Joystick Widget

Ce document a pour but de décrire le fonctionnement et les méthodes d'utilisation du widget.

L'objectif a été d'avoir un joystick sous forme de widget, avec un comportement aussi réaliste que possible. Les principales règles pour ce mimétisme ont été les suivantes :

1) Devoir attraper la partie mobile du joystick pour pouvoir la déplacer.

2) La partie mobile du joystick ne doit pas pouvoir sortir d'un cercle défini.

3) Si le curseur de l'utilisateur sort du cercle défini, la partie mobile doit être positionnée à sa distance maximale, et toujours suivre le curseur.

Ensuite, l'information envoyée par le joystick est déterminée en fonction de la position de la partie mobile.

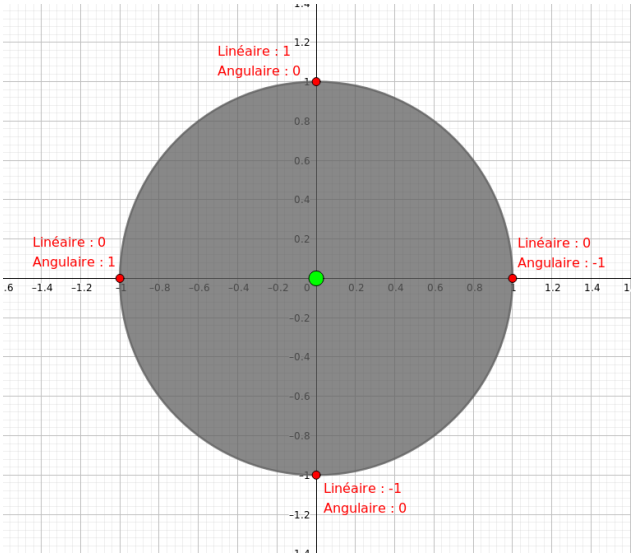

Illustration 1: Graphique de comportement du joystick

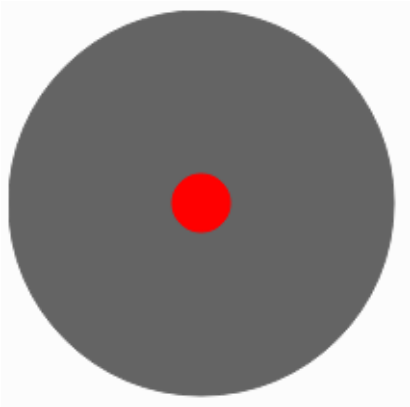

Illustration 2: Joystick

Comme nous pouvons le voir sur l'illustration ci-dessus, la commande linéaire est déterminée en fonction de la position sur l'axe Y de la partie mobile (sur cette illustration en vert), et la commande angulaire est déterminée en fonction de la position sur l'axe X de la partie mobile.

Le joystick est un widget personnalisé du type JoystickWidget qui hérite de la classe QWidget. Il s'agit de deux cercles, le gris correspond à la surface d'utilisation du joystick, et le rouge est l'élément mobile. Ces cercle sont redessinés à chaque nouvel événement lancé par la souris. La taille idéale du widget est de 150x150.

Commençons par énoncer les différents attributs, méthodes et signaux du joystick :

```
#ifndef JOYSTICKWIDGET_H
#define JOYSTICKWIDGET_H

#include <QWidget>

class JoystickWidget : public QWidget
{
  Q_OBJECT

public:
  JoystickWidget( QWidget *parent=0 );
  ~JoystickWidget(){}
  QSize sizeHint() const;

  void reset();
  double getYBase();
  double getY();
  double getXBase();
  double getX();
  void setY(double y);
  void setX(double x);


Q_SIGNALS : void hasMoved();


protected:
  void paintEvent( QPaintEvent* );

  void mouseMoveEvent( QMouseEvent* );
  void mouseReleaseEvent( QMouseEvent* );

private:
  double x, y, r, xBase, yBase, rBase;
  QColor color, colorBase;
  bool isMoving;

};

#endif // JOYSTICKWIDGET_H
```

Illustration 3: Header du joystick

Premièrement, nous avons un constructeur et un destructeur. Ensuite, la méthode sizeHint() renvoie une taille idéale pour le widget.

