Travail pratique #02

Ce travail doit être réalisé en utilisant comme point de départ le petit engin de jeu présenté en classe. L'engin est basé sur OpenGL, et utilise les librairies suivantes :

SFML Librairie permettant de créer le contexte OpenGL (la fenêtre du jeu), à gérer les entrées clavier et les mouvenents/clicks de souris, le temps, etc. Une de ses grande qualité est d'être multi-plateforme, ce qui nous assurera que notre jeu peut fonctionner autant sur Windows que sur Linux ou Macintosh.

Devil Cette librairie est utilisée pour le chargement des images qui seront utilisées comme textures dans le jeu. Cette librairie est utilisée dans l'industrie du jeu vidéo parce qu'elle est flexible et supporte la majorité des formats d'images. Elle est elle aussi portable sur plusieurs OS.

La partie de la création du contexte opengl, du rendu graphique, de la gestion du clavier et de la souris (le *game loop*) est définit dans la classe abstraite OpenglContext. Notre jeu, qui sera contenu dans la classe Engine héritera de cette classe et devra s'assurer d'implémenter correctement chacune des méthodes virtuelles de la classe de base. La classe OpenglContext vous est fournie, et la classe Engine avec une implémentation minimale l'est aussi. Vous ne devez pas modifier la classe OpenglContext.

Par défaut, du code est ajouté pour gérer trois raccourcis claviers qui sont soit utiles, ou qui permettront de diagnostiquer les problèmes (et corriger vos tp !) plus facilement. Il s'agit de la touche ESC qui ferme le jeu, la touche F10 qui change l'état plein-écran de la fenêtre de jeu et finalement la touche Y qui permet de passer en mode d'affichage en fil (wireframe) pour mieux voir la géométrie. En aucun temps ces raccourcis et le code qui leur permet de fonctionner ne doivent être enlevés du jeu.

Le jeu final sera en mode *singleplayer*, mais dans notre design nous devons garder en tête qu'il pourrait éventuellement devenir multijoueur, et que dans ce cas il devra être le plus simple possible de le faire sans changer toute la structure de celui-ci.

Objectifs du travail

1. Utiliser le polymorphisme
2. Se famialiariser avec les transformation et le système de coordonnées de OpenGL
3. Utilisation d'un outil de gestion de code source (git)

Règles importantes

1. Chaque classe doit avoir son propre fichier *.h* et *.cpp*
2. N'oubliez pas les *"include guard"*
3. Prenez grand soin de respecter la casse pour les noms de classe, méthode, etc
4. Les règles de l'encapsulation doivent être respectées au maximum, seulement ce qui doit absolument être public peut l'être
5. Le code doit compiler sous visual studios 2012
6. Le programme doit fonctionner sur les ordinateurs de notre local de cours
7. Veuillez commenter votre code *intelligemment*
8. **Il est primordial de soumettre votre code régulièrement sur git, et cet aspect sera évalué pour ce travail ainsi que les subséquents**
9. **Fiez-vous au résultat du démo exécutable qui est fournit avec ce TP**

Avant de commencer...

N'oubliez pas de renommer le projet de base pour qu'il porte le nom de votre jeu.

Un premier cube !

La base de jeu qui est fournie avec ce TP possède déjà un plancher de 200x200 unités de dimension, et situé à la position -2 sur l'axe des Y. Vous devez créer et afficher un cube qui :

1. Est centré à l'origine
2. Est de dimension 1 unité x 1 unité x 1 unité
3. Possède une texture différente que celle du plancher. Le choix de la texture vous appartient (la texture doit être mise dans le même répertoire que celle du plancher)
4. Doit tourner sur lui-même.

Classe Player

La classe Player sert à représenter le joueur du jeu (vous, moi), c'est à dire sa position et son orientation. Cette classe fait donc office de *caméra* (dans ce tp) et ses fonctionnalités seront étendues ultérieurement pour gérer plus de choses (collisions, etc).

Dans notre jeu (comme c'est le cas pour minecraft aussi) chaque cube qui compose le monde est de 1 mètre x 1 mètre x 1 mètre. Pour se faciliter la tâche, nous assumerons que 1 mètre est équivalent à une unité (1.f) dans opengl. Le joueur mesure environ 1.7 mètre de haut et est moins large que 1 mètre (!!). Ces détails seront importants lorsque nous auront à gérer les collisions dans un prochain travail.

La vue du joueur (la caméra) est de type *first-person* comme c'est le cas des jeux FPS (*first-person shooter*). La classe Player est responsable de gérer le déplacement et la rotation du joueur, et d'appliquer ces transformations dans opengl.

