

La documentation va servir à anticiper les difficultés que vous allez rencontrer.
Vous allez devoir brainstormer pour créer des diagrammes UML et une documentation de l’état actuel de votre moteur.

Un document devra recenser les différents sujets :

- **Lister les membres de l’équipe** ainsi que les **rôles attribués**
  - responsable UI, Rendering, Gameplay, Core...
- **Analyse du moteur**
  - Identifier les problématiques actuelles
- **Lister les actions** à faire pour **régler** les problèmes du moteur actuel
- **Analyse du projet**
  - Utilisez une application d’UML comme Lucidchart ou Star UML 2 pour créer des diagrammes détaillant votre architecture “idéale”
  - Lister les problématiques à résoudre découpées en tâches avec une estimation de temps accompagnée d’un “indice de confiance” (certain / incertain / très incertain).

Pour le jeudi 14 mai au plus tard, il faudra remettre à l’enseignant ce document et attendre sa validation avant d’aller plus loin.
Ce document devra être fourni dans le dossier “doc” sur GIT en **.docx** et **.pdf.**

2. Alpha : 23 mai

Sont attendus pour l'alpha :

- l'essentiel d'un **character controller FPS** :
    - Se déplacer avec les touches **WASD**
    - Sauter avec la barre d'espace
    - Diriger la caméra avec la souris.
- Le jeu devra pouvoir générer un terrain simple (non aléatoire).
- Au moins un type de bloc devra être implémenté.
- Un début de GUI.
- L'inventaire doit être fonctionnel même si pas de GUI / pas intégré au reste du jeu.
- Système de sauvegarde basique.
- Un ennemi doit être présent même si il n'a aucune IA.

3. Beta : 1er Juin

Sont attendus pour la beta :

- La majorité de la GUI
- les bases de **destruction** et de **placement** de blocs.
- Début de génération aléatoire du terrain (avec des variations de hauteurs). En bonus : vous pouvez utiliser une height map, générée avec un bruit de Perlin (voir liens utiles).
- une arme doit être présente même si pas finie.
- On devrait avoir au moins un "proof of concept" du crafting même si l'UX n'est pas finale
- Le bloc d'eau peut être implémenté partiellement : couler et se déplacer dedans avec une certaine résistance.
- Le bloc de gravier peut être implémenté partiellement aussi.
- Au moins une BGM (background music) qui tourne en boucle et un bruitage dynamique spatialisé en 3D.
- Un ennemi qui bouge.

4. Gold : 13 Juin

Votre projet devra contenir la majorité des features et avoir le moins de bugs possible.

Il est attendu une génération procédurale du terrain avec variation logique (dirt / eau / etc.).

Toute tentative d'optimisation sera valorisée : chunks, sauvegarde de la map en format compressé (Run-Length Encoding par exemple), rendu utilisant le frustum culling et/ou l'occlusion culling...

Bon courage !!!

Liens utiles

SON

https://learnopengl.com/In-Practice/2D-Game/Audio

https://www.ambiera.com/irrklang/tutorial-helloworld.html

TEXTE

https://learnopengl.com/In-Practice/Text-Rendering

https://www.freetype.org/freetype2/docs/tutorial/index.html

MODELS

https://learnopengl.com/Model-Loading/Assimp

VOXEL ENGINE

https://en.wikibooks.org/wiki/OpenGL_Programming/Glescraft_1

https://weigert.vsos.ethz.ch/2019/10/27/homebrew-voxel-engine/

https://www.gedge.ca/dev/2013/12/01/what-does-the-voxel-say

https://www.youtube.com/watch?v=Xq3isov6mZ8

https://www.youtube.com/watch?v=4Rg1RriQZ9Q

https://sites.google.com/site/letsmakeavoxelengine/

https://www.azurefromthetrenches.com/a-simple-voxel-engine-christmas-2016-project/

https://flafla2.github.io/2014/08/09/perlinnoise.html

https://cmaher.github.io/posts/working-with-simplex-noise/

http://glm.g-truc.net/0.9.5/api/a00177.html

ASSETS

https://www.minecrafttexturepacks.com/smoothic/