La méthode reset() replace le joystick dans son état de départ.

S'en suivent une série de getters et de setters traditionnels pour les attributs. Les attributs x et y définissent la position de la partie mobile du joystick (rond rouge sur l'illustration 1). Ensuite, les attributs xBase et yBase déterminent le point 0, le centre du joystick. Ensuite, rBase défini le rayon du cercle dans lequel la partie mobile peut bouger (rond gris sur l'illustration 1).

Les attributs color et colorBase, sont respectivement la couleur de la partie mobile et la couleur du cercle dans lequel la partie mobile peut bouger.

Enfin le booléen isMoving indique simplement si la partie mobile est en train de bouger. Ce booléen est lié au signal hasMoved(). Ce signal est envoyé chaque fois que la partie mobile du joystick bouge.

Trois événements sont ensuite utiles pour faire fonctionner le joystick en temps réel. Ce widget étant contrôlé par la souris, il est logique de voir apparaître un mouseMoveEvent, qui est lancé chaque fois qu'un bouton de la souris est cliqué ou que la souris est bougée, puis un mouseReleaseEvent, qui est lancé chaque fois qu'un bouton de la souris est relâché. Finalement, un paintEvent permet de redessiner le widget à chaque nouveau changement.

Voyons maintenant le code de chacune de ces fonctions :

Le constructeur est très simple, il initialise simplement les attributs, pour que le joystick soit au centre du widget. Il initialise également les couleurs. Le booléen isMoving est initialisé à false, pour indiquer que le joystick ne bouge pas.

```
JoystickWidget::JoystickWidget( QWidget *parent ) : QWidget( parent )
{
    rBase = 100;
    xBase = rBase;
    yBase = rBase;
    colorBase = QColor(100,100,100);

    r = 15;
    x=xBase;
    y=yBase;
    color = QColor( 255, 0, 0 );

    isMoving = false;
}
```

Illustration 4: Constructeur du joystick

La fonction reset() sert donc à replacer le joystick dans son état de départ. Pour cela, elle replace la partie mobile au centre du widget, met à jour le dessin, puis elle passe le booléen isMoving à false pour indiquer que le joystick ne bouge plus.

```
void JoystickWidget::reset()
{
    x=xBase;
    y=yBase;
    update();
    isMoving = false;
}
```

Illustration 5: Fonction reset du joystick

Le premier événement est le paintEvent. Son rôle est simplement de dessiner le joystick. Il dessine des ellipses avec deux rayons similaires pour faire des cercles. Pour forcer la mise à jour du dessin, il faut appeler la fonction update() . Un appel à cette fonction doit être fait après chaque mouvement du joystick.

```
void JoystickWidget::paintEvent( QPaintEvent* )
{
    QPainter painter( this );

    painter.setRenderHint( QPainter::Antialiasing );
    if(this->isEnabled())
    {
        painter.setPen( colorBase );
        painter.setBrush( colorBase );
        painter.drawEllipse( 0, 0, 2*rBase, 2*rBase );

        painter.setPen( color );
        painter.setBrush( color );
        painter.drawEllipse( x-r, y-r, 2*r, 2*r );
    }
    else
    {
        painter.setPen( QColor(175,175,175) );
        painter.setBrush( QColor(175,175,175) );
        painter.drawEllipse( 0, 0, 2*rBase, 2*rBase );

        painter.setPen( QColor(255,100,100) );
        painter.setBrush( QColor(255,100,100) );
        painter.drawEllipse( x-r, y-r, 2*r, 2*r );
    }

}
```

