CLIN Matthieu
VECCHIO Quentin

Groupe 3

Université de Metz
DUT INFO
2ème Année

Rapport de projet

C++

Logiciel de dessin assisté

Année 2013-2014

SOMMAIRE

I. Introduction

Dans le cadre du projet UML, nous avons modélisé, avec différents sortes de diagrammes, des objets géométriques à deux dimensions. Ce projet nous a permis d'analyser les différents objets et cas d'utilisation afin de pourvoir développer le logiciel de dessin assisté de façon optimisée.

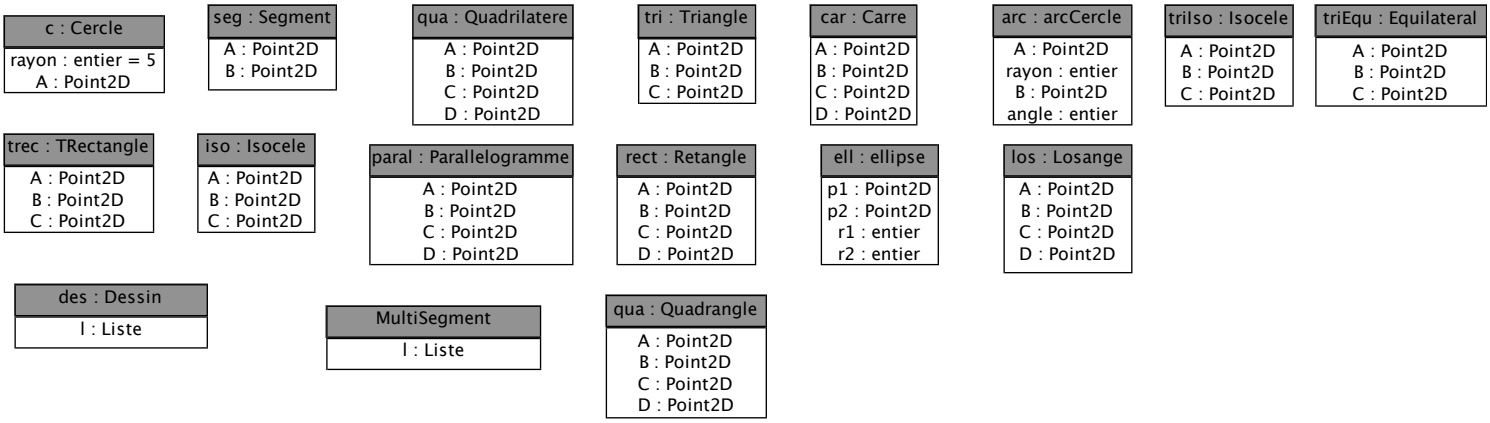
Le rôle du logiciel de dessin assisté est de pouvoir modéliser plusieurs objets mathématique tel que le segment ou le carré mais également des objets plus complexe tels que le multi segment ou le dessin technique.

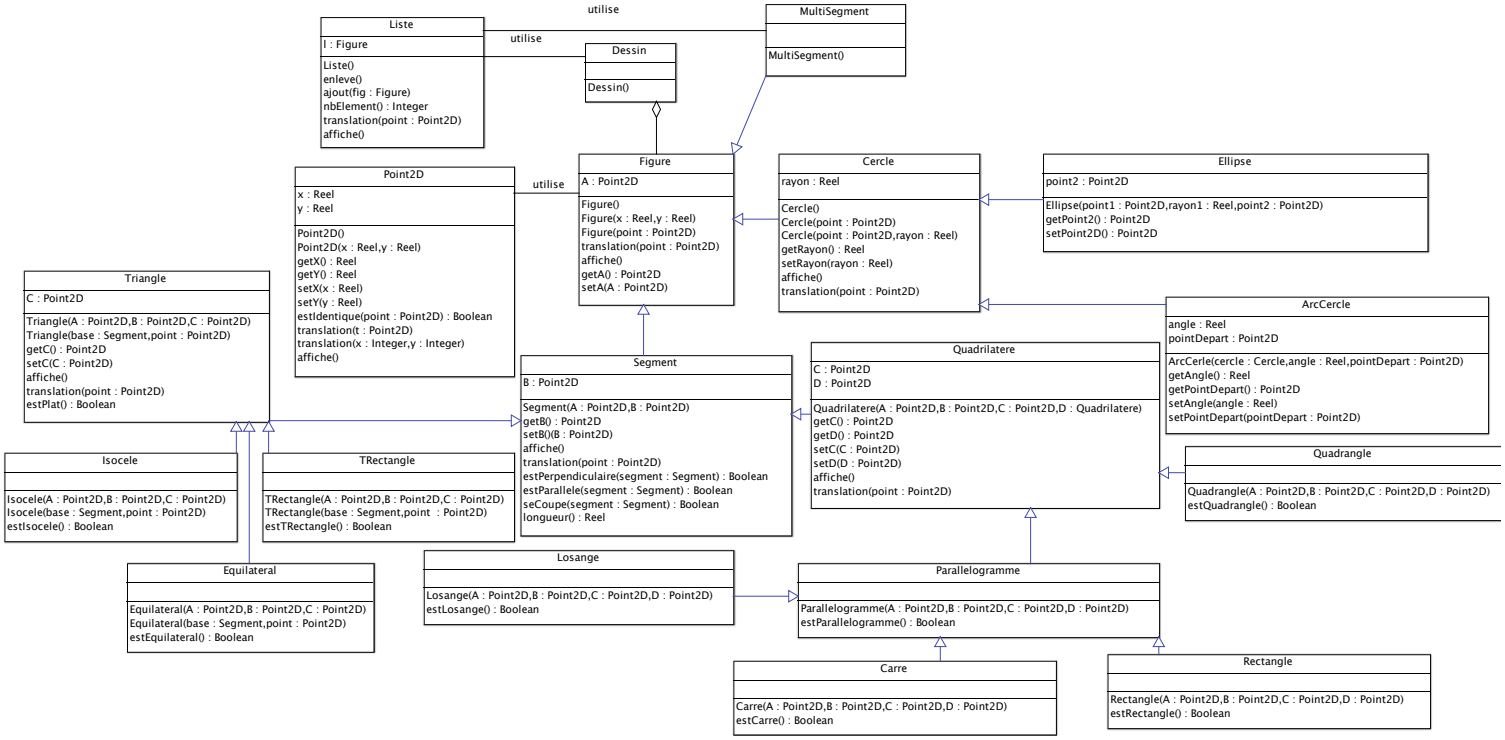
Le logiciel doit également permettre la modification et la suppression de ces objets, ainsi que la gestion de fonctions mathématique comme la translation ou la rotation.

Dans ce rapport, nous tâcherons de vous expliquer les codes sources de chaque objets, ainsi que nos choix de développement. Nous passerons également par une analyse du graphique afin de justifier nos choix concernant le développent d'une GUI. Un mode d'emploi de l'application est également fourni avec ce rapport, il vous en apprendra plus sur la prise en main du logiciel.

II. Rappels

Afin de bien comprendre la partie concernant les classes développé en C++, il est primordiale de se remémorer le diagramme de classe et le diagramme d'objets fait durant le projet d'UML.
En effet, nous sommes parties de ces diagrammes afin de développer toutes nos classes. Ce diagramme nous a servis de base pour pour définir les différents héritages entre les classes, mais aussi les attributs et méthode de chaque unes.





III. Classes

Dans cette parties, nous décrivons brièvement chaque classes et nous expliquons nos choix de programmation. Vous trouverez tout le code source dans l'annexe de ce rapport.

A) Point2D

Cette classe est utilisée dans toutes les autres classes. En effet, chaque figure est constitué d'un ou plusieurs points. Elle est constitué de deux attributs, X et Y. Nous avons fait le choix de les définir en type float car certaines figures n'auraient pus être modélisé, comme le cas du triangle équilatéral qui ne peux être construit avec des coordonnées entières.

B) Figure

Cette classe est la classe mère. En effet nous avons choisit de faire hériter toutes nos classe de la classe Figure pour permettre un développement de la classe Dessin Technique plus simple. Cette classe est constituée d'un seul attribut un Point2D. Nous avons également implémenté des méthodes qui ont été redéfinis dans les classes fille. Cette classe est une classe abstraite.

C) Segment

Cette classe hérite de Figure. Elle a un attribut en plus, un autre point. Différentes méthodes permettent d'interagir avec, comme la méthode qui calcule le milieu du segment.

D) Quadrilatere

Cette classe hérite de Segment. Elle a deux attributs en plus, deux autres points. Cette classe est la base de toutes les figures étant constitué de quatre points.

E) Quadrangle

Cette classe hérite de Quadrilatère. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Quadrangle.

F) Parallelogramme

Cette classe hérite de Quadrilatère. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Parallélogramme. Cette classe sert de classe mère au carré, rectangle et losange.

G) Losange

Cette classe hérite de Parallélogramme. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Losange.

H) Carre

Cette classe hérite de Parallélogramme. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Carré.

I) Rectangle

Cette classe hérite de Parallélogramme. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Rectangle.

J) Triangle

Cette classe hérite de Segment. Elle a un attribut en plus, un autres point. Cette classe est la base de toutes les figures étant constitué de trois points.

K) Isocele

Cette classe hérite de Triangle. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Triangle isocèle.

L) Equilateral

Cette classe hérite de Triangle. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Triangle équilatéral.

M) Triangle Rectangle

Cette classe hérite de Triangle. Elle n'est constitué que d'une méthode en plus, la méthode qui permet de savoir si la figure est bien un Triangle rectangle.

N) Cercle

Cette classe hérite de Figure. Elle a un attribut en plus, un rayon . Cette classe est la base de toutes les figures étant constitué de rayon.

O) Ellipse

Cette classe hérite de Cercle. Elle a un attribut en plus, un second point.

P) ArcCercle

Cette classe hérite de Cercle. Elle a un attribut en plus, un angle.

Q) MultiSegment

Nos choix de programmation nous ont amené à faire de la classe MultiSegment un vector. Un vector est un tableau déclaré de façon dynamique qui permet l'ajout et la suppression de variables. MultiSegment est un vector contenant des segments, nous l'avons implémenter directement dans le graphisme.

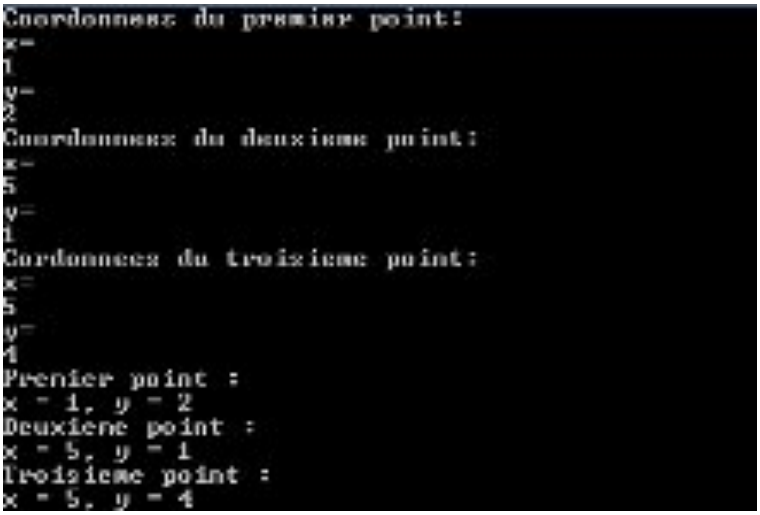
R) Dessin Technique

Nos choix de programmation nous ont amené à faire de la classe DessinTechnique un vector. DessinTechnique est un vector contenant des figures, nous l'avons implémenter directement dans le graphisme.

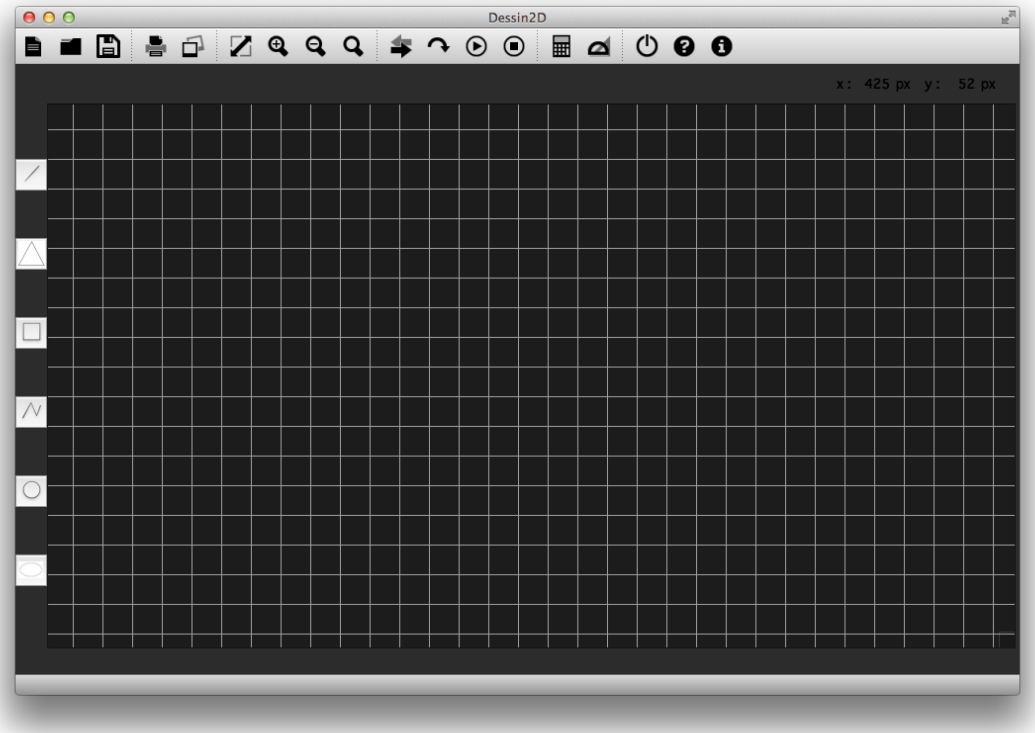
IV. Interface Graphique

A) Terminal

Le but du projet est de permettre à l'utilisateur d'interagir avec le programme, via un terminal, afin de créer des dessins et leur appliquer des translations.

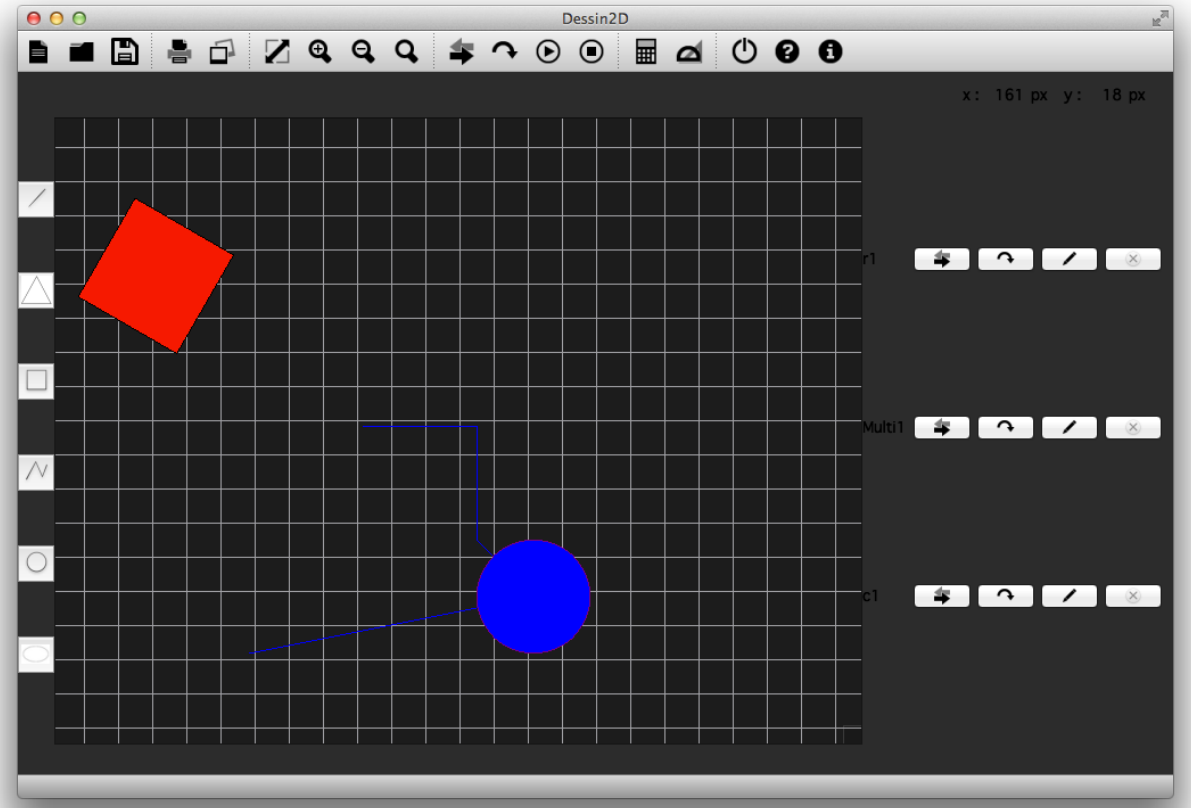


B) Interface avec Qt



L'interaction avec un terminal peut sembler compliqué, de plus nous n'avons pas une idée concrète des dessins technique créés car aucun visuel n'est implanté. C'est donc la raison pour laquelle nous avons choisi de créer une interface graphique compatible Windows et Mac et qui permet une visualisation facile des dessins. Nous avons choisi la bibliothèque Qt qui permet de créer des applications puissantes et performantes, nous avons évidemment développé le logiciel en C++.

Un utilisateur peut, via l'application, créer plusieurs figures et leur appliquer des translations et même des rotations. L'avantage avec une interface graphique et que nous avons tout de suite un rendu et nous pouvons déterminé si l'opération mathématique s'est bien déroulé ou non. Un système de couleurs permet de mettre en évidences des figures, l'utilisateur peut changer la couleur du trait et du fond de chaque figure.



