Compte rendu : TP2 – MOTEUR DE JEUX

Vincent de Rousiers

Question 1 :

Actions des touches :

- A : Rotation verticale vers le haut
- Z : Zoom avant
- E : Rotation verticale vers le bas
- Q : Rotation horizontale vers la gauche
- S : Zoom arrière
- D : Rotation horizontale vers la droite
- W : Modifier le mode de visualisation (Point, triangle etc...)
- X : Changer la map à visualiser

Paramètres :

- etat : sert à stocker le mode d'affichage de la map (etat à 0 map représentée par des points, etat à 1 map représentée par des lignes etc.)
- ss : sert à stocker le zoom (le multiplicateur de la matrice)
- rotX : stocke l'angle représentant une rotation à effectuer sur la matrice, cet angle associé à un vecteur (1,0,0) va donc réaliser une rotation sur l'axe vertical
- rotY : tout comme rotX, mais associé à un vecteur (0,0,1) afin d'effectuer la rotation sur l'axe horizontal

Question 2 :

Création de deux attributs, *fps* et *timer*, dans la classe **gamewindow** afin de stocker la fréquence et l'état du timer, puis ajout d'un constructeur prenant en paramètre un int pour initialiser *fps*.
Ce constructeur se contente d'affecter le int à *fps* et de créer et démarrer un nouveau timer basé sur l'attribut précédent.

```
GameWindow::GameWindow(int i){
    fps = i;
    timer = new QTimer(this);
    timer->connect(timer, SIGNAL(timeout()), this, SLOT(renderNow()));
    timer->start(1000 / this->getFps());
}
```

Rajout également d'un getter, getFps(), qui permet de récuperer la valeur de l'attribut *fps*.

Question 3 :

Création dans le main de 4 instances **gamewindow** différentes (window1, window30, window60 et window120) respectivement à 1, 30, 60 et 120 fps.
Ajout dans la classe **gamewindow** d'un setter setCam() qui permet d'attribuer une nouvelle caméra à l'instance **gamewindow**.

Dans le main :

```
Camera *c = new Camera();

window1.setCam(c);
window30.setCam(c);
window60.setCam(c);
window120.setCam(c);
```

Les 4 instances possédent ainsi toutes la même caméra. Toutes les actions se basant sur la caméra seront dont réalisées dans les 4 fenêtres (à savoir les rotations, les zooms et les changements d'états).
Seuls le changement de fréquence et le changement de map restent spécifiques à chaque fenêtre.

Question 4 :

Ajout d'un attribut *tourniquet* dans la classe **gamewindow**, il s'agit d'un boolean représentant l'état de la visualisation, qui tourne ou non sur l'axe Y.
On prend donc ce nouvel attribut en compte dans la fonction keyPressEvent :

```
void GameWindow::keyPressEvent(QKeyEvent *event)
{
...
      case 'C':
        tourniquet = !tourniquet;
        break;
...
}
```

Si on tourne et qu'on presse 'C', on arrête de tourner et inversement.

Et dans le render, si on tournicote :

```
      if(tourniquet){
        cam.setRotY(cam.getRotY() + rotation);
      }
```

Le fait de tourner n'empêche pas l'utilisateur de zoomer.
rotation est un attribut de **gamewindow** (tout comme zoom) qui permet de stocker les pas de rotation ou de zoom.

Question 5 :

Tout comme dans la question 4, on modifie la fonction keyPressEvent :

```
void GameWindow::keyPressEvent(QKeyEvent *event)
{
...
case 'P':
        fps *= 2;
        if(fps < 240){
            fps = 240;//limitation
        }
```

```
            timer->setInterval(1000 / fps);
            break;
        case 'M':
            fps /= 2;
            if(fps < 1){
                fps = 1;//limitation
            }
            timer->setInterval(1000 / fps);
            break;

...
}
```

Avec une limitation de la fréquence à la hausse et à la baisse ainsi que l'actualisation du timer. Cette limitation est nécessaire car le programme ne pourra pas rendre un affichage supérieur à un certain nombre de fps, ni inférieur. De plus, l'utilisateur ne verra pas la différence une fois que les fps auront atteint une valeur suffisamment élevée.

Bonus :

gamewindow hérite de **openglwindow** car celle-ci hérite de **QWindow**, or **gamewindow** a pour but d'afficher une carte de hauteur dans une fenêtre. De plus **gamewindow** utilise des fonctions opengl et **openglwindow** hérite aussi de **QopenGLFunctions**.
gamewindow nécessite donc les deux héritages de **openglwindow**.

L'avantage de procéder à cet héritage est de stocker toutes les fonctions de gestion de la fenêtre dans une classe et tout ce qui concerne la carte dans une autre. Comme pour la création de la caméra, on compartimente le code, ce qui le rend plus lisible et plus simple à modifier.

L'inconvénient est de devoir vérifer dans la classe parente les redéfinitions de fonctions et de s'assurer qu'elles correspondent à l'usage que l'on souhaite.

On pourrait éviter cet héritage en faisant directement hériter **QWindow** à **gamewindow**.