Illustration 6: Fonction de dessin du joystick

```
void JoystickWidget::mouseMoveEvent( QMouseEvent *e )
{
    int dist2 =(xBase-e->x())*(xBase-e->x()) + (yBase-e->y())*(yBase-e->y());
    int distRedDot2 = (x-e->x())*(x-e->x()) + (y-e->y())*(y-e->y());

    if( distRedDot2 < (r*r) )
    {
        isMoving = true;
    }

    if( isMoving )
    {
        if( dist2 > (rBase*rBase) )
        {
            float ratio = rBase / sqrt(dist2);    // ------
            x = xBase + ( e->x() - xBase)*ratio; // Thalès
            y = yBase + ( e->y() - yBase)*ratio; // ------
        }
        else
        {
            x = e->x();
            y = e->y();
        }
        Q_EMIT JoystickWidget::hasMoved();
    }
    update();
}
```

Illustration 7: Événement du mouvement de la souris

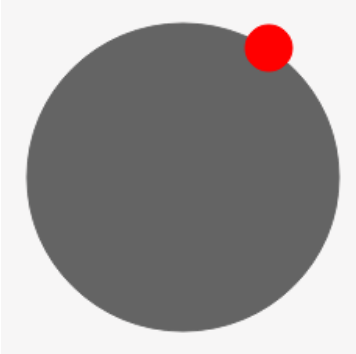

Illustration 8: Partie mobile du joystick à la limite

Le mouseMoveEvent est lui plus complexe. En premier deux distances sont calculées, elles sont exprimées à la puissance deux pour éviter un traitement via une racine carrée. La première est la distance entre le curseur de la souris et le centre du widget et la seconde est la distance entre le centre de la partie mobile et le curseur de la souris.

Ensuite la partie « intelligente » du joystick est mise en place. Rappelons que cet événement n'aura lieu que si un bouton de la souris est cliqué et que la souris est déplacée.

Premier test : si la distance au carré entre le curseur et le centre de la partie mobile est inférieure au rayon au carré de la partie mobile, alors cela signifie que l'utilisateur vient de bouger la partie mobile. On passe donc le booléen isMoving à true pour indiquer un mouvement du joystick.

Deuxième test : si le joystick est en mouvement, alors un nouveau test se fait. Si la distance au carré entre le curseur et le centre du widget est supérieur au rayon au carré du cercle, alors cela signifie que le curseur est au-delà de la limite. Dans ce cas, on ramène via un simple théorème de Thalès, la partie mobile du joystick à la limite. Sinon, si ce test n'est pas vrai, alors cela signifie que la partie mobile du joystick est à l'intérieur du cercle, donc on la place au même endroit que le curseur. Finalement, on envoie le signal que le joystick a bougé, puis on met à jour le dessin.

S'en suit logiquement l'événement mouseReleaseEvent, qui intervient quand un bouton de la souris est relâché. Cet événement est très simple, il appelle simplement la fonctions reset() vue plus haut (Illustration 9), puis envoie le signal que le joystick a bougé.

```
void JoystickWidget::mouseReleaseEvent( QMouseEvent *e )
{
    reset();
    Q_EMIT JoystickWidget::hasMoved();

}
```

Illustration 9: Événement du clique de la souris relâché

Pour ce qui est de l'implémentation dans une application, il suffit d'instancier un objet de type JoystickWidget, puis de connecter le joystick et le signal hasMoved() à un slot qui enverra la commande désirée.

Voici un exemple de commande :

```
void gui_leenby::MainWindow::joystickCallback()
{
    // Les coefficents 100 peuvent et doivent être modifiés en fonction de l'utilisation
    qnode.changerCommande(
                (( joystick->getYBase()-joystick->getY() )/100.0) * VIT_LIN_MAX ,
                0,
                0,
                0,
                0,
                 ((joystick->getXBase()-joystick->getX())/100.0) * VIT_ROT_MAX );
}
```

Illustration 10: Exemple de slot