Mini-projet Qt/C++

Ce TP a pour but de vous lancer sur la réalisation d'un mini-projet en Qt/C++.

Cahier des charges minimales :

- Le joueur contrôle le personnage au clavier
- Le meilleur score est mémorisé (temps d'arrivée)
- Le choix de l'esthétique et des règles du jeu est libre
- On doit pouvoir recommencer une partie lorsque le joueur a perdu sans devoir relancer le programme

Chaque code est personnel. Si du code est utilisé à plusieurs, merci de l'indiquer dans vos sources. Si du code provient de l'extérieur, merci d'en indiquer la source en commentaire dans le code. Dans le cas contraire, c'est du plagiat et les codes plagiés seront sanctionnés.

Rendu du code :

- **Dimanche 15 mai, 20h00**. Attention, tout retard sera sévèrement sanctionné par des points en moins.
- Une archive à déposer sur Moodle de votre code portant votre nom de famille en minuscules et sans espace contenant :
  - Les fichiers .h et .cpp
  - Le fichier de compilation (CMakLists.txt, Makefile, fichier .pro…)
  - Les répertoires contenant les différentes sources utiles au projet (images, son…)

Une attention particulière va être portée sur le rendu. Notamment le bon déroulement de la compilation de votre code source et de sa première exécution, ainsi que la présence de commentaires dans le code source.

Le code peut être écrit sur la plateforme et avec l'EDI de votre choix (CLion, QtCreator…).

1. Pour bien démarrer

1.1 Récupérer l'archive CPP_QT_TPminiprojet.zip sur Moodle.
1.2 Créer un nouveau projet C++ sur QtCreator.
1.3 Copier/coller le code contenu dans l'archive dans votre projet.
1.4 Compilez et exécutez le code source. **Prenez soin de vous approprier le code source et de poser des questions si vous en avez**, il représente la base de votre application !

2. Ajout d'objets à la scène

La scène représente la classe qui contient tous les éléments du jeu. Evidemment, plus le code est conséquent, plus cette classe doit être décomposée en sous-classes, chacune s'occupant de taches particulières à la scène. Pour ce projet, vous pouvez rester concentrés sur cette classe et ne pas nécessairement vous préoccuper de créer d'autres classes.

Les objets ajoutés à la scène héritent tous d'une classe QGraphicsItem. Cette classe est la classe mère de classes filles spécialisées : QGraphicsTextItem, QGraphicsRectItem, QGraphicsPixmapItem…

Commencez par tester l'ajout d'un objet à votre scène : dans le constructeur de MyScene, ajouter :

```
QGraphicsRectItem* qgri = new QGraphicsRectItem(10, 100, 300, 200);
this->addItem(qgri);
```

Observer le résultat.

Ajouter maintenant :

```
QGraphicsTextItem* qgti = new QGraphicsTextItem("CIR2 Rennes");
this->addItem(qgti);
```

Vous aurez ensuite besoin d'ajouter des QGraphicsPixmapItem qui sont des objets contenant une image. Vous pourrez essayer à la fin de la séance de TP s'il vous reste du temps.

3. Animation de la scène

Pour l'animation, il suffit d'ajouter un timer à la scène. Créer un attribut timer dans MyScene puis initialisez-le dans le constructeur :

```
timer = new QTimer(this);
connect(timer, SIGNAL(timeout()), this, SLOT(update()));
timer->start(30); //toutes les 30 millisecondes
```

Créer ensuite un slot dans votre classe qui est la méthode update(). Dans cette méthode, essayez de faire tout d'abord un affichage simple (avec un cout ou un qDebug), puis de bouger l'objet qgti défini précédemment en le faisant descendre vers l'objet qgri.

Pour cela, vous pouvez utiliser le code suivant dans la méthode update() :

```
QPointF pos = qgti->pos(); //récupération de la position de l'objet qgti
qgti->setPos(pos.rx(), pos.ry()+1); //incrémentation de la coordonnée y
```

4. Gestion de la collision

Pour pouvoir réaliser le jeu demander, une question qui va vite se poser est de savoir quand un objet percute un autre objet ? Pour cela, Qt vous fournit un mécanisme de gestion des collisions sur les objets de type QGraphicsItem.

