TP8 : Pong

utilisation de la bibliothèque Qt5

1. Objectif

L'objectif de ce TP est d'utiliser seulement la bibliothèque Qt5 pour créer un jeu de Pong (cf Figure 1), le tout premier jeu vidéo commercialisé et édité par Atari.

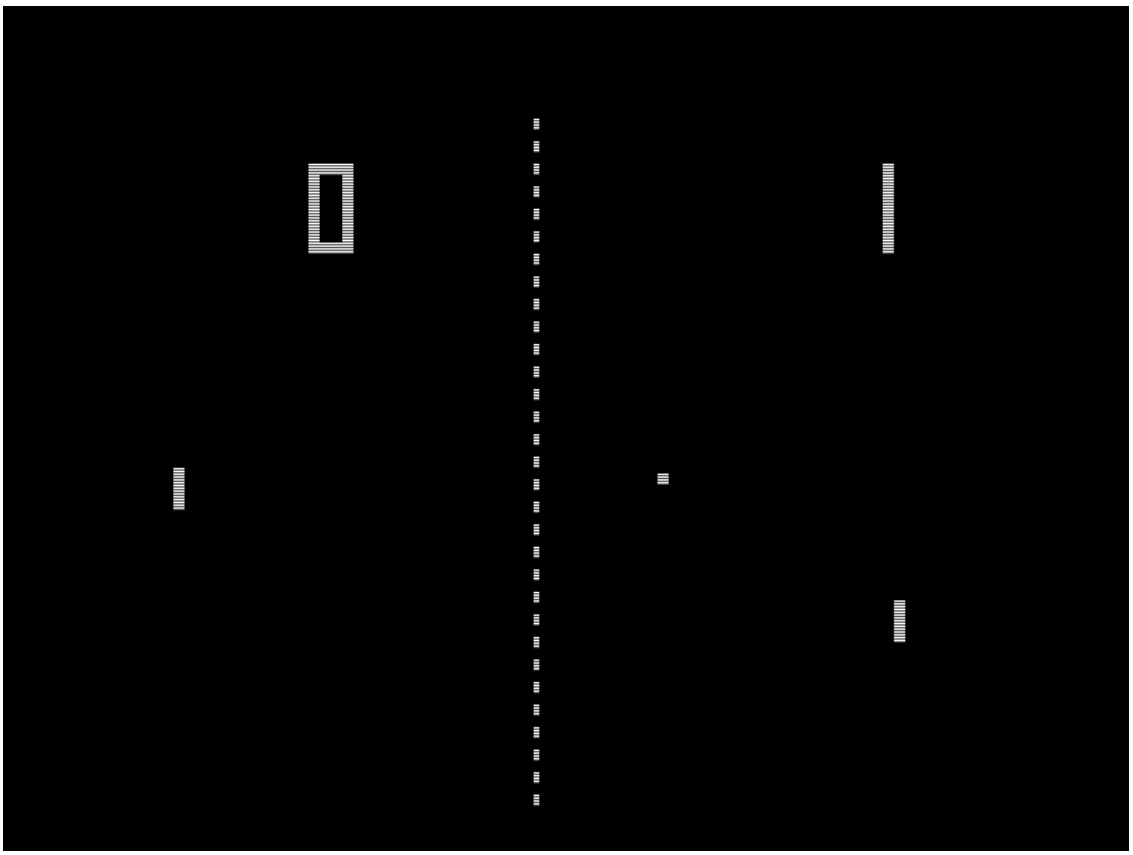

Figure 1 : le premier jeu de Pong : deux joueurs de chaque côté contrôlant une palette ayant des mouvements verticaux, une balle, et de longues heures de jeu !

Le cahier des charges minimal à respecter est le suivant :

- Chaque joueur contrôle sa palette avec 2 touches du clavier
- La balle rebondit sur la palette et est envoyée chez l'adversaire
- Si l'adversaire ne la rattrape pas, le score du joueur s'incrémente
- Le jeu se termine lorsqu'un score est atteint par l'un des joueurs

Votre jeu devra ensuite contenir des améliorations personnelles. Par exemple, on pourra avoir une augmentation de la vitesse de la balle au fur et à mesure des coups, plusieurs

niveaux de jeu (présence d'obstacles), l'affichage du jeu en plein écran, la possibilité de changer l'image de fond, des bruitages, de la musique… Les exercices de ce TP consistent à vous faire partir sur de « bonnes » bases en Qt. Libre à vous de choisir une autre stratégie si vous connaissez déjà la bibliothèque Qt.

2. Création d'une interface

Commencez par créer un nouveau projet Qt. Créez une classe « FenetrePrincipale » héritant de la classe QMainWindow. Cette classe représente l'interface graphique de votre application.

```
#ifndef FENETREPRINCIPALE_H
#define FENETREPRINCIPALE_H
#include <QtWidgets>

class FenetrePrincipale : public QMainWindow {
	Q_OBJECT
	public:
		FenetrePrincipale();
		~FenetrePrincipale();
		QGroupBox * BuildGroupBoxControle();

	public slots:

	private:
		QWidget * widget_general;
		QMenu * menuFichier;
};

#endif
```

```
#include "fenetrePrincipale.h"

FenetrePrincipale::FenetrePrincipale() : QMainWindow() {
	// le widget dans lequel tout s'affiche
	widget_general = new QWidget;
	QHBoxLayout * qbl_general = new QHBoxLayout;
	widget_general->setLayout(qbl_general);
	this->setCentralWidget(widget_general);

	//exemple de creation d'une zone ou mettre boutons...
	qbl_general->addWidget(BuildGroupBoxControle());

	// exemple de création d'un menu
	menuFichier = menuBar()->addMenu(tr("&Fichier"));
}

FenetrePrincipale::~FenetrePrincipale() {
}

QGroupBox * FenetrePrincipale::BuildGroupBoxControle() {
	QGroupBox * qgb = new QGroupBox(tr("Contrôle"));
	return qgb;
}
```

