Bibliothèque de gestion d'objets graphiques 2D

1 Objectifs

Ce projet a pour objet de faire développer une application complète, plus complexe que celles habituellement proposées lors de sujets de TP en temps limité.

En laissant une grande part à l'initiative des étudiants, ceux-ci pourront approfondir leur maîtrise du langage C, mettre en évidence leurs capacités à trouver des solutions originales mais également être confrontés aux problèmes de complexité de développement d'applications de moyenne envergure.

Ce projet pourra commencer par une familiarisation avec la bibliothèque graphique fournie à travers l'étude et la modification d'un exemple fourni.

Ensuite les étudiants pourront choisir une application de leur choix (casse-brique, circuit automobile, tetris, tennis ...) et ils proposeront alors à l'enseignant une étude de la solution envisagée (sur papier, technique utilisée, analyse descendante ...). Une fois les étudiants et l'enseignant d'accord sur la réalisation à entreprendre, celle-ci pourra avoir lieu.

Un rapport présentant l'analyse du problème, les détails de réalisation et les possibilités d'évolution du logiciel fera l'objet d'une évaluation.

2 La bibliothèque graphique

Dans le répertoire `LibObjects2D` se trouvent des fichiers et répertoires utiles au développement de votre application. Le fichier d'interface (`LibObjects2D/include/CObject2D.h`) donne accès à une bibliothèque vous permettant de manipuler des `Object2D`s. Il s'agit d'un type opaque décrivant un objet graphique situé dans un plan (x, y, θ) structuré en couches (avant/arrière-plan).

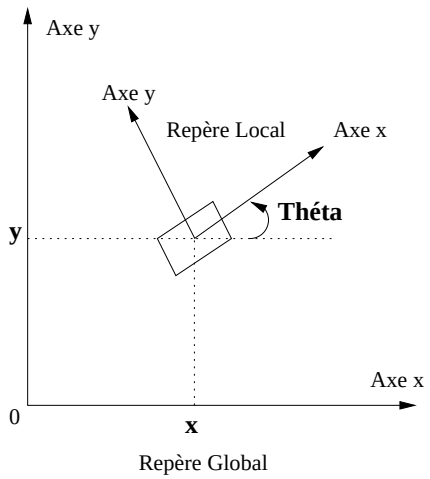

Figure 1: Description du repère global et du repère local associé à un objet graphique

Le lecteur trouvera dans la suite de cette section les fonctions qui sont associées à ce type de donnée.

2.1 Fonctions de Création/destruction/copie

▷
```
Object2D *
Object2D_new(void);
```

- ◇ Instancier un `Object2D`.
- ◇ Sa représentation est un point blanc.
- ◇ Il se trouve sur la couche de visualisation 0.
- ◇ Sa situation absolue sur le plan est $(x = 0, y = 0, \theta = 0)$

▷
```
Object2D *
Object2D_newCopy(const Object2D * anObj2d);
```

- ◇ Instancier un `Object2D`.
- ◇ Sa représentation est celle de l'`Object2D` passé en paramètre.

▷
```
void
Object2D_copy(Object2D * obj2d, const Object2D * anObj2d);
```

- ◇ Copier `anObj2d` dans `obj2d` (c'est-à-dire $\simeq$ `*obj2d = *anObj2d`).

▷
```
void
Object2D_delete(Object2D * obj2d);
```

- ◇ Détruire `obj2d`.

2.2 Fonctions de gestion de la position et de l'orientation

▷
```
void
Object2D_setLocation(Object2D * obj2d,
                     double x, double y,
                     double theta);
```

- ◇ Situer `obj2d` en absolu dans le plan.

▷
```
void
Object2D_getLocation(const Object2D * obj2d,
                     double * xOut, double * yOut,
                     double * thetaOut);
```

- ◇ Obtenir la siuuation absolue d'`obj2d` dans le plan.

▷
```
void
Object2D_setX(Object2D * obj2d,
              double x);
```

- ◇ Fixer l'abscisse absolue d'`obj2d` dans le plan.

▷
```
double
Object2D_getX(const Object2D * obj2d);
```

◇ Obtenir l'abscisse absolue d'`obj2d` dans le plan.

▷
```
void
Object2D_setY(Object2D * obj2d,
              double x);
```

◇ Fixer l'ordonnée absolue d'`obj2d` dans le plan.

▷
```
double
Object2D_getY(const Object2D * obj2d);
```

◇ Obtenir l'ordonnée absolue d'`obj2d` dans le plan.

▷
```
void
Object2D_setTheta(Object2D * obj2d,
                  double theta);
```

◇ Fixer l'orientation absolue d'`obj2d` dans le plan.

▷
```
double
Object2D_getTheta(const Object2D * obj2d);
```

◇ Obtenir l'orientation absolue d'`obj2d` dans le plan.