L'utilisateur peut sauvegarder les dessins afin de pouvoir travailler ultérieurement avec. Nous avons choisi une sauvegarde des données de type XML car Qt permet une gestion flexible de ce type de document,

Il est également possible d'imprimer son dessin, mais aussi de l'exporter en pdf et en image.

Vous trouverez plus d'informations de fonctionnement dans le mode d'emploi de l'application Dessin2D.

V. Conclusion

Nous sommes parvenus à créer un logiciel de dessin assisté 2D. Ce projet nous aura permis d'apprendre en profondeur le C++ mais aussi à être plus rigoureux dans notre analyse grâce à l'UML. De plus nous avons pu apprendre à développé une interface graphique avec la célèbre bibliothèque Qt. Nous n'avons pas rencontré de réelle difficultés durant le projet aussi bien du point de vue Analyse que développement. De plus la planification des tâches et la gestion du temps ont été respecté tout le long du projet. Nous avons donc respecté nos objectifs et ce projet fut une très belle expériences.

Nous avons fait face à plusieurs problème durant le développement mais tous ont été surmonté. Le projet a été terminé dans les temps, néanmoins, avec du temps en plus, la gestion de la 3D aurait pu être réalisable.

VI . Annexe

1- Codes sources des classes

A) Point2D

1) Header

```
/**
*       Classe Point2D. Cette classe modélise un pofloat positionné dans un environnement 2D
*
*
*     @Author Clin Matthieu
*     @Author Vecchio Quentin
*
*     @version 1.0.0.0 on 2013-11-15
*/
#ifndef Point2D_h
#define Point2D_h

#include <iostream>
#include <ostream>

using namespace std;
class Point2D {

 public:

        /**
        *       Constructeur sans paramètre.
        *       Instancie un Point2D de coordonnée (0.0,0.0)
        */
  Point2D();

        /**
        *       Constructeur avec comme paramètres les coordonnées du Point2D
        *       @params float x, l'abscisse du pofloat
        *       @params float y, l'ordonnée du pofloat
        */
  Point2D(float x, float y);

        /**
        *       Getters et setters de X et Y
        */
  float getX();
```

```
  float getY();

  void setX(float x);

  void setY(float y);


        /**
        *       Translation du pofloat par rapport à un autre
        *       On déplace le pofloat suivant les coordonées du pofloat donné
        *       @params Point2D t, le pofloat pour la translation
        */
        void translation(Point2D t);

        /**
        *       Idem
        *       @params float x, le déplacement selon l'axe x
        *       @params float y, le déplacement selon l'axe y
        */
        void translation(float x, float y);


        /**
        *       Fonction permettant de savoir si deux pofloats sont placés aux mêmes endroit sur
l'environnement 2D
        *       @params Point2D pofloat, le pofloat à comparé avec l'objet courant
        */
  bool estIdentique(Point2D pofloat);


        /**
        *       Fonction d'affichage du pofloat
        *
        */
        void affiche(ostream& flux) const;

        friend ostream& operator <<(ostream& flux, const Point2D& p);

 private:
   float x;
   float y;

};

#endif // Point2D_h
```

2)Corps

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#include "Point2D.h"

using namespace std;

Point2D::Point2D()
{
		this->x = 0;
		this->y = 0;
}

Point2D::Point2D(float x, float y)
{
		this->x = x;
		this->y = y;
}

float Point2D::getX()
{
		return this->x;
}

float Point2D::getY()
{
		return this->y;
}

void Point2D::setX(float x)
{
		this->x = x;
}

void Point2D::setY(float y)
{
		this->y = y;
}

void Point2D::translation(Point2D t){
		this->x += t.getX();
		this->y += t.getY();
}

void Point2D::translation(float x, float y){
		this->x += x;
		this->y += y;
}


bool Point2D::estIdentique(Point2D pofloat)
```

```
{
	return this->x == pofloat.x && this->y == pofloat.y;
}

void Point2D::affiche(ostream& flux) const
{
	flux << "(" << this->x << "," << this->y << ")" << endl;
}

ostream& operator <<(ostream& flux, const Point2D& p)
{
	p.affiche(flux);
	return flux;
}
```

B) Figure

1)Header

```
/**
*	Classe figure virtuelle représentant les éléments nécessaires pour être considérer comme une figure.
*	Une figure est forcément composée d'un point qui corespondra à son origine.
*	Une figure est muni de deux opérations indispensables:
*				- La translation
*				- L'affichage
*
*
*
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef Figure_h
#define Figure_h

#include "Point2D.h"
#include <iostream>
#include <ostream>

using namespace std;
class Figure {

 protected:
   Point2D A;

 public:

	/**
	*	Constructeur d'une figure.
	*	La figure peut être instanciée avec aucun paramètre, des valeurs par défaut sont alors prises
	*/
   Figure();
```

```
        /**
        *       Constructeur avec deux paramètres
        *       @params float x, l'abscisse du point
        *       @params float y, l'ordonnée du point
        */
    Figure(float x, float y);

        /**
        *       Constructeur avec un point
        *       @params Point2d point, le point d'origine de la figure
        */
    Figure(Point2D point);

        /*
        *       Fonction de translation de la figure.
        *       Chaque figure doit être muni de cette opération.
        *       L'opération de translation selon les axes X et Y d'une figure consiste à "déplacer"
        *       celle-ci dans le repère. On applique donc la fonction translation à chaque élément de la
figure
        *       @params Point2D point, le déplacement a effectuer sur la figure
        *
        */
    virtual void translation(Point2D point);

        /*
        *       Fonction d'affichage dans le terminal de la figure
        *       Une figure doit pouvoir être affichée dans le terminal, c'est à dire que l'on représente
        *       la figure en détaillant ses caractéristiques (les points)
        */
    virtual void affiche(ostream& flux) const;

        /*
        *       Les getters et setters du Point2D A
        */
    Point2D getA();

    void setA(Point2D A);


        /*
        *       Surcharge de l'opérateur d'affichage
        */

    friend ostream& operator <<(ostream& flux, const Figure& p);

};

#endif // Figure_h
```

2)Corps

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#include "Figure.h"

Figure::Figure()
{
	this->A = Point2D();
}

Figure::Figure(float x, float y)
{
	this->A = Point2D(x,y);
}

Figure::Figure(Point2D point)
{
	this->A = point;
}

void Figure::translation(Point2D point)
{
	this->A.translation(point);
}

void Figure::affiche(ostream& flux) const
{
	this->A.affiche(flux);
}

Point2D Figure::getA()
{
	return this->A;
}

void Figure::setA(Point2D A)
{
	this->A = A;
}

ostream& operator <<(ostream& flux, const Figure& p)
{
	p.affiche(flux);
	return flux;
}
```

C) Segment

1)Header

```
/**
*       Classe Segment. Représente un Segment 2D
*     @Author Clin Matthieu
*     @Author Vecchio Quentin
*
*     @version 1.0.0.0 on 2013-11-15
*/
#ifndef Segment_h
#define Segment_h

#include "Figure.h"
#include "Point2D.h"
#include <iostream>
#include <ostream>
#include <cmath>
using namespace std;
class Segment : public Figure {

 protected:
   Point2D B;

 public:

        Segment();

        /**
        *       @params Point2D A, le point de départ A
        *       @params Point2D B, le point d'arrivée B
        */
   Segment(Point2D A, Point2D B);


        /**
        *       Getters et Setters du point B
        */
   Point2D getB();

   void setB(Point2D B);

        /**
        *       Affichage du segment dans le terminal
        */
   void affiche(ostream& flux) const;

        /**
        *       Translation du segment suivant un point
        */
   void translation(Point2D point);

        /**
```

```
        *       Fonction permettant de savoir si deux segments sont perpendiculaires
        *       @params Segment segment, le segment qui peut être perpendiculaire à l'objet courant
        *       @return true, si les segments sont perpendiculaires
        *       @return false, sinon
        */
    bool estPerpendiculaire(Segment segment);

        /**
        *       Fonction permettant de savoir si deux segments sont parallèles
        *       @params Segment segment, le segment qui peut être parrallèle à l'objet courant
        *       @return true, si les segments sont parrallèles
        *       @return false, sinon
        *
        */
    bool estParallele(Segment segment);

        /**
        *       Fonction permettant de savoir si deux segments se coupent
        *       @params Segment segment, le segment qui peut être couper l'objet courant
        *       @return true, si les segments se coupent
        *       @return false, sinon
        */
    bool seCoupe(Segment segment);

        /**
        *       Renvoit la longueur du segment
        *       @return float, la longueur du segment
        */
         float longueur();

        friend ostream& operator <<(ostream& flux, const Segment& s);
};
#endif // Segment_h
```

2)Corps

```
/**
*     @Author Clin Matthieu
*     @Author Vecchio Quentin
*
*     @version 1.0.0.0 on 2013-11-15
*/
#include "Segment.h"


Segment::Segment(){}

Segment::Segment(Point2D A, Point2D B):Figure(A),B(B)
{
}

Point2D Segment::getB()
{
        return this->B;
}
```

```
void Segment::setB(Point2D B)
{
	this->B = B;
}

void Segment::affiche(ostream& flux) const
{
	Figure::affiche(flux);
	this->B.affiche(flux);
}

void Segment::translation(Point2D point)
{
	Figure::translation(point);
	this->B.translation(point);
}

bool Segment::estPerpendiculaire(Segment segment)
{
	Point2D v1( this->A.getX() - this->B.getX(),
					this->A.getY() - this->B.getY());

	Point2D v2( segment.getA().getX() - segment.getB().getX(),
					segment.getA().getY() - segment.getB().getY());

	return v1.getX() * v2.getX() + v2.getY() * v1.getY() == 0;
}

bool Segment::estParallele(Segment segment)
{
	Point2D v1( this->A.getX() - this->B.getX(),
					this->A.getY() - this->B.getY());

	Point2D v2( segment.getA().getX() - segment.getB().getX(),
					segment.getA().getY() - segment.getB().getY());

	return v1.getX() * v2.getY() == v2.getX() * v1.getY();
}

bool Segment::seCoupe(Segment segment)
{
   return false;
}

ostream& operator <<(ostream& flux, const Segment& s)
{
	s.affiche(flux);
	return flux;
}

float Segment::longueur(){
	int deltaX = this->B.getX()- this->A.getX();
	int deltaY = this->B.getY()- this->A.getY();
	return sqrt(deltaX*deltaX + deltaY*deltaY);
}
```

D) Quadrilatere

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Quadrilatere_h
#define Quadrilatere_h

#include "../Base/Point2D.h"
#include "../Base/Segment.h"
#include <iostream>
#include <ostream>
using namespace std;

class Quadrilatere : public Segment {
 public:
        Quadrilatere();

        /**
        *       Constructeur de la class quadrilatère
        *       @params Point2D A, le point A du quadrilatère
        *       @params Point2D B, le point B du quadrilatère
        *       @params Point2D C, le point C du quadrilatère
        *       @params Point2D D, le point D du quadrilatère
        */
   Quadrilatere(Point2D A, Point2D B, Point2D C, Point2D D);

        /**
        *       Getters et Setters des points C et D
        */
   Point2D getC();

   Point2D getD();

   void setC(Point2D C);

   void setD(Point2D D);

   void affiche(ostream& flux) const;

   void translation(Point2D point);

 protected:
   Point2D C;
   Point2D D;
};

#endif // Quadrilatere_h
```

2)Corps

```
#include "Quadrilatere.h"

Quadrilatere::Quadrilatere(){}

Quadrilatere::Quadrilatere(Point2D A, Point2D B, Point2D C, Point2D D) : Segment(A,B), C(C), D(D)
{
}

Point2D Quadrilatere::getC()
{
   return this->C;
}

Point2D Quadrilatere::getD()
{
   return this->D;
}

void Quadrilatere::setC(Point2D C)
{
   this->C = C;
}

void Quadrilatere::setD(Point2D D)
{
   this->D = D;
}

void Quadrilatere::affiche(ostream& flux) const
{
   Segment::affiche(flux);
   this->C.affiche(flux);
   this->D.affiche(flux);
}

void Quadrilatere::translation(Point2D point)
{
   Segment::translation(point);
         this->C.translation(point);
   this->D.translation(point);
}

ostream& operator <<(ostream& flux, const Quadrilatere& q)
{
         q.affiche(flux);
         return flux;
}
```

E) Quadrangle

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Quadrangle_h
#define Quadrangle_h

#include "../Base/Point2D.h"
#include "Quadrilatere.h"

using namespace std;

class Quadrangle : public Quadrilatere {

 public:
        /**
        *       Constructeur de la class Quadrangle
        *       @params Point2D A, le point A du Quadrangle
        *       @params Point2D B, le point B du Quadrangle
        *       @params Point2D C, le point C du Quadrangle
        *       @params Point2D D, le point D du Quadrangle
        */
   Quadrangle(Point2D A, Point2D B, Point2D C, Point2D D);

        /**
        *       Fonction qui test si les points correspondent à un quadrangle
        *       @return true, si c'est un quadrangle
        *       @return false, sinon
        */
   bool estQuadrangle();
};

#endif // Quadrangle_h
```

2)Corps

F) Parallelogramme

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Parallelogramme_h
#define Parallelogramme_h

#include "../Base/Point2D.h"
#include "Quadrilatere.h"
#include "../Base/Segment.h"

using namespace std;
class Parallelogramme : public Quadrilatere {

 public:

        Parallelogramme();

        /**
        *       Constructeur de la class Parallelogramme
        *       @params Point2D A, le point A du parallélogramme
        *       @params Point2D B, le point B du parallélogramme
        *       @params Point2D C, le point C du parallélogramme
        *       @params Point2D D, le point D du parallélogramme
        */
   Parallelogramme(Point2D A, Point2D B, Point2D C, Point2D D);

        /**
        *       Fonction qui test si les points correspondent à un parallelogramme
        *       @return true, si c'est un parallélogramme
        *       @return false, sinon
        */
   bool estParallelogramme();
};

#endif // Parallelogramme_h
```

2)Corps

```
#include "Parallelogramme.h"

Parallelogramme::Parallelogramme(){}

Parallelogramme::Parallelogramme(Point2D A, Point2D B, Point2D C, Point2D D) : Quadrilatere(A,B,C,D)
{
}

bool Parallelogramme::estParallelogramme()
```

```
{
	Segment s1(A,B);
	Segment s2(B,C);
	Segment s3(C,D);
	Segment s4(D,A);

	return s1.estParallele(s3) && s2.estParallele(s4);

}
```

G) Losange

1)Header

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef Losange_h
#define Losange_h

#include "Parallelogramme.h"
#include "../Base/Point2D.h"
#include "../Base/Segment.h"

using namespace std;
class Losange : public Parallelogramme {

 public:

	/**
	*	Constructeur de la class Losange
	*	@params Point2D A, le point A du losange
	*	@params Point2D B, le point B du losange
	*	@params Point2D C, le point C du losange
	*	@params Point2D D, le point D du losange
	*/
   Losange(Point2D A, Point2D B, Point2D C, Point2D D);

	/**
	*	Fonction qui test si les points données correspondent à un losange
	*	@return true, si c'est un losange
	*	@return false, sinon
	*/
   bool estLosange();
};

#endif // Losange_h
```

2)Corps

```
#include "Losange.h"

Losange::Losange(Point2D A, Point2D B, Point2D C, Point2D D) : Parallelogramme(A,B,C,D)
{
}


bool Losange::estLosange()
{
	Segment s1(A,B);
	Segment s2(B,C);
	Segment s3(C,D);
	Segment s4(D,A);

  return s1.estParallele(s3) && s1.longueur() == s2.longueur();
}
```

H) Carre

1)Header

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef Carre_h
#define Carre_h

#include "Parallelogramme.h"
#include "../Base/Point2D.h"
#include "../Base/Segment.h"

using namespace std;

class Carre : public Parallelogramme {

 public:

	Carre();

	/**
	*	Constructeur de la class Carre
	*	@params Point2D A, le point A du carré
	*	@params Point2D B, le point B du carré
	*	@params Point2D C, le point C du carré
	*	@params Point2D D, le point D du carré
	*
	*/
   Carre(Point2D A, Point2D B, Point2D C, Point2D D);
```

```
        /**
        *       Fonction de test pour savoir si les coordonnées des points correspondent à un carré
        *       @return bool, true si c'est un carré
        *       @return bool, false sinon
        */
   bool estCarre();
};

#endif // Carre_h
```

2)Corps

```
#include "Carre.h"

Carre::Carre(){}

Carre::Carre(Point2D A, Point2D B, Point2D C, Point2D D) : Parallelogramme(A,B,C,D)
{
}

bool Carre::estCarre()
{
        Segment s1(A,B);
        Segment s2(B,C);
        Segment s3(C,D);
        Segment s4(D,A);

  return estParallelogramme() && s1.longueur() == s2.longueur() && s1.estPerpendiculaire(s2);
}
```

I) Rectangle

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Rectangle_h
#define Rectangle_h

#include "Parallelogramme.h"
#include "../Base/Point2D.h"
#include "../Base/Segment.h"

using namespace std;
class Rectangle : public Parallelogramme {

 public:
```

```
        /**
        *       Constructeur de la class Rectangle
        *       @params Point2D A, le point A du Rectangle
        *       @params Point2D B, le point B du Rectangle
        *       @params Point2D C, le point C du Rectangle
        *       @params Point2D D, le point D du Rectangle
        */
   Rectangle(Point2D A, Point2D B, Point2D C, Point2D D);

        /**
        *       Fonction qui test si les points données correspondent à un rectangle
        *       @return true, si c'est un rectangle
        *       @return false, sinon
        */
   bool estRectangle();
};

#endif // Rectangle_h
```

2)Corps

```
#include "Rectangle.h"

Rectangle::Rectangle(Point2D A, Point2D B, Point2D C, Point2D D) : Parallelogramme(A,B,C,D)
{
}

bool Rectangle::estRectangle()
{
        Segment s1(A,B);
        Segment s2(B,C);
        Segment s3(C,D);
        Segment s4(D,A);

  return estParallelogramme() && s1.longueur() == s3.longueur() && s1.estPerpendiculaire(s2);
}
```