Dans la méthode update(), ajouter la gestion de la collision :

```
if (qgti->collidesWithItem(qgri)) {
   qDebug() << "Collision !";
}
```

Le message de collision doit s'afficher lorsque les 2 items entrent en collision. Attention, le calcul est réalisé sur les "bounding box" (boites englobantes) associées à chaque objet. Cette bounding box correspond à un rectangle qui englobe tout l'objet.

Les animations et les déplacements des items doivent se réaliser dans la méthode update() de la classe MyScene (ou dans une méthode appelée dans la méthode update). Cette méthode update() est appelée toutes les X millisecondes et permet le rafraîchissement du jeu. C'est ce qui va déterminer notamment la vitesse de rafraîchissement (les FPS pour les gamers...). Il faut donc que les animations soient gérées dans cette méthode.

5. Gestion des touches clavier

Un dernier point à aborder est la gestion des touches au clavier. Dans la classe MyScene :

- définir dans le .h void keyPressEvent(QKeyEvent* event); et éventuellement void keyReleaseEvent(QKeyEvent* event); en protected (surcharge)
- les écrire dans le .cpp :

```
void MaClasse::keyPressEvent(QKeyEvent* event){
     if(event->key() == Qt::Key_P) { // appui sur la touche P du clavier
     ...
     }
}
```

Inutile de faire un connect ou de gérer les signaux, les évènements claviers/souris sont automatiquement gérés dans la classe QGraphicsScene.

Pour mettre le jeu en pause, il suffit d'arrêter le timer.

6. Développement du jeu…

Vous avez maintenant tous les éléments en votre possession pour développer le jeu.

Concentrez-vous sur les taches principales. Il est essentiel de **savoir hiérarchiser** les problèmes. Par exemple, si vous n'arrivez pas à mettre une image en fond, ce n'est pas essentiel au bon fonctionnement de votre programme. Vous reviendrez dessus plus tard, vous allez monter au fur et à mesure en compétences sur Qt, des choses vont s'éclaircir au fur et à mesure du temps passer à développer.

Ne mélangez pas tout, au risque de créer du "code spaghetti" : **découpez votre code en méthodes**, chaque méthode réalisant une tâche simple et claire.

Testez et débuggez votre code le plus souvent possible. Evitez de reproduire sans cesse les mêmes tests, au risque de passer à coté de certaines erreurs.

Aide : redimensionnement de la fenêtre et systèmes de coordonnées

Il existe plusieurs façons de faire (comme très souvent). Vous remarquerez qui si vous utilisez une petite image en fond de votre scène, elle va se répéter. Voici une solution "propre" avec en fond une image (png, jpg...).

La méthode consiste à :

- Réécrire la méthode drawBackground de votre classe héritant de QGraphicsScene pour dessiner une seule fois l'image dans la scène.
- Réécrire la méthode resizeEvent de la classe s'occupant de la vue (hérite de QGraphicsView) pour redéfinir l'affichage lors du redimensionnement.

Concrètement cela donne :

```
void MyScene::drawBackground(QPainter* painter, const QRectF &rect) {
    Q_UNUSED(rect);
    painter->drawPixmap(QPointF(0,0), pixBackground, sceneRect());
   // pixBackgroud est un attribut de type QPixmap qui contient l'image de
fond
}
```

Il faut ensuite redéfinir une classe MyView héritant de QGraphicsView et surcharger la méthode resizeEvent :

```
protected:
    virtual void resizeEvent (QResizeEvent* event) {
         this->fitInView(sceneRect());
    }
```

Normalement, les redimensionnements de votre fenêtre principale donneront un comportement "normal".

A partir du moment où vous faites cela, toutes les coordonnées utilisées vont dépendre des coordonnées de la scène : il faut donc tout développer en fonction de la taille de l'image que vous allez positionner au fond. N'hésitez pas à la redimensionner une bonne fois pour toute.