3. Création d'une classe pour l'affichage de l'animation

Qt dispose d'un mécanisme efficace basé sur le modèle MVC (Model – View – Control) pour l'affichage. L'idée principale est de disposer d'une part d'une vue s'occupant de l'affichage et d'autre part d'une scène s'occupant de ce qu'il y a concrètement dans la vue. Ce mécanisme est très pratique pour répondre à la question suivante : comment se repérer dans une scène définie dans un système de coordonnées personnalisé d'une vue (ce qui est affiché sur l'écran) définie justement par les coordonnées de votre écran ?

L'animation se passe donc dans une fenêtre de type QGraphicsScene. On crée pour cela une classe MyScene :

```
#ifndef MYSCENE_H
#define MYSCENE_H
#include <QtWidgets>

class MyScene : public QGraphicsScene {
	Q_OBJECT
public :
	MyScene(QObject *parent = 0);
};
#endif
```

```
#include "MyScene.h"
MyScene::MyScene(QObject *parent) : QGraphicsScene(parent)
{
}
```

Instanciez cet objet dans votre FenetrePrincipale, puis affichez votre scène dans une « vue » QGraphicsView :

```
QGraphicsView * myview;
```

```
myscene = new MyScene(this);
myview = new QGraphicsView(myscene, this);
qbl_general->addWidget(myview);
```

Testez et essayez de changer la couleur de fond de la scène.

4. Ajouter un objet à la scène

Nous allons maintenant ajouter des objets à la scène. Ces objets héritent de la classe QGraphicsItem : il a des QGraphicsTextItem, QGraphicsRectItem, QGraphicsPixmapItem…

- Commencez par tester brutalement l'ajout d'un objet à votre scène : dans le constructeur de MyScene, ajoutez ceci :

```
QGraphicsRectItem * qgri = new QGraphicsRectItem(10, 10, 300, 300);
```

```
this->addItem(qgri) ;
```

- Essayer de créer de la même façon un QGraphicsPixmapItem que vous ajoutez à la scène. C'est un objet défini par une image.
- Essayez de créer un QGraphicsTextItem et ajoutez le à votre scène.

5. Animez la scène

Ajoutez un timer à votre scène. Créez un attribut timer dans votre classe MyScene puis initialisez-le dans le constructeur :

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

Redéfinissez enfin la méthode update() dans votre classe MyScene et essayez de faire bouger votre QGraphicsPixmapItem à l'aide de la méthode setPos(int, int).

6. Poursuivez le développement du jeu et quelques conseils

Une fois ces étapes réalisées, à vous de jouer. Faites en sortes de pouvoir bouger les items à l'aide du clavier, d'animer la balle, de gérer les rebonds, le score…

- Prenez le temps de bien tester votre code à chaque nouvelle étape. Même si cela prend du temps, mieux vaut tester à chaque nouvelle ligne que l'on écrit !
- Utilisez le debugger et/ou des affichages lors de l'exécution de votre application. N'hésitez pas à faire un affichage exhaustif des valeurs des variables utilisées.
- Utilisez Internet et Google « intelligemment » ! Vous irez bien plus vite dans le développement en recopiant des petits bouts de code et en les testant, plutôt que de reprendre un code existant que vous n'avez pas écrit.
- La documentation Qt est très bien écrite, servez-vous-en !

7. Aide : n'afficher qu'une et une seule image en fond de la scène et gérer son redimensionnement

Plusieurs solutions sont possibles (comme très souvent en Qt). La méthode présentée ici consiste à surcharger la méthode drawBackground de QGraphicsScene pour dessiner une seule fois l'image dans la scène, puis de redéfinir une class MyView héritant de QGraphicsView pour redéfinir l'affichage lors du redimensionnement.

Créez un attribut pixbackground de type QPixmap dans votre classe MyScene. Charger l'image correspondante dans le constructeur : pixbackground.load("monimage.jpg"). Redéfinissez ensuite la méthode drawBackground dans la classe MyScene. Cette méthode est appelée à chaque fois que la scène est dessinée. Ajoutez ensuite les lignes suivantes :

```
void MyScene::drawBackground(QPainter *painter, const QRectF &rect) {
       Q_UNUSED(rect);
       painter->drawPixmap(QPointF(0,0), pixbackground, sceneRect());
}
```

Observez le résultat : votre image est dessinée qu'une seule fois et n'est plus répétée. Nous venons de redéfinir une méthode existante (drawBackground) en modifiant son comportement par défaut. Nous avons donc utilisé une des propriétés du polymorphisme propre aux langages-objets.

Ensuite, pour que l'image s'adapte à la taille de votre vue, il faut adopter la même stratégie : redéfinir la méthode resizeEvent de la classe QGraphicsView. On crée donc une nouvelle classe héritant de QGraphicsView et on redéfinit la méthode en question. Voici le code complet, où tout est défini dans le .h :

```
#ifndef MYVIEW_H
#define MYVIEW_H
#include <QtGui>

class MyView : public QGraphicsView {
       Q_OBJECT

public :
       MyView(){}
       ~MyView(){}

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

Observez le résultat.