J) Triangle

1)Header

```
#include "Triangle.h"

using namespace std;
Triangle::Triangle(Point2D A, Point2D B, Point2D C) : Segment(A,B),C(C)
{
}

Triangle::Triangle(Segment base, Point2D C) : Segment(base),C(C)
{
}

Point2D Triangle::getC()
```

```
{
	return this->C;
}

void Triangle::setC(Point2D C)
{
	this->C = C;
}

void Triangle::affiche(ostream& flux) const
{
	Segment::affiche(flux);
	this->C.affiche(flux);
}

void Triangle::translation(Point2D point)
{
	Segment::translation(point);
	this->C.translation(point);
}

bool Triangle::estPlat()
{
	if((this->A.getX() == this->B.getX() && this->B.getX() == this->C.getX()) || (this->A.getY() == this->B.getY() && this->B.getY() == this->C.getY()))
	{
		return true;
	}
	return false;
}

ostream& operator <<(ostream& flux, const Triangle& t)
{
	t.affiche(flux);
	return flux;
}
```

2)Corps

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef TRectangle_h
#define TRectangle_h

#include "../Base/Point2D.h"
#include "../Base/Segment.h"
#include "Triangle.h"

using namespace std;
class TRectangle : public Triangle {

 public:
```

```
        /**
        *       Construction
        *       @params Point2D A, le point A
        *       @params Point2D B, le point B
        *       @params Point2D C, le point C
        */
   TRectangle(Point2D A, Point2D B, Point2D C);

        /**
        *       Constructeur
        *       @params Segment base, le segment de base
        *       @params Point2D point, le point C du triangle
        */
   TRectangle(Segment base, Point2D point );

        /**
        *       Fonction qui test si le triangle est rectangle
        *       @return true, si il est rectangle
        *       @return false, sinon
        */
   bool estTRectangle();
};

#endif // TRectangle_h
```

K) Isocele

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Isocele_h
#define Isocele_h

#include "../Base/Point2D.h"
#include "../Base/Segment.h"
#include "Triangle.h"

using namespace std;
class Isocele : public Triangle {

 public:

        /**
        *       Constructeur
        *       @params Point2D A, le point A
        *       @params Point2D B, le point B
        *       @params Point2D C, le point C
        */
```

```
    Isocele(Point2D A, Point2D B, Point2D C);

        /**
        *       Constructeur
        *       @params Semgnet base, la base du triangle
        *       @params Point2D point, le point C
        */
    Isocele(Segment base, Point2D point);

        /**
        *       Fonction qui test si le triangle est isocèle
        *       @return true, si il est isocele
        *       @return false, sinon
        */
    bool estIsocele();
};

#endif // Isocele_h
```

2)Corps

```
#include "Isocele.h"

Isocele::Isocele(Point2D A, Point2D B, Point2D C) : Triangle(A,B,C)
{
}

Isocele::Isocele(Segment base, Point2D point) : Triangle(base,point)
{
}

bool Isocele::estIsocele()
{
        Segment s1(A,B);
        Segment s2(A,C);
        Segment s3(B,C);

  return s1.longueur() == s2.longueur() || s1.longueur() == s3.longueur() || s2.longueur() == s3.longueur();
}
```

L) Equilateral

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef Equilateral_h
#define Equilateral_h
```

```
#include "../Base/Point2D.h"
#include "../Base/Segment.h"
#include "Triangle.h"

using namespace std;
class Equilateral : public Triangle {

 public:

        /**
        *       Constructeur
        *       @params Point2D A, le point A
        *       @params Point2D B, le point B
        *       @params Point2D C, le point C
        */
   Equilateral(Point2D A, Point2D B, Point2D C);

        /**
        *       Constructeur
        *       @params Segment base, la base du triangle
        *       @params Point2D point, le point C du triangle
        */
   Equilateral(Segment base, Point2D point);

        /**
        *       Fonction qui test si le triangle est équilatéral
        *       @return true, si il est équilatéral
        *       @return false, sinon
        */
   bool estEquilateral();
};

#endif // Equilateral_h
```

2)Corps

```
#include "Equilateral.h"

Equilateral::Equilateral(Point2D A, Point2D B, Point2D C) : Triangle(A,B,C)
{
}

Equilateral::Equilateral(Segment base, Point2D point) : Triangle(base,point)
{
}

bool Equilateral::estEquilateral()
{
        Segment s1(A,B);
        Segment s2(A,C);
        Segment s3(B,C);

  return s1.longueur() == s2.longueur() && s3.longueur() == s2.longueur();
}
```

M) TRectangle

1)Header

```
/**
*       @Author Clin Matthieu
*       @Author Vecchio Quentin
*
*       @version 1.0.0.0 on 2013-11-15
*/
#ifndef TRectangle_h
#define TRectangle_h

#include "../Base/Point2D.h"
#include "../Base/Segment.h"
#include "Triangle.h"

using namespace std;
class TRectangle : public Triangle {

 public:

        /**
        *       Construction
        *       @params Point2D A, le point A
        *       @params Point2D B, le point B
        *       @params Point2D C, le point C
        */
   TRectangle(Point2D A, Point2D B, Point2D C);

        /**
        *       Constructeur
        *       @params Segment base, le segment de base
        *       @params Point2D point, le point C du triangle
        */
   TRectangle(Segment base, Point2D point );

        /**
        *       Fonction qui test si le triangle est rectangle
        *       @return true, si il est rectangle
        *       @return false, sinon
        */
   bool estTRectangle();
};

#endif // TRectangle_h
```

2)Corps

```
#include "TRectangle.h"

TRectangle::TRectangle(Point2D A, Point2D B, Point2D C) : Triangle(A,B,C)
{
}

TRectangle::TRectangle(Segment base, Point2D point ) : Triangle(base,point)
{
}

bool TRectangle::estTRectangle()
{
	Segment s1(A,B);
	Segment s2(A,C);
	Segment s3(B,C);
	return s1.estPerpendiculaire(s2) || s1.estPerpendiculaire(s3) || s2.estPerpendiculaire(s3);
}
```

N) Cercle

1)Header

```
/**
*	Classe Cercle. Modélise un cercle dans un environnement 2D
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef Cercle_h
#define Cercle_h

#include "../Base/Figure.h"
#include "../Base/Point2D.h"

using namespace std;

class Cercle : public Figure {


 protected:
   int rayon;


 public:

	/**
	*	Constructeur par défaut de cercle
	*	Point d'origine (2,2) et rayon 1
	*/
   Cercle();
```

```
        /**
        *       Constructeur avec le centre du cercle et un rayon par défaut de 1
        *       @params Point2D point, l'origine du cercle
        */
    Cercle(Point2D point);

        /**
        *       Constructeur avec le centre du cercle et le rayon
        *       @params Point2D point, le point d'origine du cercle
        *       @params int rayon, le rayon du cercle
        */
    Cercle(Point2D point, int rayon);

        /**
        *       Getter et Setter du rayon
        */
    int getRayon();

    void setRayon(int rayon);

        /**
        *       Fonction d'affichage du cercle dans un terminal
        */
    void affiche(ostream& flux) const;

        /**
        *       Translation du cercle
        */
    void translation(Point2D point);

        friend ostream& operator <<(ostream& flux, const Cercle& c);

};

#endif // Cercle_h
```

2)Corps

```
/**
*     @Author Clin Matthieu
*     @Author Vecchio Quentin
*
*     @version 1.0.0.0 on 2013-11-15
*/
#include "Cercle.h"


Cercle::Cercle()
{
        this->A = Point2D(2,2);
        this->rayon = 1;
}

Cercle::Cercle(Point2D point):Figure(point)
{
```

```
	this->rayon = 1;
}

Cercle::Cercle(Point2D point, int rayon):Figure(point)
{
	this->rayon = rayon;
}

int Cercle::getRayon()
{
	return this->rayon;
}

void Cercle::setRayon(int rayon)
{
	this->rayon = rayon;
}

void Cercle::affiche(ostream& flux) const
{
	cout << "centre: " <<this->A;
	cout << "rayon : "<< this->rayon << endl;
}

void Cercle::translation(Point2D point)
{
	Figure::translation(point);
}

ostream& operator <<(ostream& flux, const Cercle& c)
{
	c.affiche(flux);
	return flux;
}
```

O) Ellipse

1)Header

```
/**
*	Classe Elippse. Modélise une Elippse dans un environnement 2D
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef Elippse_h
#define Elippse_h
```

```
#include "../Base/Figure.h"
#include "../Base/Point2D.h"
#include "Cercle.h"

using namespace std;

class Elippse : public Cercle {
 private :
        Point2D point2;
 public:

        /**
        *       Constructeur par défaut d'Elippse
        *       Point d'origine (2,2) et rayon 1
        */
   Elippse();

        /**
        *       Constructeur avec le centre du cercle et un rayon par défaut de 1
        *       @params Point2D point, l'origine du cercle
        */
   Elippse(Point2D point1,Point2D point2);

        /**
        *       Constructeur avec le centre du cercle et le rayon
        *       @params Point2D point, le point d'origine du cercle
        *       @params int rayon, le rayon du cercle
        */
   Elippse(Point2D point1,Point2D point2, int rayon);

        /**
        *       Getter et Setter du point 2
        */
        Point2D getPoint2();
        void setPoint2(Point2D point);

   void affiche(ostream& flux) const;

        /**
        *       Translation de l'ellipse
        */
   void translation(Point2D point);

        friend ostream& operator <<(ostream& flux, const Elippse& c);

};

#endif // Cercle_h
```

2)Corps

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#include "Elippse.h"


Elippse::Elippse()
{
	this->A = Point2D(2,2);
	this->rayon = 1;
}

Elippse::Elippse(Point2D point1,Point2D point2): Cercle(point1), point2(point2)
{
	this->rayon = 1;
}

Elippse::Elippse(Point2D point1,Point2D point2, int rayon):Cercle(point1,rayon), point2(point2)
{
}

Point2D Elippse::getPoint2()
{
	return this->point2;
}

void Elippse::setPoint2(Point2D point)
{
	this->point2 = point;
}

void Elippse::affiche(ostream& flux) const
{
	cout << "Point 1 : " << this->A;
   cout << "Point 2 : " << this->point2;
	cout << "rayon : "<< this->rayon << endl;
}

void Elippse::translation(Point2D point)
{
	Cercle::translation(point);
}

ostream& operator <<(ostream& flux, const Elippse& e)
{
	e.affiche(flux);
	return flux;
}
```

P) ArcCercle

1)Header

```
/**
*	@Author Clin Matthieu
*	@Author Vecchio Quentin
*
*	@version 1.0.0.0 on 2013-11-15
*/
#ifndef ArcCercle_h
#define ArcCercle_h

#include "Cercle.h"


class ArcCercle : public Cercle {

private:
	float angle;
	Point2D pointDepart;
public:

	/**
	* Constructeur de la class ArcCercle
	*	@params Cercle cercle, le cercle où se situe l'arc de cercle
	*	@params float angle, l'angle à partir du point
	*	@params Point2ED pointDepart, le point de départ de l'arc de cercle
	*/
	ArcCercle(Cercle cercle, float angle ,Point2D pointDepart);

	/**
	*	Les getters et setters de l'angle et du point de départ
	*/
	float getAngle();

	Point2D getPointDepart();

	void setAngle(float angleA);

	void setPointDepart(Point2D pointDepart);


};

#endif // ArcCercle_h
```

2)Corps

```
/**
*      @Author Clin Matthieu
*      @Author Vecchio Quentin
*
*      @version 1.0.0.0 on 2013-11-15
*/
#include "ArcCercle.h"

        // :Cercle(cercle),angle(angle),pointDepart(pointDepart){}
    ArcCercle::ArcCercle(Cercle cercle, float angle ,Point2D pointDepart){
                this->A = cercle.getA();
                this->rayon = cercle.getRayon();
                this->angle = angle;
                this->pointDepart = pointDepart;
        }

    float ArcCercle::getAngle(){
                return this->angle;
        }

    Point2D ArcCercle::getPointDepart(){
                return this->pointDepart;
        }

    void ArcCercle::setAngle(float angleA){
                this->angle = angleA;
        }

    void ArcCercle::setPointDepart(Point2D pointDepart){
                this->pointDepart = pointDepart;
        }
```

2- Codes sources du graphisme

A) Fenêtre principale

1) Header

```
#ifndef FENETREPRINCIPALE_H
#define FENETREPRINCIPALE_H

#include <QMainWindow>
#include <QGraphicsScene>
#include <QLabel>
#include <QSpinBox>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include <QMouseEvent>
#include <QGraphicsSceneMouseEvent>
#include <vector>
#include <QDialog>
#include <QPrinter>
#include <QPrintDialog>
#include <QtXml/QDomDocument>
#include <QtXml/QDomElement>
#include <QtXml/QDomNode>
#include <QDebug>
//Mes objets
#include "threadanimation.h"
#include "gui_point.h"
#include "vuegraphique.h"
#include "formsegment.h"
#include "formcarre.h"
#include "formtriangle.h"
#include "formcercle.h"
#include "formelippse.h"
#include "formmultisegment.h"
#include "presentationobjet.h"
#include "bu.h"
namespace Ui {
class fenetrePrincipale;
}
using namespace std;
class fenetrePrincipale : public QMainWindow
{
   Q_OBJECT
public:
   explicit fenetrePrincipale(QWidget *parent = 0);
   ~fenetrePrincipale();
   //Mes fonctions
      void afficheFormNvObj();
```

```
    void traceRec();
    void traceSeg();
    void traceCercle();
    void traceTri();
    void traceElippse();
    void traceMultiSeg();
private slots:
  //Slots personnels
    void annuleNouveauObjet();
    void nouveauObjet();
    void rafraichir();
    void suppression();
    void couleurT();
    void couleurF();
    void valideTrans();
    void annuleTrans();
    void valideRot();
    void annuleRot();
  //Slots pour les btns de dessin
    void on_btnSegment_clicked();
    void on_btnCarre_clicked();
    void on_btnMultiSegment_clicked();
    void on_btnTriangle_clicked();
    void on_btnCercle_clicked();
    void on_btnEllipse_clicked();
  //Slots pour les actions du menus
    void on_actionImprimer_triggered();
    void on_actionQuitter_3_triggered();
    void on_actionPlein_cran_triggered();
    void on_actionTutoriels_triggered();
    void on_actionD_veloppeur_triggered();
    void on_actionZoom_triggered();
    void on_actionZoom_2_triggered();
    void on_actionVue_ensemble_triggered();
    void on_actionCalculette_triggered();
    void on_actionMath_matique_triggered();
    void on_actionPr_c_dent_triggered();
    void on_actionSuivant_triggered();
    void on_actionTranslation_3_triggered();
    void on_actionRotation_3_triggered();
    void on_actionLancer_Animation_2_triggered();
    void on_actionEnregistrer_triggered();
  //Fonction de sauvegarde
    void ecritureDOM();
    void ajoutDOM(QString *nomObjet,QFigure *fig, int id);
    void sauvegardeDOM();
    void on_actionOuvrir_triggered();
  //Fonction d'ouverture
    void lectureDOM();
    void lireDOM();
    void on_actionArreter_Animation_triggered();

    void on_actionExportation_triggered();

private:
  //Variables
```

```
    Ui::fenetrePrincipale *ui;
    Objet obj;
    QColor colorF;
    QColor colorT;
    int rot;
    int transX;
    int transY;
    QString *lienSauvegarde;
  //Fonctions privées
    void changePosCurseur(int x,int y);
  //Objets pour trans et rotation
    QHBoxLayout *boxTrans;
    QHBoxLayout *boxRot;
    QLabel *labelT;
    QLabel *labelR;
    QLabel *labelAngle;
    QLabel *labelX;
    QLabel *labelY;
    QLabel *labelTps;
    QSpinBox *spinX;
    QSpinBox *spinY;
    QSpinBox *spinTps;
    QPushButton *btnValide;
    QPushButton *btnAnnule;
    vector<Anim> pileAnim;
    vector<ThreadAnimation *> pileThread;
    bool cont;
  //Formulaire Objets
    formSegment *formSeg;
    formCarre *formCar;
    formTriangle *formTri;
    formCercle *formCer;
    formElippse *formEli;
    formmultisegment *formMultiSeg;
  //Dessin technique
    vector<QDessin> dessin;
  //Fichier pour la sauvegarde
    QDomDocument *doc;
    QDomElement *dessinX;
    QFile *file;
    QTextStream *out;
};

#endif // FENETREPRINCIPALE_H
```