Vous devez implémenter les méthodes publiques suivantes dans la classe Player (voir description ci-dessous) :

Listing 1– Méthodes de Player

```
Player(float posX = 0, float posY = 0, float posZ = 0, float rotX = 0, float rotY = 0);
~Player();

void TurnLeftRight(float value);
void TurnTopBottom(float value);

void Move(bool front, bool back, bool left, bool right, float elapsedTime);

void ApplyRotation() const;
void ApplyTranslation() const;
```

NOTE : Le code de ces méthode (et surtout les formules) est fourni à l'adresse suivante. Il y aura quelques modifications mineures à faire pour l'adapter à notre projet et c'est pourquoi vous devez comprendre la logique derrière celles-ci : `http://www.swiftless.com/tutorials/opengl/camera2.html`

Vous devrez créer un objet de type Player dans la classe Engine (m_player), et appeler les méthodes ci-haut en réponse aux évènements clavier et souris. La méthode **TurnLeftRight** permet de gérer le déplacement horizontal de la souris (rotation par rapport à l'axe Y dans opengl) et la méthode **TurnTopBottom** permet de gérer le déplacement vertical de la souris (rotation par rapport à l'axe des X dans opengl). Pour ces deux méthodes le signe de la valeur passée en paramètre indique la direction du mouvement (si négatif c'est un mouvement vers la gauche ou vers le bas, sinon c'est un mouvement vers la droite ou vers le haut). Vous aurez besoin d'utiliser la méthode MakeRelativeToCenter définie dans la classe de base OpenglContext pour ajuster la position de la souris par rapport à la position du centre de l'écran. Après chaque lecture de position, il sera nécessaire de recentrer la souris (CenterMouse).

La méthode **Move** sert à déplacer le joueur, et le déplacement doit se faire selon son orientation courante. Les quatre premiers paramètres indiquent dans quel sens le déplacement doit se faire. Il est possible qu'un déplacement survienne dans plus d'une direction à la fois, si le joueur appuie sur plus d'une touche de direction en même temps. Il est évident que les touches de direction opposées (front et back, left et right)

s'annullent si elles sont appuyées simultanément. Le paramètre elapsedTime sert à ajuster le mouvement selon le temps depuis le dernier appel.

Conseils pour gérer le mouvement : La meilleure façon de faire est de vous garder 4 variables de type bool dans votre classe Engine (metton m_keyW, m_keyA, m_keyS, m_keyD) qui sont initialement initialisées à *false*, et qui serviront à garder l'état enfoncé ou non de chacune des touches de direction WASD. Dans la méthode KeyPressEvent de la classe Engine vous changez l'état de ces variables à *true* lorsqu'elles sont appuyées, et vous changez leur état à *false* lorsque les touches sont relachées dans la méthode KeyReleaseEvent. Il vous suffira d'appeler m_player.Move(m_keyW, m_keyS, m_keyA, m_keyD, elapsedTime) ; au début de la méthode Render de la classe Engine pour que le mouvement se fasse correctement et qu'il tienne compte du elapsedTime reçu en paramètre par Render.

Finalement, les méthodes **ApplyRotation** et **ApplyTranslation** servent respectivement à appliquer les transformation de rotation et de translation avec opengl. La méthode **ApplyRotation** devra faire deux appels successifs à glRotatef, une première fois pour la rotation selon l'axe des X, et une seconde fois pour la rotation selon l'axe des Y. La méthode **ApplyTranslation** devra quant-à elle faire un seul appel à glTranslatef qui fera une translation selon la position du joueur en x, y et z. Ces deux méthodes devront être appelées juste après le LoadIdentity de la méthode Render de la classe Engine :

Listing 2– Appel de ApplyRotation et ApplyTransformation

```
m_player.ApplyRotation();
m_player.ApplyTranslation();
```

Tests

Aucuns tests unitaires ne sont à réaliser pour ce travail, mais vous devez vous assurer que votre jeu fait la même chose (au minimum) que la démo fournit avec cet énoncé.

Remise

A remettre sur Vortex :

1. Un fichier texte (.txt) contenant le hash `sha1` (exemple : `8fea0e32a53c59eac95c157fa060e112cf88b7a0`) du dernier commit que vous voulez que je corrige. Assurez-vous d'avoir fait un `push` sur le serveur pour que je puisse voir votre dernière version. **Ne pas remettre votre projet directement sur vortex !**