2) Corps

```
#include "fenetreprincipale.h"
#include "ui_fenetreprincipale.h"

fenetrePrincipale::fenetrePrincipale(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::fenetrePrincipale)
{
  //Définition de la fenetre;
```

```
    ui->setupUi(this);
  //Definition gestion souris
    ui->view->setMouseTracking(true);
  //Définition des formulaires
    this->formSeg = new formSegment();
    this->formCar = new formCarre();
    this->formTri = new formTriangle();
    this->formCer = new formCercle();
    this->formEli = new formElippse();
    this->formMultiSeg = new formmultisegment();
  //label
    ui->view->labelX = ui->labelX;
    ui->view->labelY = ui->labelY;
  //Init les rot et trans
    this->rot = this->transX = this->transY = 0;
  //Init lien
    this->lienSauvegarde = new QString();
}

fenetrePrincipale::~fenetrePrincipale()
{
  delete ui;
}
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  Déclaration des fonctions                                                   //
//
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  void fenetrePrincipale::afficheFormNvObj()
  {
    //Création du formulaire
      ui->btnCarre->setEnabled(false);
      ui->btnCercle->setEnabled(false);
      ui->btnEllipse->setEnabled(false);
      ui->btnMultiSegment->setEnabled(false);
      ui->btnSegment->setEnabled(false);
      ui->btnTriangle->setEnabled(false);
      switch (this->obj)
      {
        case SEGMENT:
          this->formSeg->affiche();
          QObject::connect(this->formSeg->btnValide, SIGNAL(clicked()), this, SLOT(nouveauObjet()));
          QObject::connect(this->formSeg->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
          QObject::connect(this->formSeg->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
          ui->layoutDroit->addLayout(this->formSeg->boxPrincipale);
          break;
        case CERCLE:
          this->formCer->affiche();
          QObject::connect(this->formCer->btnValide, SIGNAL(clicked()), this, SLOT(nouveauObjet()));
          QObject::connect(this->formCer->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
          QObject::connect(this->formCer->btnCFond, SIGNAL(clicked()), this, SLOT(couleurF()));
          QObject::connect(this->formCer->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
```

```
            ui->layoutDroit->addLayout(this->formCer->boxPrincipale);
            break;
        case ELLIPSE:
            this->formEli->affiche();
            QObject::connect(this->formEli->btnValide, SIGNAL(clicked()), this, SLOT(nouveauObjet()));
            QObject::connect(this->formEli->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
            QObject::connect(this->formEli->btnCFond, SIGNAL(clicked()), this, SLOT(couleurF()));
            QObject::connect(this->formEli->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
            ui->layoutDroit->addLayout(this->formEli->boxPrincipale);
            break;
        case TRIANGLE:
            this->formTri->affiche();
            QObject::connect(this->formTri->btnValide, SIGNAL(clicked()), this, SLOT(nouveauObjet()));
            QObject::connect(this->formTri->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
            QObject::connect(this->formTri->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
            ui->layoutDroit->addLayout(this->formTri->boxPrincipale);
            break;
        case CARRE:
            this->formCar->affiche();
            QObject::connect(this->formCar->btnValide, SIGNAL(clicked()), this, SLOT(nouveauObjet()));
            QObject::connect(this->formCar->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
            QObject::connect(this->formCar->btnCFond, SIGNAL(clicked()), this, SLOT(couleurF()));
            QObject::connect(this->formCar->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
            ui->layoutDroit->addLayout(this->formCar->boxPrincipale);
            break;
        case MULTISEG :
            this->formMultiSeg->affiche(2);
            QObject::connect(this->formMultiSeg->btnValide, SIGNAL(clicked()), this,
SLOT(nouveauObjet()));
            QObject::connect(this->formMultiSeg->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleNouveauObjet()));
            QObject::connect(this->formMultiSeg->btnCTrait, SIGNAL(clicked()), this, SLOT(couleurT()));
            ui->layoutDroit->addLayout(this->formMultiSeg->boxPrincipale);
            break;
        default:
            break;
      }
  }
//
*****************************************************************************************************************
*****************************************************************//
//                                        Déclaration des slots                                                  //
//
*****************************************************************************************************************
*****************************************************************//
  //*************************** SLOT MENUS ***********************************
  void fenetrePrincipale::on_actionImprimer_triggered()
  {
    QPrinter printer(QPrinter::HighResolution);
    QPrintDialog *dialog = new QPrintDialog(&printer, this);
    printer.setPaperSize(QPrinter::A4);
    dialog->setWindowTitle(tr("Impression du document"));
    dialog->addEnabledOption(QAbstractPrintDialog::PrintSelection);
```

```
    if (dialog->exec() != QDialog::Accepted)
    {
      return;
    }
    else
    {
       QPainter painter(&printer);
       ui->view->render(&painter);
    }
}

void fenetrePrincipale::on_actionExportation_triggered()
{
    QImage img(1024,768,QImage::Format_ARGB32_Premultiplied);
    QPainter p(&img);
    ui->view->render(&p);
    p.end();
    QString lienFichier = QFileDialog::getSaveFileName(this, "Exportation Dessin", QString());
    img.save(lienFichier + ".png");
}

void fenetrePrincipale::on_actionQuitter_3_triggered()
{
    this->close();
}

void fenetrePrincipale::on_actionPlein_cran_triggered()
{
    this->setWindowState(this->windowState() ^ Qt::WindowFullScreen);
}

void fenetrePrincipale::on_actionTutoriels_triggered()
{
    QMessageBox::about(this, "Info","Cette fonctionnalité sera développée prochainement\n");
}

void fenetrePrincipale::on_actionD_veloppeur_triggered()
{
    QMessageBox::about(this, "Développeurs",
            "Cette application a été développé par :\n" \
            "Matthieu Clin\n" \
            "Quentin Vecchio\n");
}

void fenetrePrincipale::on_actionZoom_triggered()
{
    ui->view->scale(2,2);
}

void fenetrePrincipale::on_actionZoom_2_triggered()
{
    ui->view->scale(0.5,0.5);
}

void fenetrePrincipale::on_actionVue_ensemble_triggered()
{
```

```
   ui->view->scale(0.5,0.5);
}

void fenetrePrincipale::on_actionCalculette_triggered()
{
    QMessageBox::about(this, "Info","Cette fonctionnalité sera développée prochainement\n");
}

void fenetrePrincipale::on_actionMath_matique_triggered()
{
   QMessageBox::about(this, "Info","Cette fonctionnalité sera développée prochainement\n");
}

void fenetrePrincipale::on_actionPr_c_dent_triggered()
{
   QMessageBox::about(this, "Info","Cette fonctionnalité sera développée prochainement\n");
}

void fenetrePrincipale::on_actionSuivant_triggered()
{
    QMessageBox::about(this, "Info","Cette fonctionnalité sera développée prochainement\n");
}

void fenetrePrincipale::on_actionTranslation_3_triggered()
{
   //Création du box
      this->boxTrans = new QHBoxLayout();
   //Création des labels
      this->labelT = new QLabel("Translation Dessin");
      this->labelX = new QLabel("x :");
      this->labelY = new QLabel("y :");
   //Création des spinbox
      this->spinX = new QSpinBox();
      this->spinY = new QSpinBox();
      this->spinTps = new QSpinBox();
      this->spinX->setMaximum(999);
      this->spinY->setMaximum(999);
      this->spinX->setMinimum(-999);
      this->spinY->setMinimum(-999);
   //Création des boutons
      this->btnValide = new QPushButton();
      this->btnAnnule = new QPushButton();
      this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
      this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
      QObject::connect(this->btnValide,SIGNAL(clicked()),this,SLOT(valideTrans()));
      QObject::connect(this->btnAnnule,SIGNAL(clicked()),this,SLOT(annuleTrans()));
   //Ajout des widgets dans le box
      this->boxTrans->addWidget(this->labelT);
      this->boxTrans->addWidget(this->labelX);
      this->boxTrans->addWidget(this->spinX);
      this->boxTrans->addWidget(this->labelY);
      this->boxTrans->addWidget(this->spinY);
      this->boxTrans->addWidget(this->btnValide);
      this->boxTrans->addWidget(this->btnAnnule);
   //Ajout du tout dans la fenetre
      ui->layoutAnim->addLayout(this->boxTrans);
```

```
}

void fenetrePrincipale::valideTrans()
{
   Anim a;
   a.x = this->spinX->value();
   a.y = this->spinY->value();
   a.tps = this->spinTps->value();
   this->pileAnim.push_back(a);
   this->btnAnnule->click();
}

void fenetrePrincipale::annuleTrans()
{
   delete this->boxTrans;
   delete this->labelT;
   delete this->labelX;
   delete this->labelY;
   delete this->labelTps;
   delete this->spinX;
   delete this->spinY;
   delete this->spinTps;
   delete this->btnAnnule;
   delete this->btnValide;
}

void fenetrePrincipale::on_actionRotation_3_triggered()
{
    //Création du box
       this->boxRot = new QHBoxLayout();
    //Création des labels
       this->labelR = new QLabel("Rotation Dessin");
       this->labelAngle = new QLabel("Angle :");
    //Création des spinbox
       this->spinX = new QSpinBox();
       this->spinX->setMaximum(360);
       this->spinX->setMinimum(-360);
    //Création des boutons
       this->btnValide = new QPushButton();
       this->btnAnnule = new QPushButton();
       this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
       this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
       QObject::connect(this->btnValide,SIGNAL(clicked()),this,SLOT(valideRot()));
       QObject::connect(this->btnAnnule,SIGNAL(clicked()),this,SLOT(annuleRot()));
    //Ajout des widgets dans le box
       this->boxRot->addWidget(this->labelR);
       this->boxRot->addWidget(this->labelAngle);
       this->boxRot->addWidget(this->spinX);
       this->boxRot->addWidget(this->btnValide);
       this->boxRot->addWidget(this->btnAnnule);
    //Ajout du tout dans la fenetre
       ui->layoutAnim->addLayout(this->boxRot);
}

void fenetrePrincipale::valideRot()
{
```

```
    int x = this->spinX->value();
    int i,y;
    this->btnAnnule->click();
    for(i=0;i<this->dessin.size();i++)
    {
        QDessin des;
        QFigure *fig = new QFigure();
        des = this->dessin[i];
        fig = des.f;
        for(y=0;y<fig->item.size();y++)
        {
            QGraphicsItem *item = fig->item[y];
            item->setTransformOriginPoint(fig->oX,fig->oY);
            item->setRotation(x);
        }
    }
}

void fenetrePrincipale::annuleRot()
{
    delete this->boxRot;
    delete this->labelR;
    delete this->labelAngle;
    delete this->spinX;
    delete this->btnAnnule;
    delete this->btnValide;
}
void fenetrePrincipale::on_actionLancer_Animation_2_triggered()
{
    int i,z,w;
    for(i=0;i<this->dessin.size();i++)
        {
            QDessin des;
            QFigure *fig = new QFigure();
            des = this->dessin[i];
            fig = des.f;
            if(fig->estAnim == false)
            {
                for(w=0;w<this->pileAnim.size();w++)
                {
                    Anim a = this->pileAnim[w];
                    for(z=0;z<fig->item.size();z++)
                    {
                        QGraphicsItem *item = fig->item[z];
                        item->moveBy(a.x,a.y);
                    }
                }
            }
            else
            {
                for(w = 0;w<fig->pileAnim.size();w++)
                {
                    Anim a = fig->pileAnim[w];
                    for(z=0;z<fig->item.size();z++)
                    {
                        QGraphicsItem *item = fig->item[z];
```

```
                item->moveBy(a.x,a.y);
              }
            }
          }
        }
    }

    void fenetrePrincipale::on_actionArreter_Animation_triggered()
    {
        int i, limit = this->pileAnim.size();
        for(i=0;i<limit;i++)
        {
            this->pileAnim.pop_back();
        }
    }

    void fenetrePrincipale::on_actionEnregistrer_triggered()
    {
        QString lienFichier = QFileDialog::getSaveFileName(this, "Enregistrer un Dessin", QString(), "Dessin2D
(*.xml)");
        *(this->lienSauvegarde) = lienFichier;
        this->ecritureDOM();
    }

    void fenetrePrincipale::on_actionOuvrir_triggered()
    {
        QString lienFichier = QFileDialog::getOpenFileName(this, "Ouvrir un Dessin", QString(), "Dessin2D
(*.xml)");
        *(this->lienSauvegarde) = lienFichier;
        lectureDOM();
    }
    //*************************** SLOT BTNS DESSINER **********************************
    void fenetrePrincipale::on_btnSegment_clicked()
    {
        this->obj = SEGMENT;
        afficheFormNvObj();
    }
    void fenetrePrincipale::on_btnCercle_clicked()
    {
        this->obj = CERCLE;
        afficheFormNvObj();
    }
    void fenetrePrincipale::on_btnEllipse_clicked()
    {
        this->obj = ELLIPSE;
        afficheFormNvObj();
    }
    void fenetrePrincipale::on_btnCarre_clicked()
    {
        this->obj = CARRE;
        afficheFormNvObj();
    }
    void fenetrePrincipale::on_btnMultiSegment_clicked()
    {
        this->obj = MULTISEG;
        afficheFormNvObj();
```

```
}
void fenetrePrincipale::on_btnTriangle_clicked()
{
   this->obj = TRIANGLE;
   afficheFormNvObj();
}
//*************************** AUTRES SLOTS ***********************************
void fenetrePrincipale::annuleNouveauObjet()
{
   ui->btnCarre->setEnabled(true);
   ui->btnCercle->setEnabled(true);
   ui->btnEllipse->setEnabled(true);
   ui->btnMultiSegment->setEnabled(true);
   ui->btnSegment->setEnabled(true);
   ui->btnTriangle->setEnabled(true);
   switch (this->obj)
   {
      case SEGMENT:
         this->formSeg->cache();
         break;
      case CARRE:
         this->formCar->cache();
         break;
      case TRIANGLE:
         this->formTri->cache();
         break;
      case CERCLE :
         this->formCer->cache();
         break;
      case ELLIPSE :
         this->formEli->cache();
         break;
      case MULTISEG :
         this->formMultiSeg->cache();
         break;
      default:
         break;
   }
}
void fenetrePrincipale::nouveauObjet()
{
   ui->btnCarre->setEnabled(true);
   ui->btnCercle->setEnabled(true);
   ui->btnEllipse->setEnabled(true);
   ui->btnMultiSegment->setEnabled(true);
   ui->btnSegment->setEnabled(true);
   ui->btnTriangle->setEnabled(true);
   switch (this->obj)
   {
      case SEGMENT:
         this->traceSeg();
         break;
      case CARRE:
        this->traceRec();
         break;
      case TRIANGLE :
```

```
            this->traceTri();
            break;
        case CERCLE :
            this->traceCercle();
            break;
        case ELLIPSE :
            this->traceElippse();
            break;
        case MULTISEG :
            this->traceMultiSeg();
            break;
        default:
            break;
      }
    }

    void fenetrePrincipale::rafraichir()
    {

      QMessageBox::about(this, "Info","Rafraichir\n");
    }

    void fenetrePrincipale::suppression()
    {
      QMessageBox::about(this, "Info","Supprime");
    }

    void fenetrePrincipale::couleurF()
    {
      this->colorF = QColorDialog::getColor("Couleur du fond");
    }

    void fenetrePrincipale::couleurT()
    {
      this->colorT = QColorDialog::getColor("Couleur du trait");
    }

//
**********************************************************************************************************
**********************************************************************//
//                                        Déclaration des
méthodes                                                    //
//
**********************************************************************************************************
**********************************************************************//
    //
**********************************************************************************************************
**********************************************//
    //                                        Déclaration des méthodes ouverture                                    //
    //
**********************************************************************************************************
**********************************************//
     void fenetrePrincipale::lectureDOM()
    {
      QFile file(*(this->lienSauvegarde));
      QDomDocument doc;
```

```
  if(!file.open(QIODevice::ReadOnly))
    return;
  if (!doc.setContent(&file))
  {
    *(this->doc) = doc;
    lireDOM();
    file.close(); // établit le document XML à partir des données du fichier (hiérarchie, etc.)
    return;
  }
  *(this->doc) = doc;
  lireDOM();
  file.close();
}

void fenetrePrincipale::lireDOM()
{
    QDomNodeList tab;
    QDomNodeList tabP;
    QDomNodeList tabCT;
    QDomNodeList tabCF;
    QDomNodeList tabR;
    QDomElement objet;
    QDomElement point;
    QDomElement rayon;
    QDomElement couleurF;
    QDomElement couleurT;
    QDomNode n;
    QDomElement racine = doc->documentElement(); // renvoie la balise racine
    QDomNode noeud = racine.firstChild(); // renvoie la première balise « mesure »
    while(!noeud.isNull())
    {
      QFigure *f = new QFigure();
      // convertit le nœud en élément pour utiliser les
      // méthodes tagName() et attribute()
      objet = noeud.toElement();
      // vérifie la présence de la balise
      QDessin des;
      //Nom de la figure
        tab = objet.childNodes();//Stocke les premiers rang
        n = tab.item(0);
        f->nom = QString(n.firstChild().toText().data());
      //Options
        f->etat = true;
        QDomNode noeud1 = objet.firstChild();
        noeud1 = noeud1.nextSibling();
        point =  noeud1.toElement();
        tabP = point.childNodes();
        int x1, y1, x2, y2, r;
      if(objet.tagName() != "MultiSeg")
      {
        //Point 1
          n = tabP.item(0);
          x1 = QString(n.firstChild().toText().data()).toInt();
          n = tabP.item(1);
          y1 = QString(n.firstChild().toText().data()).toInt();
          Point2D p1(x1,y1);
```

```
        f->points.push_back(p1);
    if(objet.tagName() != "Cercle")
    {
        //Point 2
        noeud1 = noeud1.nextSibling();
        point = noeud1.toElement();
        tabP = point.childNodes();
        n = tabP.item(0);
        x2 = QString(n.firstChild().toText().data()).toInt();
        n = tabP.item(1);
        y2 = QString(n.firstChild().toText().data()).toInt();
        Point2D p2(x2,y2);
        f->points.push_back(p2);
    }
    if(objet.tagName() == "Cercle")
    {
        //Rayon
        n = tab.item(2);
        r = QString(n.firstChild().toText().data()).toInt();
    }
    if(objet.tagName() == "Elippse")
    {
        //Rayon
        n = tab.item(3);
        r = QString(n.firstChild().toText().data()).toInt();
    }
}
if(objet.tagName() == "Segment")
{
        f->o = SEGMENT;
        //Definiton de l'objet segment
        Segment s(Point2D(x1,y1),Point2D(x2,y2));
        //Ajout du segment dans l'objet
        f->f = s;
        //Tracage de la figure
        //Gestion Couleurs
            noeud1 = noeud1.nextSibling();
            couleurT = noeud1.toElement();
            tabCT = couleurT.childNodes();
            n = tabCT.item(0);
            f->cR = QString(n.firstChild().toText().data()).toInt();
            n = tabCT.item(1);
            f->cG = QString(n.firstChild().toText().data()).toInt();
            n = tabCT.item(2);
            f->cB = QString(n.firstChild().toText().data()).toInt();
            this->colorT.setRgb(f->cR,f->cG,f->cB);
        QGraphicsItem *item = ui->view->scene->addLine(x1,y1,x2,y2,QPen(this->colorT));
         f->item.push_back(item);

}
else if(objet.tagName() == "Carre")
{
        f->o = CARRE;
        //Point 3
        noeud1 = noeud1.nextSibling();
        point = noeud1.toElement();
```

```
                tabP = point.childNodes();
                n = tabP.item(0);
                int x3 = QString(n.firstChild().toText().data()).toInt();
                n = tabP.item(1);
                int y3 = QString(n.firstChild().toText().data()).toInt();
                Point2D p3(x3,y3);
                f->points.push_back(p3);
                //Point 4
                noeud1 = noeud1.nextSibling();
                point = noeud1.toElement();
                tabP = point.childNodes();
                n = tabP.item(0);
                int x4 = QString(n.firstChild().toText().data()).toInt();
                n = tabP.item(1);
                int y4 = QString(n.firstChild().toText().data()).toInt();
                Point2D p4(x4,y4);
                f->points.push_back(p4);
                //Definiton de l'objet segment
                Rectangle r(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3),Point2D(x4,y4));
                //Ajout du segment dans l'objet
                f->f = r;
                //Tracage de la figure
                //Gestion Couleurs
                noeud1 = noeud1.nextSibling();
                couleurT = noeud1.toElement();
                tabCT = couleurT.childNodes();
                n = tabCT.item(0);
                f->cR = QString(n.firstChild().toText().data()).toInt();
                n = tabCT.item(1);
                f->cG = QString(n.firstChild().toText().data()).toInt();
                n = tabCT.item(2);
                f->cB = QString(n.firstChild().toText().data()).toInt();
                this->colorT.setRgb(f->cR,f->cG,f->cB);
               //Gestion Couleurs Fond
                noeud1 = noeud1.nextSibling();
                couleurF = noeud1.toElement();
                tabCF = couleurF.childNodes();
                n = tabCF.item(0);
                f->cfR = QString(n.firstChild().toText().data()).toInt();
                n = tabCF.item(1);
                f->cfG = QString(n.firstChild().toText().data()).toInt();
                n = tabCF.item(2);
                f->cfB = QString(n.firstChild().toText().data()).toInt();
                this->colorF.setRgb(f->cfR,f->cfG,f->cfB);
                QPalette cpalette = palette();
                QBrush brush(this->colorF);
                brush.setStyle(Qt::SolidPattern);
                cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
                QGraphicsItem *item = ui->view->scene->addRect(x1,y1,x2-x1,y3-y1,QPen(this-
>colorT),brush);
                f->item.push_back(item);
        }
        else if(objet.tagName() == "MultiSeg")
        {

        }
```

```
else if(objet.tagName() == "Triangle")
{
   f->o = TRIANGLE;
   //Point 3
   noeud1 = noeud1.nextSibling();
   point = noeud1.toElement();
   tabP = point.childNodes();
   n = tabP.item(0);
   int x3 = QString(n.firstChild().toText().data()).toInt();
   n = tabP.item(1);
   int y3 = QString(n.firstChild().toText().data()).toInt();
   Point2D p3(x3,y3);
   f->points.push_back(p3);
   //Definiton de l'objet segment
   Triangle t(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3));
   //Ajout du segment dans l'objet
   f->f = t;
   //Tracage de la figure
   //Gestion Couleurs
   noeud1 = noeud1.nextSibling();
   couleurT = noeud1.toElement();
   tabCT = couleurT.childNodes();
   n = tabCT.item(0);
   f->cR = QString(n.firstChild().toText().data()).toInt();
   n = tabCT.item(1);
   f->cG = QString(n.firstChild().toText().data()).toInt();
   n = tabCT.item(2);
   f->cB = QString(n.firstChild().toText().data()).toInt();
   this->colorT.setRgb(f->cR,f->cG,f->cB);
   QGraphicsItem *item = ui->view->scene->addLine(x1,y1,x2,y2,QPen(this->colorT));
   f->item.push_back(item);
   item = ui->view->scene->addLine(x2,y2,x3,y3,QPen(this->colorT));
   f->item.push_back(item);
   item = ui->view->scene->addLine(x1,y1,x3,y3,QPen(this->colorT));
   f->item.push_back(item);
}
else if(objet.tagName() == "Cercle")
{
   f->o = CERCLE;
   //Definiton de l'objet segment
   Cercle c(Point2D(x1,y1),r);
   //Ajout du segment dans l'objet
   f->f = c;
   //Gestion Couleurs
   noeud1 = noeud1.nextSibling();
   noeud1 = noeud1.nextSibling();
   couleurT = noeud1.toElement();
   tabCT = couleurT.childNodes();
   n = tabCT.item(0);
   f->cR = QString(n.firstChild().toText().data()).toInt();
   n = tabCT.item(1);
   f->cG = QString(n.firstChild().toText().data()).toInt();
   n = tabCT.item(2);
   f->cB = QString(n.firstChild().toText().data()).toInt();
   this->colorT.setRgb(f->cR,f->cG,f->cB);
  //Gestion Couleurs Fond
```

```
    noeud1 = noeud1.nextSibling();
    couleurF = noeud1.toElement();
    tabCF = couleurF.childNodes();
    n = tabCF.item(0);
    f->cfR = QString(n.firstChild().toText().data()).toInt();
    n = tabCF.item(1);
    f->cfG = QString(n.firstChild().toText().data()).toInt();
    n = tabCF.item(2);
    f->cfB = QString(n.firstChild().toText().data()).toInt();
    this->colorF.setRgb(f->cfR,f->cfG,f->cfB);
    QPalette cpalette = palette();
    QBrush brush(this->colorF);
    brush.setStyle(Qt::SolidPattern);
    cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
    QGraphicsItem *item = ui->view->scene->addEllipse(x1,y1,r,r,QPen(this->colorT),brush);
    f->item.push_back(item);
}
else if(objet.tagName() == "Elippse")
{
    f->o = ELLIPSE;
    QMessageBox::about(this,"Info",QString::number(r));
    //Definiton de l'objet segment
    Elippse e(Point2D(x1,y1),Point2D(x2,y2),r);
    //Ajout du segment dans l'objet
    f->f = e;
    //Gestion Couleurs
    noeud1 = noeud1.nextSibling();
    noeud1 = noeud1.nextSibling();
    couleurT = noeud1.toElement();
    tabCT = couleurT.childNodes();
    n = tabCT.item(0);
    f->cR = QString(n.firstChild().toText().data()).toInt();
    n = tabCT.item(1);
    f->cG = QString(n.firstChild().toText().data()).toInt();
    n = tabCT.item(2);
    f->cB = QString(n.firstChild().toText().data()).toInt();
    this->colorT.setRgb(f->cR,f->cG,f->cB);
   //Gestion Couleurs Fond
    noeud1 = noeud1.nextSibling();
    noeud1 = noeud1.nextSibling();
    couleurF = noeud1.toElement();
    tabCF = couleurF.childNodes();
    n = tabCF.item(0);
    f->cfR = QString(n.firstChild().toText().data()).toInt();
    n = tabCF.item(1);
    f->cfG = QString(n.firstChild().toText().data()).toInt();
    n = tabCF.item(2);
    f->cfB = QString(n.firstChild().toText().data()).toInt();
    this->colorF.setRgb(f->cfR,f->cfG,f->cfB);
    QPalette cpalette = palette();
    QBrush brush(this->colorF);
    brush.setStyle(Qt::SolidPattern);
    cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
    QGraphicsItem *item = ui->view->scene->addEllipse(x2-x1,y2,r,2*r,QPen(this->colorT),brush);
    f->item.push_back(item);
}
```

```
            //Ajout du dessin
            des.f = f;
            //Initialisation d'un nouveau panel de gestion d'objet
            presentationObjet *pre = new presentationObjet();
            pre->setFig(f);
            pre->setNom(f->nom);
            ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
            //Ajout du dessin
            this->dessin.push_back(des);
            noeud = noeud.nextSibling(); // passe à l'objet" suivante
        }
    }
    //
**********************************************************************************************************************
*********************************************//
    //                                    Déclaration des méthodes sauvegarde                                    //
    //
**********************************************************************************************************************
*********************************************//
    void fenetrePrincipale::ecritureDOM()
    {
        this->doc = new QDomDocument();
        this->file = new QFile();
        this->out = new QTextStream();
        this->dessinX = new QDomElement(doc->createElement("dessin")); // création de la balise "dessin"
        doc->appendChild(*(this->dessinX)); // filiation de la balise "mesures"
        file->setFileName(*(this->lienSauvegarde));
        if (!file->open(QIODevice::WriteOnly)) // ouverture du fichier de sauvegarde
            return; // en écriture
        out->setDevice(file); // association du flux au fichier
        QString *nomObjet = new QString();
        for(int i=0;i<this->dessin.size();i++)
        {
            QDessin des = this->dessin[i];
            if(des.f->etat == true)
            {
                switch (des.f->o)
                {
                    case SEGMENT:
                        *nomObjet = "Segment";
                        break;
                    case CARRE:
                       *nomObjet = "Carre";
                        break;
                    case TRIANGLE :
                        *nomObjet = "Triangle";
                        break;
                    case CERCLE :
                        *nomObjet = "Cercle";
                        break;
                    case ELLIPSE :
                        *nomObjet = "Elippse";
                        break;
                    case MULTISEG :
                         *nomObjet = "MultiSeg";
                        break;
```

```
            default:
               break;
         }
         this->ajoutDOM(nomObjet,des.f,i);
      }
   }
   this->sauvegardeDOM();
}

void fenetrePrincipale::ajoutDOM(QString *nomObjet,QFigure *fig, int id)
{
   //création de la balise "nomObjet"
      QDomElement objet = doc->createElement(*(nomObjet));
      this->dessinX->appendChild(objet); // filiation de la balise "nomObjet"
      objet.setAttribute("id",id); // création de l'attribut "numero"
      QString *x = new QString();
      QString *y = new QString();
      QString *r = new QString();
      Point2D p;
      QDomElement baliseNom = doc->createElement("nom");
      objet.appendChild(baliseNom);
      QDomText nom = doc->createTextNode(fig->nom);
      baliseNom.appendChild(nom);
      for(int i=0;i<fig->points.size();i++)
      {
         p = fig->points[i];
         //création de la balise "point2D"
         QDomElement point = doc->createElement("point2D");
         point.setAttribute("id",i);
         objet.appendChild(point);
         //Creation de la balise X
         QDomElement baliseX = doc->createElement("X");
         point.appendChild(baliseX);
         x->setNum((int)p.getX());
         y->setNum((int)p.getY());
         QDomText coorX = doc->createTextNode(*(x));
         baliseX.appendChild(coorX);
         // création de la balise Y
         QDomElement baliseY = doc->createElement("Y");
         point.appendChild(baliseY);
         QDomText coorY = doc->createTextNode(*(y));
         baliseY.appendChild(coorY);
      }
      if(fig->o == CERCLE || fig->o == ELLIPSE)
      {
         //Ajout du rayon
         QDomElement baliseR = doc->createElement("rayon");
         objet.appendChild(baliseR); // filiation de la balise "point"
         r->setNum((int)fig->r);
         QDomText rayon = doc->createTextNode(*(r)); //création de la donnée f1
         baliseR.appendChild(rayon); // filiation du nœud "f1"
      }
      //Ajout de la couleur du trait
      QDomElement baliseCouleurTrait = doc->createElement("couleurT");
      objet.appendChild(baliseCouleurTrait);
      QDomElement baliseCouleurTraitR = doc->createElement("R");
```

```
        baliseCouleurTrait.appendChild(baliseCouleurTraitR);
        QDomElement baliseCouleurTraitG = doc->createElement("G");
        baliseCouleurTrait.appendChild(baliseCouleurTraitG);
        QDomElement baliseCouleurTraitB = doc->createElement("B");
        baliseCouleurTrait.appendChild(baliseCouleurTraitB);
        QDomText cR = doc->createTextNode(QString::number(fig->cR));
        baliseCouleurTraitR.appendChild(cR);
        QDomText cG = doc->createTextNode(QString::number(fig->cG));
        baliseCouleurTraitG.appendChild(cG);
        QDomText cB = doc->createTextNode(QString::number(fig->cB));
        baliseCouleurTraitB.appendChild(cB);
        if(fig->o == CARRE || fig->o == CERCLE || fig->o == CERCLE)
        {
           //Ajout de la couleur du fond
           QDomElement baliseCouleurFond = doc->createElement("couleurF");
           objet.appendChild(baliseCouleurFond);
           QDomElement baliseCouleurFondR = doc->createElement("R");
           baliseCouleurFond.appendChild(baliseCouleurFondR);
           QDomElement baliseCouleurFondG = doc->createElement("G");
           baliseCouleurFond.appendChild(baliseCouleurFondG);
           QDomElement baliseCouleurFondB = doc->createElement("B");
           baliseCouleurFond.appendChild(baliseCouleurFondB);
           QDomText cR = doc->createTextNode(QString::number(fig->cfR));
           baliseCouleurFondR.appendChild(cR);
           QDomText cG = doc->createTextNode(QString::number(fig->cfG));
           baliseCouleurFondG.appendChild(cG);
           QDomText cB = doc->createTextNode(QString::number(fig->cfB));
           baliseCouleurFondB.appendChild(cB);
        }
    }

    void fenetrePrincipale::sauvegardeDOM()
    {
        // insertion en début de document de <?xml version="1.0" ?>
        QDomNode noeud = doc->createProcessingInstruction("xml","version=\"1.0\"");
        doc->insertBefore(noeud,doc->firstChild());
        // sauvegarde dans le flux (deux espaces de décalage dans l'arborescence)
        doc->save(*(out),2);
        file->close();
    }

    //
*****************************************************************************************************************
*********************************************//
    //                                    Déclaration des méthodes de tracage                                    //
    //
*****************************************************************************************************************
*********************************************//
    void fenetrePrincipale::traceRec()
    {
      //Création de la structure figure
         QFigure *fig = new QFigure();
      //Ajout de la scene dans la figure
         fig->estAnim = false;
      //Création d'un dessin
         QDessin d;
```

```
    //Initialisation d'un nouveau panel de gestion d'objet
        presentationObjet *pre = new presentationObjet();
    //On récupère les valeurs
        int x1 = this->formCar->X1();
        int y1 = this->formCar->Y1();
        int x2 = this->formCar->X2();
        int y2 = this->formCar->Y2();
        int x3 = this->formCar->X3();
        int y3 = this->formCar->Y3();
        int x4 = this->formCar->X4();
        int y4 = this->formCar->Y4();
        fig->nom = this->formCar->lineNom->text();
    //Création de la figure rectangle
        fig->points.push_back(Point2D(x1,y1));
        fig->points.push_back(Point2D(x2,y2));
        fig->points.push_back(Point2D(x3,y3));
        fig->points.push_back(Point2D(x4,y4));
        fig->etat = true;
        fig->oX = x1;
        fig->oY = y1;
        this->colorF.getRgb(&fig->cfR,&fig->cfG,&fig->cfB);
        this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
        Rectangle r(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3),Point2D(x4,y4));
    //on met le rectangle dans la structure figure
        fig->f = r;
    //Définiton de la couleur du fond de l'objet
        QPalette cpalette = palette();
        QBrush brush(this->colorF);
        brush.setStyle(Qt::SolidPattern);
        cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
    //Traçage de la figure dans le QGraphicsScene
        QGraphicsItem *item = ui->view->scene->addRect(x1,y1,x2-x1,y3-y1,QPen(this->colorT),brush);
    //Ajout du pointeur sur le tracé de la figure
        fig->item.push_back(item);
    //Ajout du type de figure
        fig->o = CARRE;
    //Ajout de la figure au dessin
        d.f = fig;
        d.id = this->dessin.size() + 1;
    //ajout du type dessin dans le dessin technique
        this->dessin.push_back(d);
    //Affichage et configuration du nv panel
        pre->setFig(fig);
        pre->setNom(fig->nom);
        ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
    //On supprime le formulaire
        this->formCar->cache();
}
void fenetrePrincipale::traceSeg()
{
    //Création de la structure figure
        QFigure *fig = new QFigure();
    //Création d'un dessin
        QDessin d;
        fig->estAnim = false;
    //Initialisation d'un nouveau panel de gestion d'objet
```

```
    presentationObjet *pre = new presentationObjet();
  //On récupère les valeurs
    int x1 = this->formSeg->X1();
    int y1 = this->formSeg->Y1();
    int x2 = this->formSeg->X2();
    int y2 = this->formSeg->Y2();
    fig->nom = this->formSeg->lineNom->text();
  //Création de la figure segment
    fig->points.push_back(Point2D(x1,y1));
    fig->points.push_back(Point2D(x2,y2));
    fig->etat = true;
    fig->oX = x1;
    fig->oY = y1;
    this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
    Segment s(Point2D(x1,y1),Point2D(x2,y2));
  //on met le segment dans la structure figure
    fig->f = s;
  //Traçage de la figure dans le QGraphicsScene
    QGraphicsItem *item = ui->view->scene->addLine(x1,y1,x2,y2,QPen(this->colorT));
  //Ajout du pointeur sur le tracé de la figure
    fig->item.push_back(item);
  //Ajout du type de figure
    fig->o = SEGMENT;
  //Ajout de la figure au dessin
    d.f = fig;
    d.id = this->dessin.size() + 1;
  //ajout du type dessin dans le dessin technique
    this->dessin.push_back(d);
  //Affichage et configuration du nv panel
    pre->setFig(fig);
    pre->setNom(fig->nom);
    ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
  //On supprime le formulaire
    this->formSeg->cache();
}

void fenetrePrincipale::traceCercle()
{
  //Création de la structure figure
    QFigure *fig= new QFigure();
  //Création d'un dessin
    QDessin d;
    fig->estAnim = false;
  //Initialisation d'un nouveau panel de gestion d'objet
    presentationObjet *pre = new presentationObjet();
  //On récupère les valeurs
    int x1 = this->formCer->X1();
    int y1 = this->formCer->Y1();
    int r = this->formCer->ChangeRayon->value();
    fig->nom = this->formCer->lineNom->text();
  //Création de la figure segment
    fig->etat = true;
    fig->points.push_back(Point2D(x1,y1));
    fig->r = r;
    fig->oX = x1;
    fig->oY = y1;
```

```
    this->colorF.getRgb(&fig->cfR,&fig->cfG,&fig->cfB);
    this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
    Cercle c(Point2D(x1,y1),r);
  //on met le segment dans la structure figure
    fig->f = c;
  //Définiton de la couleur du fond de l'objet
    QPalette cpalette = palette();
    QBrush brush(this->colorF);
    brush.setStyle(Qt::SolidPattern);
    cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
  //Traçage de la figure dans le QGraphicsScene
    QGraphicsItem *item = ui->view->scene->addEllipse(x1,y1,r,r,QPen(this->colorT),brush);
  //Ajout du pointeur sur le tracé de la figure
    fig->item.push_back(item);
  //Ajout du type de figure
    fig->o = CERCLE;
  //Ajout de la figure au dessin
    d.f = fig;
    d.id = this->dessin.size() + 1;
  //ajout du type dessin dans le dessin technique
      this->dessin.push_back(d);
  //Affichage et configuration du nv panel
    pre->setFig(fig);
    pre->setNom(fig->nom);
    ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
  //On supprime le formulaire
    this->formCer->cache();
}

void fenetrePrincipale::traceTri()
{
  //Création de la structure figure
    QFigure *fig= new QFigure();
  //Création d'un dessin
    QDessin d;
    fig->estAnim = false;
  //Initialisation d'un nouveau panel de gestion d'objet
    presentationObjet *pre = new presentationObjet();
  //On récupère les valeurs
    int x1 = this->formTri->X1();
    int x2 = this->formTri->X2();
    int y1 = this->formTri->Y1();
    int y2 = this->formTri->Y2();
    int x3 = this->formTri->X3();
    int y3 = this->formTri->Y3();
    fig->nom = this->formTri->lineNom->text();
  //Création de la figure rectangle
    fig->points.push_back(Point2D(x1,y1));
    fig->points.push_back(Point2D(x2,y2));
    fig->points.push_back(Point2D(x3,y3));
    fig->etat = true;
    fig->oX = x1;
    fig->oY = y1;
    this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
    Triangle t(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3));
  //on met le rectangle dans la structure figure
```

```
    fig->f = t;
  //Définiton de la couleur du fond de l'objet
    QPalette cpalette = palette();
    QBrush brush(this->colorF);
    brush.setStyle(Qt::SolidPattern);
    cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
  //Traçage de la figure dans le QGraphicsScene
    QGraphicsItem *item = ui->view->scene->addLine(x1,y1,x2,y2,QPen(this->colorT));
    fig->item.push_back(item);
    item = ui->view->scene->addLine(x2,y2,x3,y3,QPen(this->colorT));
    fig->item.push_back(item);
    item = ui->view->scene->addLine(x1,y1,x3,y3,QPen(this->colorT));
    fig->item.push_back(item);
  //Ajout de la figure au dessin
    d.f = fig;
    d.id = this->dessin.size() + 1;
  //ajout du type dessin dans le dessin technique
    this->dessin.push_back(d);
  //Ajout du type de figure
    fig->o = TRIANGLE;
  //Affichage et configuration du nv panel
    pre->setFig(fig);
    pre->setNom(fig->nom);
    ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
  //On supprime le formulaire
    this->formTri->cache();
}

void fenetrePrincipale::traceElippse()
{
  //Création de la structure figure
    QFigure *fig= new QFigure();
 //Création d'un dessin
    QDessin d;
    fig->estAnim = false;
  //Initialisation d'un nouveau panel de gestion d'objet
    presentationObjet *pre = new presentationObjet();
  //On récupère les valeurs
    int x1 = this->formEli->X1();
    int y1 = this->formEli->Y1();
    int x2 = this->formEli->X2();
    int y2 = this->formEli->Y2();
    fig->oX = x1;
    fig->oY = y1;
    int r = this->formEli->ChangeRayon->value();
    fig->nom = this->formEli->lineNom->text();
  //Création de la figure elippse
    fig->etat = true;
    fig->points.push_back(Point2D(x1,y1));
    fig->points.push_back(Point2D(x2,y2));
    fig->r = r;
    this->colorF.getRgb(&fig->cfR,&fig->cfG,&fig->cfB);
    this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
    Elippse e(Point2D(x1,y1),Point2D(x2,y2),r);
  //on met l'elippse dans la structure figure
    fig->f = e;
```

```
    //Définiton de la couleur du fond de l'objet
      QPalette cpalette = palette();
      QBrush brush(this->colorF);
      brush.setStyle(Qt::SolidPattern);
      cpalette.setBrush(QPalette::Active, QPalette::Window, brush);
    //Traçage de la figure dans le QGraphicsScene
      QGraphicsItem *item = ui->view->scene->addEllipse(x2-x1,y2,r,2*r,QPen(this->colorT),brush);
    //Ajout du pointeur sur le tracé de la figure
      fig->item.push_back(item);
    //Ajout du type de figure
      fig->o = ELLIPSE;
    //Ajout de la figure au dessin
      d.f = fig;
      d.id = this->dessin.size() + 1;
    //ajout du type dessin dans le dessin technique
        this->dessin.push_back(d);
    //Affichage et configuration du nv panel
      pre->setFig(fig);
      pre->setNom(fig->nom);
      ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
    //On supprime le formulaire
      this->formEli->cache();
  }

  void fenetrePrincipale::traceMultiSeg()
  {
    //Création de la structure figure
      QFigure *fig = new QFigure();
    //Création d'un dessin
      QDessin d;
      fig->estAnim = false;
    //Initialisation d'un nouveau panel de gestion d'objet
      presentationObjet *pre = new presentationObjet();
    //On récupère les valeurs
      fig->etat = true;
      this->colorT.getRgb(&fig->cR,&fig->cG,&fig->cB);
      fig->nom = this->formMultiSeg->lineNom->text();
      int i;
      GuiPoint *p1 = this->formMultiSeg->point[0];
      for(i=1;i<this->formMultiSeg->Nombre();i++)
      {
        fig->points.push_back(Point2D(p1->ptx->value(),p1->pty->value()));
        GuiPoint *p2 = this->formMultiSeg->point[i];
        //Traçage de la figure dans le QGraphicsScene
        QGraphicsItem *item = ui->view->scene->addLine(p1->ptx->value(),p1->pty->value(),p2->ptx-
>value(),p2->pty->value(),QPen(this->colorT));
        //Ajout du pointeur sur le tracé de la figure
        fig->item.push_back(item);
        p1 = this->formMultiSeg->point[i];
      }
    //Ajout du type de figure
      fig->o = MULTISEG;
    //Ajout de la figure au dessin
      d.f = fig;
      d.id = this->dessin.size() + 1;
    //ajout du type dessin dans le dessin technique
```

```
        this->dessin.push_back(d);
    //Affichage et configuration du nv panel
        pre->setFig(fig);
        pre->setNom(fig->nom);
        ui->zoneControleObj->addLayout(pre->box,this->dessin.size(),1,1,3);
    //On supprime le formulaire
        this->formMultiSeg->cache();
  }
```

B) Formulaire Carré

1) Header

```
#ifndef FORMCARRE_H
#define FORMCARRE_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"

class formCarre: public QWidget
{
  Q_OBJECT
public:
  formCarre();
  void cache();
  void affiche();
  int X1();
  int X2();
  int Y1();
  int Y2();
  int X3();
  int Y3();
  int X4();
  int Y4();
  QPushButton *btnValide;
  QPushButton *btnAnnule;
  QVBoxLayout *boxPrincipale;
  QLineEdit *lineNom;
  QPushButton *btnCTrait;
  QPushButton *btnCFond;
  GuiPoint *pointA;
  GuiPoint *pointB;
  GuiPoint *pointC;
  GuiPoint *pointD;
private:
  QHBoxLayout * nomLayout;
  QHBoxLayout * CtraitLayout;
  QHBoxLayout * CFondLayout;
```

```
    QLabel * nomLabel;
    QLabel * nomCTrait;
    QLabel * nomCFond;
    QHBoxLayout *btnLayout;
};

#endif // FORMCARRE_H
```

2) Source

```
#include "formcarre.h"

formCarre::formCarre() : QWidget()
{
}

void formCarre::affiche()
{
    //Init Box le principale qui contient le formulaire
        this->boxPrincipale = new QVBoxLayout();
    //Init tous les points
        this->pointA = new GuiPoint();
        this->pointB = new GuiPoint();
        this->pointC = new GuiPoint();
        this->pointD = new GuiPoint();
        this->pointA->labelPoint->setText("Point A :");
        this->pointB->labelPoint->setText("Point B :");
        this->pointC->labelPoint->setText("Point C :");
        this->pointD->labelPoint->setText("Point D :");
        this->pointA->reinit();
        this->pointB->reinit();
        this->pointC->reinit();
        this->pointD->reinit();
    //Init tous les box horizontales
        this->nomLayout = new QHBoxLayout();
        this->CFondLayout = new QHBoxLayout();
        this->CtraitLayout = new QHBoxLayout();
        this->btnLayout = new QHBoxLayout();
    //Init le nom
        this->nomLabel = new QLabel("Nom : ");
        this->lineNom = new QLineEdit();
        this->nomLayout->addWidget(this->nomLabel);
        this->nomLayout->addWidget(this->lineNom);
        this->boxPrincipale->addLayout(this->nomLayout);
        this->lineNom->clear();
    //Init les points
        this->boxPrincipale->addLayout(this->pointA->pointLayout);
        this->boxPrincipale->addLayout(this->pointB->pointLayout);
        this->boxPrincipale->addLayout(this->pointC->pointLayout);
        this->boxPrincipale->addLayout(this->pointD->pointLayout);
    //Init la couleur du trait
        this->nomCTrait = new QLabel("Couleur du trait :");
        this->btnCTrait = new QPushButton();
        this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
        this->CtraitLayout->addWidget(this->nomCTrait);
```

```
    this->CtraitLayout->addWidget(this->btnCTrait);
    this->boxPrincipale->addLayout(this->CtraitLayout);
  //Init la couleur du fond
    this->nomCFond = new QLabel("Couleur du fond :");
    this->btnCFond = new QPushButton();
    this->btnCFond->setIcon(QIcon(":/icones/logos/water.png"));
    this->CFondLayout->addWidget(this->nomCFond);
    this->CFondLayout->addWidget(this->btnCFond);
    this->boxPrincipale->addLayout(this->CFondLayout);
  //Btn de validation
    this->btnValide = new QPushButton();
    this->btnAnnule = new QPushButton();
    this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
    this->btnLayout->addWidget(btnValide);
    this->btnLayout->addWidget(btnAnnule);
    this->boxPrincipale->addLayout(btnLayout);
}

void formCarre::cache()
{
  //Suppression points
    this->pointA->detruit();
    this->pointB->detruit();
    this->pointC->detruit();
    this->pointD->detruit();
  //Suppression nom
    delete this->nomLabel;
    delete this->nomLayout;
    delete this->lineNom;
  //Suppresion couleur trait
    delete this->nomCTrait;
    delete this->btnCTrait;
    delete this->CtraitLayout;
  //Suppresion couleur trait
    delete this->nomCFond;
    delete this->btnCFond;
    delete this->CFondLayout;
  //Suppresion btn valide
    delete this->btnLayout;
    delete this->btnValide;
    delete this->btnAnnule;
  //Suppresion de tout le box
    delete this->boxPrincipale;
}

int formCarre::X1()
{
  return this->pointA->ptx->value();
}

int formCarre::X2()
{
  return this->pointB->ptx->value();
}
```

```
int formCarre::Y1()
{
   return this->pointA->pty->value();
}

int formCarre::Y2()
{
   return this->pointB->pty->value();
}

int formCarre::X3()
{
   return this->pointC->ptx->value();
}

int formCarre::Y3()
{
   return this->pointC->pty->value();
}

int formCarre::X4()
{
   return this->pointD->ptx->value();
}

int formCarre::Y4()
{
   return this->pointD->pty->value();
}
```

C) Formulaire Cercle

1) Header

```
#ifndef FORMCERCLE_H
#define FORMCERCLE_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"

class formCercle: public QWidget
{
   Q_OBJECT
public:
   formCercle();
   void cache();
   void affiche();
```

```
    int X1();
    int Y1();
    QPushButton *btnValide;
    QPushButton *btnAnnule;
    QVBoxLayout *boxPrincipale;
    QSpinBox *ChangeRayon;
    QLineEdit *lineNom;
    QPushButton *btnCTrait;
    QPushButton *btnCFond;
    GuiPoint *pointA;
private:
    QHBoxLayout * nomLayout;
    QHBoxLayout * rayonLayout;
    QHBoxLayout * CtraitLayout;
    QHBoxLayout * CFondLayout;
    QLabel * nomLabel;
    QLabel * rayonLabel;
    QLabel * nomCTrait;
    QLabel * nomCFond;
    QHBoxLayout *btnLayout;
};

#endif // FORMCERCLE_H
```

2) Source

```
#include "formcercle.h"

formCercle::formCercle() : QWidget()
{
}

void formCercle::affiche()
{
    //Init Box le principale qui contient le formulaire
        this->boxPrincipale = new QVBoxLayout();
    //Init tous les points
        this->pointA = new GuiPoint();
        this->pointA->labelPoint->setText("Point A :");
        this->pointA->reinit();
    //Init tous les box horizontales
        this->nomLayout = new QHBoxLayout();
        this->CFondLayout = new QHBoxLayout();
        this->CtraitLayout = new QHBoxLayout();
        this->btnLayout = new QHBoxLayout();
    //Init le nom
        this->nomLabel = new QLabel("Nom : ");
        this->lineNom = new QLineEdit();
        this->nomLayout->addWidget(this->nomLabel);
        this->nomLayout->addWidget(this->lineNom);
        this->boxPrincipale->addLayout(this->nomLayout);
        this->lineNom->clear();
    //Init les points
        this->boxPrincipale->addLayout(this->pointA->pointLayout);
    //init le rayon
```

```
    this->rayonLabel = new QLabel("Rayon :");
    this->rayonLayout = new QHBoxLayout();
    this->ChangeRayon = new QSpinBox();
    this->rayonLayout->addWidget(this->rayonLabel);
    this->rayonLayout->addWidget(this->ChangeRayon);
    this->boxPrincipale->addLayout(this->rayonLayout);
  //Init la couleur du trait
    this->nomCTrait = new QLabel("Couleur du trait :");
    this->btnCTrait = new QPushButton();
    this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
    this->CtraitLayout->addWidget(this->nomCTrait);
    this->CtraitLayout->addWidget(this->btnCTrait);
    this->boxPrincipale->addLayout(this->CtraitLayout);
  //Init la couleur du fond
    this->nomCFond = new QLabel("Couleur du fond :");
    this->btnCFond = new QPushButton();
    this->btnCFond->setIcon(QIcon(":/icones/logos/water.png"));
    this->CFondLayout->addWidget(this->nomCFond);
    this->CFondLayout->addWidget(this->btnCFond);
    this->boxPrincipale->addLayout(this->CFondLayout);
  //Btn de validation
    this->btnValide = new QPushButton();
    this->btnAnnule = new QPushButton();
    this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
    this->btnLayout->addWidget(btnValide);
    this->btnLayout->addWidget(btnAnnule);
    this->boxPrincipale->addLayout(btnLayout);
}

void formCercle::cache()
{
  //Suppression points
    this->pointA->detruit();
  //Suppression nom
    delete this->nomLabel;
    delete this->nomLayout;
    delete this->lineNom;
  //Suppression du rayon
    delete this->rayonLabel;
    delete this->ChangeRayon;
    delete this->rayonLayout;
  //Suppresion couleur trait
    delete this->nomCTrait;
    delete this->btnCTrait;
    delete this->CtraitLayout;
  //Suppresion couleur trait
    delete this->nomCFond;
    delete this->btnCFond;
    delete this->CFondLayout;
  //Suppresion btn valide
    delete this->btnLayout;
    delete this->btnValide;
    delete this->btnAnnule;
  //Suppresion de tout le box
    delete this->boxPrincipale;
```

```
}

int formCercle::X1()
{
   return this->pointA->ptx->value();
}

int formCercle::Y1()
{
   return this->pointA->pty->value();
}
```

D) Formulaire Ellipse

1) Header

```
#ifndef FORMELIPPSE_H
#define FORMELIPPSE_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"

class formElippse: public QWidget
{
   Q_OBJECT
public:
   formElippse();
   void cache();
   void affiche();
   int X1();
   int Y1();
   int X2();
   int Y2();
   QPushButton *btnValide;
   QPushButton *btnAnnule;
   QVBoxLayout *boxPrincipale;
   QSpinBox *ChangeRayon;
   QLineEdit *lineNom;
   QPushButton *btnCTrait;
   QPushButton *btnCFond;
   GuiPoint *pointA;
   GuiPoint *pointB;
private:
   QHBoxLayout * nomLayout;
   QHBoxLayout * rayonLayout;
   QHBoxLayout * CtraitLayout;
   QHBoxLayout * CFondLayout;
```

```
    QLabel * nomLabel;
    QLabel * rayonLabel;
    QLabel * nomCTrait;
    QLabel * nomCFond;
    QHBoxLayout *btnLayout;
};

#endif // FORMELIPPSE_H
```

2) Source

```
#include "formelippse.h"

formElippse::formElippse() : QWidget()
{
}

void formElippse::affiche()
{
    //Init Box le principale qui contient le formulaire
        this->boxPrincipale = new QVBoxLayout();
    //Init tous les points
        this->pointA = new GuiPoint();
        this->pointB = new GuiPoint();
        this->pointA->labelPoint->setText("Point A :");
        this->pointA->reinit();
        this->pointB->labelPoint->setText("Point B :");
        this->pointB->reinit();
    //Init tous les box horizontales
        this->nomLayout = new QHBoxLayout();
        this->CFondLayout = new QHBoxLayout();
        this->CtraitLayout = new QHBoxLayout();
        this->btnLayout = new QHBoxLayout();
    //Init le nom
        this->nomLabel = new QLabel("Nom : ");
        this->lineNom = new QLineEdit();
        this->nomLayout->addWidget(this->nomLabel);
        this->nomLayout->addWidget(this->lineNom);
        this->boxPrincipale->addLayout(this->nomLayout);
        this->lineNom->clear();
    //Init les points
        this->boxPrincipale->addLayout(this->pointA->pointLayout);
        this->boxPrincipale->addLayout(this->pointB->pointLayout);
    //init le rayon
        this->rayonLabel = new QLabel("Rayon :");
        this->rayonLayout = new QHBoxLayout();
        this->ChangeRayon = new QSpinBox();
        this->rayonLayout->addWidget(this->rayonLabel);
        this->rayonLayout->addWidget(this->ChangeRayon);
        this->boxPrincipale->addLayout(this->rayonLayout);
    //Init la couleur du trait
        this->nomCTrait = new QLabel("Couleur du trait :");
        this->btnCTrait = new QPushButton();
        this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
        this->CtraitLayout->addWidget(this->nomCTrait);
```

```
    this->CtraitLayout->addWidget(this->btnCTrait);
    this->boxPrincipale->addLayout(this->CtraitLayout);
  //Init la couleur du fond
    this->nomCFond = new QLabel("Couleur du fond :");
    this->btnCFond = new QPushButton();
    this->btnCFond->setIcon(QIcon(":/icones/logos/water.png"));
    this->CFondLayout->addWidget(this->nomCFond);
    this->CFondLayout->addWidget(this->btnCFond);
    this->boxPrincipale->addLayout(this->CFondLayout);
  //Btn de validation
    this->btnValide = new QPushButton();
    this->btnAnnule = new QPushButton();
    this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
    this->btnLayout->addWidget(btnValide);
    this->btnLayout->addWidget(btnAnnule);
    this->boxPrincipale->addLayout(btnLayout);
}

void formElippse::cache()
{
  //Suppression points
    this->pointA->detruit();
    this->pointB->detruit();
  //Suppression nom
    delete this->nomLabel;
    delete this->nomLayout;
    delete this->lineNom;
  //Suppression du rayon
    delete this->rayonLabel;
    delete this->ChangeRayon;
    delete this->rayonLayout;
  //Suppresion couleur trait
    delete this->nomCTrait;
    delete this->btnCTrait;
    delete this->CtraitLayout;
  //Suppresion couleur trait
    delete this->nomCFond;
    delete this->btnCFond;
    delete this->CFondLayout;
  //Suppresion btn valide
    delete this->btnLayout;
    delete this->btnValide;
    delete this->btnAnnule;
  //Suppresion de tout le box
    delete this->boxPrincipale;
}

int formElippse::X1()
{
  return this->pointA->ptx->value();
}

int formElippse::Y1()
{
  return this->pointA->pty->value();
```

```
}

int formElippse::X2()
{
   return this->pointB->ptx->value();
}

int formElippse::Y2()
{
   return this->pointB->pty->value();
}
```

E) Formulaire Segment

1) Header

```
#ifndef FORMSEGMENT_H
#define FORMSEGMENT_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"


class formSegment: public QWidget
{
   Q_OBJECT
public:
   formSegment();
   void cache();
   void affiche();
   int X1();
   int X2();
   int Y1();
   int Y2();
   QPushButton *btnValide;
   QPushButton *btnAnnule;
   QVBoxLayout *boxPrincipale;
   QLineEdit *lineNom;
   QPushButton *btnCTrait;
   GuiPoint *pointA;
   GuiPoint *pointB;
private:
   QHBoxLayout * nomLayout;
   QHBoxLayout * CtraitLayout;
   QLabel * nomLabel;
   QLabel * nomCTrait;
   QHBoxLayout *btnLayout;
};
```

```
#endif // FORMSEGMENT_H
```

2) Source

```
#include "formsegment.h"

formSegment::formSegment() : QWidget()
{
}

void formSegment::affiche()
{
   //Init Box le principale qui contient le formulaire
      this->boxPrincipale = new QVBoxLayout();
   //Init tous les points
      this->pointA = new GuiPoint();
      this->pointB = new GuiPoint();
      this->pointA->labelPoint->setText("Point A :");
      this->pointB->labelPoint->setText("Point B :");
      this->pointA->reinit();
      this->pointB->reinit();
   //Init tous les box horizontales
      this->nomLayout = new QHBoxLayout();
      this->CtraitLayout = new QHBoxLayout();
      this->btnLayout = new QHBoxLayout();
   //Init le nom
      this->nomLabel = new QLabel("Nom : ");
      this->lineNom = new QLineEdit();
      this->nomLayout->addWidget(this->nomLabel);
      this->nomLayout->addWidget(this->lineNom);
      this->boxPrincipale->addLayout(this->nomLayout);
      this->lineNom->clear();
   //Init les points
      this->boxPrincipale->addLayout(this->pointA->pointLayout);
      this->boxPrincipale->addLayout(this->pointB->pointLayout);
   //Init la couleur du trait
      this->nomCTrait = new QLabel("Couleur du trait :");
      this->btnCTrait = new QPushButton();
      this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
      this->CtraitLayout->addWidget(this->nomCTrait);
      this->CtraitLayout->addWidget(this->btnCTrait);
      this->boxPrincipale->addLayout(this->CtraitLayout);
   //Btn de validation
      this->btnValide = new QPushButton();
      this->btnAnnule = new QPushButton();
      this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
      this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
      this->btnLayout->addWidget(btnValide);
      this->btnLayout->addWidget(btnAnnule);
      this->boxPrincipale->addLayout(btnLayout);
}

void formSegment::cache()
{
```

```
    //Suppression points
        this->pointA->detruit();
        this->pointB->detruit();
    //Suppression nom
        delete this->nomLabel;
        delete this->nomLayout;
        delete this->lineNom;
    //Suppresion couleur trait
        delete this->nomCTrait;
        delete this->btnCTrait;
        delete this->CtraitLayout;
    //Suppresion btn valide
        delete this->btnLayout;
        delete this->btnValide;
        delete this->btnAnnule;
    //Suppresion de tout le box
        delete this->boxPrincipale;
}

int formSegment::X1()
{
    return this->pointA->ptx->value();
}

int formSegment::X2()
{
    return this->pointB->ptx->value();
}

int formSegment::Y1()
{
    return this->pointA->pty->value();
}

int formSegment::Y2()
{
    return this->pointB->pty->value();
}
```

F) Formulaire Triangle

1) Header

```
#ifndef FORMTRIANGLE_H
#define FORMTRIANGLE_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"
```

```
class formTriangle: public QWidget
{
   Q_OBJECT
public:
   formTriangle();
   void cache();
   void affiche();
   int X1();
   int X2();
   int Y1();
   int Y2();
   int X3();
   int Y3();
   QPushButton *btnValide;
   QPushButton *btnAnnule;
   QVBoxLayout *boxPrincipale;
   QLineEdit *lineNom;
   QPushButton *btnCTrait;
   QPushButton *btnCFond;
   GuiPoint *pointA;
   GuiPoint *pointB;
   GuiPoint *pointC;
private:
   QHBoxLayout * nomLayout;
   QHBoxLayout * CtraitLayout;
   QHBoxLayout * CFondLayout;
   QLabel * nomLabel;
   QLabel * nomCTrait;
   QLabel * nomCFond;
   QHBoxLayout *btnLayout;
};

#endif // FORMTRIANGLE_H
```

2) Source

```
#include "formtriangle.h"

formTriangle::formTriangle() : QWidget()
{
}

void formTriangle::affiche()
{
   //Init Box le principale qui contient le formulaire
      this->boxPrincipale = new QVBoxLayout();
   //Init tous les points
      this->pointA = new GuiPoint();
      this->pointB = new GuiPoint();
      this->pointC = new GuiPoint();
      this->pointA->labelPoint->setText("Point A :");
      this->pointB->labelPoint->setText("Point B :");
      this->pointC->labelPoint->setText("Point C :");
      this->pointA->reinit();
```

```
    this->pointB->reinit();
    this->pointC->reinit();
  //Init tous les box horizontales
    this->nomLayout = new QHBoxLayout();
    this->CFondLayout = new QHBoxLayout();
    this->CtraitLayout = new QHBoxLayout();
    this->btnLayout = new QHBoxLayout();
  //Init le nom
    this->nomLabel = new QLabel("Nom : ");
    this->lineNom = new QLineEdit();
    this->nomLayout->addWidget(this->nomLabel);
    this->nomLayout->addWidget(this->lineNom);
    this->boxPrincipale->addLayout(this->nomLayout);
    this->lineNom->clear();
  //Init les points
    this->boxPrincipale->addLayout(this->pointA->pointLayout);
    this->boxPrincipale->addLayout(this->pointB->pointLayout);
    this->boxPrincipale->addLayout(this->pointC->pointLayout);
  //Init la couleur du trait
    this->nomCTrait = new QLabel("Couleur du trait :");
    this->btnCTrait = new QPushButton();
    this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
    this->CtraitLayout->addWidget(this->nomCTrait);
    this->CtraitLayout->addWidget(this->btnCTrait);
    this->boxPrincipale->addLayout(this->CtraitLayout);
  //Btn de validation
    this->btnValide = new QPushButton();
    this->btnAnnule = new QPushButton();
    this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
    this->btnLayout->addWidget(btnValide);
    this->btnLayout->addWidget(btnAnnule);
    this->boxPrincipale->addLayout(btnLayout);
}

void formTriangle::cache()
{
  //Suppression points
    this->pointA->detruit();
    this->pointB->detruit();
    this->pointC->detruit();
  //Suppression nom
    delete this->nomLabel;
    delete this->nomLayout;
    delete this->lineNom;
  //Suppresion couleur trait
    delete this->nomCTrait;
    delete this->btnCTrait;
    delete this->CtraitLayout;
  //Suppresion btn valide
    delete this->btnLayout;
    delete this->btnValide;
    delete this->btnAnnule;
  //Suppresion de tout le box
    delete this->boxPrincipale;
}
```

```
int formTriangle::X1()
{
   return this->pointA->ptx->value();
}

int formTriangle::X2()
{
   return this->pointB->ptx->value();
}

int formTriangle::Y1()
{
   return this->pointA->pty->value();
}

int formTriangle::Y2()
{
   return this->pointB->pty->value();
}

int formTriangle::X3()
{
   return this->pointC->ptx->value();
}

int formTriangle::Y3()
{
   return this->pointC->ptx->value();
}
```

G) Formulaire Multi segment

1) Header

```
#ifndef FORMMULTISEGMENT_H
#define FORMMULTISEGMENT_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "gui_point.h"
#include <vector>
#include <QObject>
#include <QWidget>

using namespace std;
class formmultisegment : public QWidget
{
```

```
    Q_OBJECT
public:
    formmultisegment();
    void cache();
    void affiche(int n);
    int Nombre();
    int nombrePoint;
    QPushButton *btnValide;
    QPushButton *btnAnnule;
    QPushButton *btnAjout;
    QVBoxLayout *boxPrincipale;
    QLineEdit *lineNom;
    QPushButton *btnCTrait;
    vector<GuiPoint*> point;
private:
    QHBoxLayout * nomLayout;
    QHBoxLayout * CtraitLayout;
    QLabel * nomLabel;
    QLabel * nomCTrait;
    QVBoxLayout *pointLayout;
    QHBoxLayout *btnLayout;
private slots:
    void ajout();
};

#endif // FORMMULTISEGMENT_H
```

2) Source

```
#include "formmultisegment.h"

formmultisegment::formmultisegment() : QWidget()
{
}

void formmultisegment::affiche(int n)
{
    int i;
    //Init Box le principale qui contient le formulaire
        this->boxPrincipale = new QVBoxLayout();
        this->pointLayout = new QVBoxLayout();
    //Init tous les points
        for(i=0;i<n;i++)
        {
            GuiPoint *p =  new GuiPoint();
            this->point.push_back(p);
        }
    //Init tous les box horizontales
        this->nomLayout = new QHBoxLayout();
        this->CtraitLayout = new QHBoxLayout();
        this->btnLayout = new QHBoxLayout();
    //Init le nom
        this->nomLabel = new QLabel("Nom : ");
        this->lineNom = new QLineEdit();
        this->nomLayout->addWidget(this->nomLabel);
```

```
    this->nomLayout->addWidget(this->lineNom);
    this->boxPrincipale->addLayout(this->nomLayout);
    this->lineNom->clear();
  //Init les points
    this->nombrePoint = 0;
    for(i=0;i<n;i++)
    {
      GuiPoint *p = this->point[i];
      p->labelPoint->setText("Point :");
      p->reinit();
      this->pointLayout->addLayout(p->pointLayout);
      this->nombrePoint++;
    }
    this->boxPrincipale->addLayout(this->pointLayout);
  //Init la couleur du trait
    this->nomCTrait = new QLabel("Couleur du trait :");
    this->btnCTrait = new QPushButton();
    this->btnCTrait->setIcon(QIcon(":/icones/logos/water.png"));
    this->CtraitLayout->addWidget(this->nomCTrait);
    this->CtraitLayout->addWidget(this->btnCTrait);
    this->boxPrincipale->addLayout(this->CtraitLayout);
  //Btn de validation
    this->btnValide = new QPushButton();
    this->btnAnnule = new QPushButton();
    this->btnAjout = new QPushButton();
    this->btnValide->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnAnnule->setIcon(QIcon(":/icones/logos/64.png"));
    this->btnAjout->setIcon(QIcon(":/icones/logos/plus-sign.png"));
    this->btnLayout->addWidget(btnAjout);
    this->btnLayout->addWidget(btnValide);
    this->btnLayout->addWidget(btnAnnule);
    this->boxPrincipale->addLayout(btnLayout);
    QObject::connect(this->btnAjout,SIGNAL(clicked()),this,SLOT(ajout()));
}

void formmultisegment::cache()
{
  int i;
  //Suppression points
    for(i = this->point.size()-1; i >= 0;i--)
    {
      GuiPoint *p = this->point[i];
      p->detruit();
      this->point.pop_back();
      this->nombrePoint--;
    }
    delete  this->pointLayout;
  //Suppression nom
    delete this->nomLabel;
    delete this->nomLayout;
    delete this->lineNom;
  //Suppresion couleur trait
    delete this->nomCTrait;
    delete this->btnCTrait;
    delete this->CtraitLayout;
  //Suppresion btn valide
```

```
    delete this->btnLayout;
    delete this->btnAjout;
    delete this->btnValide;
    delete this->btnAnnule;
  //Suppresion de tout le box
    delete this->boxPrincipale;
}

int formmultisegment::Nombre()
{
  return this->nombrePoint;
}

void formmultisegment::ajout()
{
  GuiPoint *p =  new GuiPoint();
  this->point.push_back(p);
  p->labelPoint->setText("Point :");
  p->reinit();
  this->pointLayout->addLayout(p->pointLayout);
  this->nombrePoint++;
}
```

H) Gui Point

1) Header

```
#ifndef GUI_POINT_H
#define GUI_POINT_H

#include <QWidget>
#include <QLabel>
#include <QSpinBox>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>

class GuiPoint : public QWidget
{
public:
  GuiPoint();
  void reinit();
  void detruit();
  QHBoxLayout *pointLayout;
  QLabel *labelPoint;
  QLabel *labelx;
  QSpinBox *ptx;
  QLabel *labely;
  QSpinBox *pty;

};

#endif // GUI_POINT_H
```

2) Source

```
#include "gui_point.h"

GuiPoint::GuiPoint() : QWidget()
{
   //Définition du point
   this->labelPoint = new QLabel("Point : ");
   this->pointLayout = new QHBoxLayout();
   this->labelx = new QLabel("x : ");
   this->ptx = new QSpinBox();
   this->labely = new QLabel("y : ");
   this->pty = new QSpinBox();
   this->pointLayout->addWidget(this->labelPoint);
   this->pointLayout->addWidget(this->labelx);
   this->pointLayout->addWidget(this->ptx);
   this->pointLayout->addWidget(this->labely);
   this->pointLayout->addWidget(this->pty);
   this->ptx->setMaximum(1000);
   this->pty->setMaximum(1000);
   this->ptx->setMinimum(0);
   this->pty->setMinimum(0);
}

void GuiPoint::reinit()
{
   this->ptx->setValue(0);
   this->pty->setValue(0);
}

void GuiPoint::detruit()
{
   delete pointLayout;
   delete labelPoint;
   delete labelx;
   delete ptx;
   delete labely;
   delete pty;
}
```

I) Presentation figure

1) Header

```
#ifndef PRESENTATIONOBJET_H
#define PRESENTATIONOBJET_H

#include <QtWidgets>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QString>
#include "bu.h"
#include "formcarre.h"
#include "formcercle.h"
#include "formelippse.h"
#include "formsegment.h"
#include "formtriangle.h"
#include "formmultisegment.h"
#include <vector>
#include <QObject>
#include <QPropertyAnimation>
#include <QWidget>

class presentationObjet : public QWidget
{
   Q_OBJECT
public:
   presentationObjet();
   void cache();
   void setNom(QString s);
   void setFig(QFigure *f);
   QString Nom();
   QPushButton *btnT;
   QPushButton *btnR;
   QPushButton *btnM;
   QPushButton *btnS;
   QPushButton *btnV;
   QPushButton *btnA;
   QHBoxLayout * layoutH;
   QVBoxLayout * box;
   formCarre *formCar;
   formCercle *formCer;
   formSegment *formSeg;
   formTriangle *formTri;
   formElippse *formEli;
   formmultisegment *formMultiSeg;
private:
   QFigure *f;
   QSpinBox *spinX;
   QSpinBox *spinY;
```

```
    QSpinBox *spinTps;
    QLabel *lX;
    QLabel *lY;
    QLabel *lTps;
    QLabel * nom;
    void annule();
private slots:
    void valideTrans();
    void valideRot();
    void valideModifSeg();
    void valideModifCar();
    void valideModifCer();
    void valideModifEli();
    void valideModifTri();
    void valideModifMultiSeg();
    void annuleTrans();
    void annuleRot();
    void annuleModifSeg();
    void annuleModifCar();
    void annuleModifCer();
    void annuleModifEli();
    void annuleModifTri();
    void annuleModifMultiSeg();
    void modif();
    void supp();
    void rotation();
    void translation();
};
#endif // PRESENTATIONOBJET_H
```

2) Source

```
#include "presentationobjet.h"

//Constructeur de l'objet présenation de l'objet
presentationObjet::presentationObjet() : QWidget()
{
    this->formSeg = new formSegment();
    this->formCar = new formCarre();
    this->formCer = new formCercle();
    this->formEli = new formElippse();
    this->formTri = new formTriangle();
    this->formMultiSeg = new formmultisegment();
    this->f = new QFigure();
    this->box = new QVBoxLayout();
    this->nom = new QLabel(this->f->nom);
    this->layoutH = new QHBoxLayout();
    this->btnT = new QPushButton();
    this->btnR = new QPushButton();
    this->btnM = new QPushButton();
    this->btnS = new QPushButton();
    this->btnR->setIcon(QIcon(":/icones/logos/arrow-small-18.png"));
    this->btnT->setIcon(QIcon(":/icones/logos/arrow-big-11.png"));
    this->btnS->setIcon(QIcon(":/icones/logos/croix.png"));
    this->btnM->setIcon(QIcon(":/icones/logos/write.png"));
```

```
    this->btnS->setStatusTip("Supprimer");
    this->btnM->setStatusTip("Modifer");
    this->btnT->setStatusTip("Translation");
    this->btnR->setStatusTip("Rotation");
    this->layoutH->addWidget(this->nom);
    this->layoutH->addWidget(this->btnT);
    this->layoutH->addWidget(this->btnR);
    this->layoutH->addWidget(this->btnM);
    this->layoutH->addWidget(this->btnS);
    this->box->addLayout(this->layoutH);
    QObject::connect(this->btnT,SIGNAL(clicked()),this,SLOT(translation()));
    QObject::connect(this->btnR,SIGNAL(clicked()),this,SLOT(rotation()));
    QObject::connect(this->btnM,SIGNAL(clicked()),this,SLOT(modif()));
    QObject::connect(this->btnS,SIGNAL(clicked()),this,SLOT(supp()));
}

//Cache la présenation de l'objet
void presentationObjet::cache()
{
    delete this->nom;
    delete this->layoutH;
    delete this->btnT;
    delete this->btnR;
    delete this->btnM;
    delete this->btnS;
    delete this->box;
}

//Setters
void presentationObjet::setNom(QString s)
{
    this->nom->setText(s);
}

void presentationObjet::setFig(QFigure *f)
{
    this->f = f;
}

//Getters
QString presentationObjet::Nom()
{
    return this->nom->text();
}

//*****************************************SLOT
TRANSLATION*****************************************
void presentationObjet::translation()
{
    //On supprime les btns existant
        delete this->btnT;
        delete this->btnR;
        delete this->btnM;
        delete this->btnS;
    //On crée les labels
        this->lX = new QLabel("x :");
```

```
    this->lY = new QLabel("y : ");
  //On crée les spinbox
    this->spinX = new QSpinBox();
    this->spinY = new QSpinBox();
    this->spinTps = new QSpinBox();
    this->spinX->setMaximum(999);
    this->spinX->setMinimum(-999);
    this->spinY->setMaximum(999);
    this->spinY->setMinimum(-999);
    this->spinTps->setMinimum(0);
    this->spinTps->setMaximum(10000);
  //On crée les btn
    this->btnV = new QPushButton();
    this->btnA = new QPushButton();
    this->btnV->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnA->setIcon(QIcon(":/icones/logos/64.png"));
    this->layoutH->addWidget(this->lX);
    this->layoutH->addWidget(this->spinX);
    this->layoutH->addWidget(this->lY);
    this->layoutH->addWidget(this->spinY);
    this->layoutH->addWidget(this->spinTps);
    this->layoutH->addWidget(this->btnV);
    this->layoutH->addWidget(this->btnA);
    QObject::connect(this->btnV,SIGNAL(clicked()),this,SLOT(valideTrans()));
    QObject::connect(this->btnA,SIGNAL(clicked()),this,SLOT(annuleTrans()));
}

//Slot valide translation
void presentationObjet::valideTrans()
{
  Anim a;
  a.x = this->spinX->value();
  a.y = this->spinY->value();
  a.tps = this->spinTps->value();
  this->f->pileAnim.push_back(a);
  this->f->estAnim = true;
  this->btnA->click();
}

//Slot annule la translation
void presentationObjet::annuleTrans()
{
  delete this->btnA;
  delete this->btnV;
  delete this->lX;
  delete this->lY;
  delete this->spinX;
  delete this->spinY;
  delete this->spinTps;
  this->annule();
}

//*****************************************SLOT ROTATION*****************************************
void presentationObjet::rotation()
{
  //On supprime les btns existant
```

```
    delete this->btnT;
    delete this->btnR;
    delete this->btnM;
    delete this->btnS;
  //On crée les labels
    this->lX = new QLabel("Angle :");
  //On crée les spinbox
    this->spinX = new QSpinBox();
    this->spinY = new QSpinBox();
    this->spinX->setMaximum(360);
    this->spinX->setMinimum(0);
  //On crée les btn
    this->btnV = new QPushButton();
    this->btnA = new QPushButton();
    this->btnV->setIcon(QIcon(":/icones/logos/65.png"));
    this->btnA->setIcon(QIcon(":/icones/logos/64.png"));
    this->layoutH->addWidget(this->lX);
    this->layoutH->addWidget(this->spinX);
    this->layoutH->addWidget(this->btnV);
    this->layoutH->addWidget(this->btnA);
    QObject::connect(this->btnV,SIGNAL(clicked()),this,SLOT(valideRot()));
    QObject::connect(this->btnA,SIGNAL(clicked()),this,SLOT(annuleRot()));
}

//Slot valide translation
void presentationObjet::valideRot()
{
  int x = this->spinX->value();
  int i;
  for(i=0;i<this->f->item.size();i++)
  {
    QGraphicsItem *item = this->f->item[i];
    item->setTransformOriginPoint(this->f->oX,this->f->oY);
    item->setRotation(x);
  }
  this->btnA->click();
}

//Slot annule la translation
void presentationObjet::annuleRot()
{
  delete this->btnA;
  delete this->btnV;
  delete this->lX;
  delete this->spinX;
  this->annule();
}

//*****************************************SLOT
MODIFCATION*****************************************
void presentationObjet::modif()
{
  //On supprime les btns existant
    delete this->btnT;
    delete this->btnR;
    delete this->btnM;
```

```
    delete this->btnS;
  switch (this->f->o)
  {
    case SEGMENT:
      this->formSeg = new formSegment();
      this->formSeg->affiche();
      QObject::connect(this->formSeg->btnValide, SIGNAL(clicked()), this, SLOT(valideModifSeg()));
      QObject::connect(this->formSeg->btnAnnule, SIGNAL(clicked()), this, SLOT(annuleModifSeg()));
      this->formSeg->lineNom->setText(this->f->nom);
      this->box->addLayout(this->formSeg->boxPrincipale);
      break;
    case CERCLE:
      this->formCer = new formCercle();
      this->formCer->affiche();
      QObject::connect(this->formCer->btnValide, SIGNAL(clicked()), this, SLOT(valideModifCer()));
      QObject::connect(this->formCer->btnAnnule, SIGNAL(clicked()), this, SLOT(annuleModifCer()));
      this->box->addLayout(this->formCer->boxPrincipale);
      break;
    case ELLIPSE:
      this->formEli = new formElippse();
      this->formEli->affiche();
      QObject::connect(this->formEli->btnValide, SIGNAL(clicked()), this, SLOT(valideModifEli()));
      QObject::connect(this->formEli->btnAnnule, SIGNAL(clicked()), this, SLOT(annuleModifELi()));
      this->box->addLayout(this->formEli->boxPrincipale);
      break;
    case TRIANGLE:
      this->formTri = new formTriangle();
      this->formTri->affiche();
      QObject::connect(this->formTri->btnValide, SIGNAL(clicked()), this, SLOT(valideModifTri()));
      QObject::connect(this->formTri->btnAnnule, SIGNAL(clicked()), this, SLOT(annuleModifTri()));
      this->box->addLayout(this->formTri->boxPrincipale);
      break;
    case CARRE:
      this->formCar = new formCarre();
      this->formCar->affiche();
      QObject::connect(this->formCar->btnValide, SIGNAL(clicked()), this, SLOT(valideModifCar()));
      QObject::connect(this->formCar->btnAnnule, SIGNAL(clicked()), this, SLOT(annuleModifCar()));
      this->box->addLayout(this->formCar->boxPrincipale);
      break;
    case MULTISEG :
      this->formMultiSeg = new formmultisegment();
      this->formMultiSeg->nombrePoint = this->f->item.size();
      this->formMultiSeg->affiche(this->f->item.size()+1);
      QObject::connect(this->formMultiSeg->btnValide, SIGNAL(clicked()), this,
SLOT(valideModifMultiSeg()));
      QObject::connect(this->formMultiSeg->btnAnnule, SIGNAL(clicked()), this,
SLOT(annuleModifMultiSeg()));
      this->box->addLayout(this->formMultiSeg->boxPrincipale);
      break;
    default:
      break;
  }
}
void presentationObjet::valideModifSeg()
{
  //On récupère les nouvelles coordonnées
```

```
    int x1 = this->formSeg->X1();
    int y1 = this->formSeg->Y1();
    int x2 = this->formSeg->X2();
    int y2 = this->formSeg->Y2();
  //On crée un nouvelle objet
    Segment s(Point2D(x1,y1),Point2D(x2,y2));
  //On réaffecte le nouvelle objet dans la structure
    this->f->f = s;
  //On crée un nouvelle item et on l'affecte

  //On change le nom
    this->f->nom = this->formSeg->lineNom->text();
    this->nom->setText(this->f->nom);
  //On cache le formulaire
    this->formSeg->btnAnnule->click();
}

void presentationObjet::valideModifCar()
{
  //On récupère les nouvelles coordonnées
    int x1 = this->formCar->X1();
    int y1 = this->formCar->Y1();
    int x2 = this->formCar->X2();
    int y2 = this->formCar->Y2();
    int x3 = this->formCar->X3();
    int y3 = this->formCar->Y3();
    int x4 = this->formCar->X4();
    int y4 = this->formCar->Y4();
  //On crée un nouvelle objet
    Rectangle r(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3),Point2D(x4,y4));
  //On réaffecte le nouvelle objet dans la structure
    this->f->f = r;
  //On crée un nouvelle item et on l'affecte
    QGraphicsRectItem *item = new QGraphicsRectItem();
    item->setRect(x1,y1,x2-x1,y3-y1);
  //On change le nom
    this->f->nom = this->formCar->lineNom->text();
    this->nom->setText(this->f->nom);
  //On cache le formulaire
    this->formCar->btnAnnule->click();
}

void presentationObjet::valideModifCer()
{
  //On récupère les nouvelles coordonnées
    int x1 = this->formCer->X1();
    int y1 = this->formCer->Y1();
    int r = this->formCer->ChangeRayon->value();
  //On crée un nouvelle objet
    Cercle c(Point2D(x1,y1),r);
  //On réaffecte le nouvelle objet dans la structure
     this->f->f = c;
  //On crée un nouvelle item et on l'affecte
    QGraphicsEllipseItem *item = new QGraphicsEllipseItem(x1,y1,r,r);
    this->f->item[0] = item;
  //On change le nom
```

```
    this->f->nom = this->formCer->lineNom->text();
    this->nom->setText(this->f->nom);
  //On cache le formulaire
    this->formCer->btnAnnule->click();
}

void presentationObjet::valideModifEli()
{
  //On récupère les nouvelles coordonnées
    int x1 = this->formEli->X1();
    int y1 = this->formEli->Y1();
    int x2 = this->formEli->X2();
    int y2 = this->formEli->Y2();
    int r = this->formEli->ChangeRayon->value();
  //On crée un nouvelle objet
    Elippse e(Point2D(x1,y1),Point2D(x2,y2),r);
  //On réaffecte le nouvelle objet dans la structure
     this->f->f = e;
  //On crée un nouvelle item et on l'affecte
    QGraphicsEllipseItem *item = new QGraphicsEllipseItem(x1,y1,r,r);
    this->f->item[0] = item;
  //On change le nom
    this->f->nom = this->formEli->lineNom->text();
    this->nom->setText(this->f->nom);
  //On cache le formulaire
    this->formEli->btnAnnule->click();

}

void presentationObjet::valideModifTri()
{
  //On récupère les nouvelles coordonnées
    int x1 = this->formTri->X1();
    int y1 = this->formTri->Y1();
    int x2 = this->formTri->X2();
    int y2 = this->formTri->Y2();
    int x3 = this->formTri->X3();
    int y3 = this->formTri->Y3();
  //On crée un nouvelle objet
    Triangle t(Point2D(x1,y1),Point2D(x2,y2),Point2D(x3,y3));
  //On réaffecte le nouvelle objet dans la structure
    this->f->f = t;
  //On crée un nouvelle item et on l'affecte
    QGraphicsLineItem *item = new QGraphicsLineItem(x1,y1,x2,y2);
    this->f->item[0] = item;
    item = new QGraphicsLineItem(x2,y2,x3,y3);
    this->f->item[1] = item;
    item = new QGraphicsLineItem(x3,y3,x1,y1);
    this->f->item[2] = item;
  //On change le nom
    this->f->nom = this->formTri->lineNom->text();
    this->nom->setText(this->f->nom);
  //On cache le formulaire
    this->formTri->btnAnnule->click();
}
```

```
void presentationObjet::valideModifMultiSeg()
{
      int i;
      GuiPoint *p1 = this->formMultiSeg->point[0];
      for(i=1;i<this->formMultiSeg->Nombre();i++)
      {
         GuiPoint *p2 = this->formMultiSeg->point[i];
         //Traçage de la figure dans le QGraphicsScene
         //QGraphicsItem *item = ui->view->scene->addLine(p1->ptx->value(),p1->pty->value(),p2->ptx-
>value(),p2->pty->value(),QPen(this->colorT));
         //Ajout du pointeur sur le tracé de la figure
         //fig->item.push_back(item);
         p1 = this->formMultiSeg->point[i];
      }
   //On change le nom
      this->f->nom = this->formMultiSeg->lineNom->text();
      this->nom->setText(this->f->nom);
      this->formMultiSeg->btnAnnule->click();

}

void presentationObjet::annuleModifSeg()
{
   this->formSeg->cache();
   this->annule();
}

void presentationObjet::annuleModifCar()
{
   this->formCar->cache();
   this->annule();
}
void presentationObjet::annuleModifCer()
{
   this->formCer->cache();
   this->annule();
}
void presentationObjet::annuleModifEli()
{
   this->formEli->cache();
   this->annule();
}
void presentationObjet::annuleModifTri()
{
   this->formTri->cache();
   this->annule();
}
void presentationObjet::annuleModifMultiSeg()
{
   this->formMultiSeg->cache();
   this->annule();
}

//*****************************************SLOT SUPPRIME*****************************************
void presentationObjet::supp()
{
```

```
    int i;
    for(i=0;i<this->f->item.size();i++)
    {
        QGraphicsItem *item = this->f->item[i];
        item->setVisible(false);
    }
    this->f->etat = false;
    this->cache();
}

//Fonctions
void presentationObjet::annule()
{
    this->btnT = new QPushButton();
    this->btnM = new QPushButton();
    this->btnS = new QPushButton();
    this->btnR = new QPushButton();
    this->btnR->setIcon(QIcon(":/icones/logos/arrow-small-18.png"));
    this->btnT->setIcon(QIcon(":/icones/logos/arrow-big-11.png"));
    this->btnS->setIcon(QIcon(":/icones/logos/croix.png"));
    this->btnM->setIcon(QIcon(":/icones/logos/write.png"));
    this->btnR->setStatusTip("Rotation");
    this->btnS->setStatusTip("Supprimer");
    this->btnM->setStatusTip("Modifer");
    this->btnT->setStatusTip("Translation");
    this->layoutH->addWidget(this->btnT);
    this->layoutH->addWidget(this->btnR);
    this->layoutH->addWidget(this->btnM);
    this->layoutH->addWidget(this->btnS);
    QObject::connect(this->btnT,SIGNAL(clicked()),this,SLOT(translation()));
    QObject::connect(this->btnR,SIGNAL(clicked()),this,SLOT(rotation()));
    QObject::connect(this->btnM,SIGNAL(clicked()),this,SLOT(modif()));
    QObject::connect(this->btnS,SIGNAL(clicked()),this,SLOT(supp()));
}
```

J) Vue graphique

1) Header

```
#ifndef VUEGRAPHIQUE_H
#define VUEGRAPHIQUE_H
#include <QtWidgets>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QString>
#include <QGraphicsTextItem>

class VueGraphique : public QGraphicsView
{
    public:
        VueGraphique();
        QGraphicsScene *scene;
        QLabel *labelX;
```

```
    QLabel *labelY;
  private :
    void mouseMoveEvent ( QMouseEvent * event );
};

#endif // VUEGRAPHIQUE_H
```

2) Source

```
#include "vuegraphique.h"

VueGraphique::VueGraphique() : QGraphicsView()
{
  this->labelX = new QLabel();
  this->labelY = new QLabel();
  this->scene = new QGraphicsScene();
  this->setScene(this->scene);
  //Création de la grille
  int i = 0,j = 0;
  for (int i = 0; i < 1500; i += 30)
  {
    for (int j = 0; j < 900; j += 30)
    {
      this->scene->addRect(25,25,i,j,QPen(Qt::gray));
    }
  }
}

void VueGraphique::mouseMoveEvent ( QMouseEvent * event )
{
  this->labelX->setText(QString::number(event->x()));
  this->labelY->setText(QString::number(event->y()));
}
```

Autres fichiers

```
#ifndef BU_H
#define BU_H

#include "vuegraphique.h"
#include "objets/Cercle/ArcCercle.h"
#include "objets/Base/Figure.h"
#include "objets/Quadrilatere/Rectangle.h"
#include "objets/Cercle/Cercle.h"
#include "objets/Triangle/Triangle.h"
#include "objets/Base/Segment.h"
#include "objets/Cercle/Elippse.h"
#include <vector>

enum Objet{SEGMENT, CARRE, TRIANGLE, CERCLE, MULTISEG, ELLIPSE};//Enumeration des figures

typedef struct
{
  int x;
```

```
    int y;
    int tps;
}Anim;
typedef struct
{
    Figure f;
    vector<Point2D> points;
    Objet o;
    vector<QGraphicsItem *> item;
    vector<Anim> pileAnim;
    QString nom;
    int oX;
    int oY;
    int r;
    int cR;
    int cG;
    int cB;
    int cfR;
    int cfG;
    int cfB;
    bool estAnim;
    bool etat;
}QFigure;

typedef struct
{
    int id;
    QFigure *f;
}QDessin;
#endif // BU_H
```