▷
```
void
Object2D_translate(Object2D * obj2d,
                   double dx, double dy);
```

◇ Translation relative d'`obj2d`.

◇ `dx` et `dy` sont exprimés dans le repère local d'`obj2d`.

▷
```
void
Object2D_rotate(Object2D * obj2d,
                double dTheta);
```

◇ Rotation relative d'`obj2d`.

◇ `dTheta` est exprimé dans le repère local d'`obj2d`.

2.3 Fonctions de changements de repères (global/local)

▷
```
void
Object2D_globalToLocalPosition(const Object2D * obj2d,
                               double * xInOut, double * yInOut);
```

◇ Conversion d'une position globale dans un repère local.

◇ `xInOut` et `yInOut` sont exprimés dans le repère global en entrée et dans le repère local d'`obj2d` en sortie.

▷
```
void
Object2D_localToGlobalPosition(const Object2D * obj2d,
                               double * xInOut, double * yInOut);
```

- ◇ Conversion d'une position locale dans le repère global.
- ◇ `xInOut` et `yInOut` sont exprimés dans le repère local d'`obj2d` en entrée et dans le repère global en sortie.

▷
```
double
Object2D_globalToLocalOrientation(const Object2D * obj2d,
                                  double orientation);
```

- ◇ Conversion d'`orientation` dans le repère local d'`obj2d`.
- ◇ `orientation` est exprimé dans le repère global.

▷
```
double
Object2D_localToGlobalOrientation(const Object2D * obj2d,
                                  double orientation);
```

- ◇ Conversion d'`orientation` dans le repère global.
- ◇ `orientation` est exprimé dans le repère local d'`obj2d`.

2.4 Fonctions de gestion de la couleur, de la couche de visualisation (layer) et de la forme des `Object2D`

▷
```
void
Object2D_setColor(Object2D * obj2d,
                  const char * colorName);
```

- ◇ Modifier la couleur d'`obj2d`.
- ◇ `colorName` peut désigner le nom d'une couleur (`"blue"`, `"grey"` ...) ou bien le motif `"rgb:RR/GG/BB"` où `RR`, `GG` et `BB` sont les composantes de la couleur choisie exprimées sur deux chiffres hexadécimaux (`00` à `FF`).

▷
```
const char *
Object2D_getColor(Object2D * obj2d);
```

- ◇ Obtenir la couleur d'`obj2d`. Retourne `NULL` si l'objet est invisible (cf `noShape`)

▷
```
void
Object2D_setLayer(Object2D * obj2d,
                  int layer);
```

- ◇ Placer `obj2d` sur une couche de visualisation.
- ◇ Les valeurs croissantes de `layer` progressent vers l'avant-plan.

▷
```
int
Object2D_getLayer(const Object2D * obj2d);
```

- ◇ Obtenir la couche de visualisation sur laquelle évolue `obj2d`.

▷
```
void
Object2D_noShape(Object2D * obj2d);
```

◇ `obj2d` ne prend aucune forme... devient invisible !

▷
```
void
Object2D_point(Object2D * obj2d);
```

◇ `obj2d` prend la forme d'un point.

▷
```
void
Object2D_text(Object2D* obj2d, const char * text);
```

◇ `obj2d` prend la forme d'une chaîne de caractères.

▷
```
void
Object2D_line(Object2D * obj2d,
              double length);
```

◇ `obj2d` prend la forme d'une ligne.

◇ La ligne va de $(0, 0)$ à $(length, 0)$ dans le repère local d'`obj2d`.

▷
```
void
Object2D_square(Object2D * obj2d,
                double side,
                int filled);
```

◇ `obj2d` prend la forme d'un carré.

◇ Le carré est centré en $(0, 0)$ sur le repère local d'`obj2d`, les cotés ont une longueur `side` et sont orientés selon les axes du repère local d'`obj2d`.

◇ Si `filled` est non nul, le carré est plein.

▷
```
void
Object2D_rectangle(Object2D * obj2d,
                   double length,
                   double width,
                   int filled);
```

◇ Même principe qu'avec `Object2D_square()` mais pour un rectangle.

◇ `length` et `width` donnent respectivement la longueur selon l'axe $\vec{x}$ local et la largeur selon l'axe $\vec{y}$ local.

▷
```
void
Object2D_polyline(Object2D * obj2d,
                  unsigned int nbPoints,
                  const double * xPoints,
                  const double * yPoints);
```

◇ Même principe qu'avec `Object2D_line()` mais pour une ligne brisée.

◇ Les coordonnées (x, y) sont exprimées dans le repère local d'`obj2d`.

▷
```
void
Object2D_polygon(Object2D * obj2d,
                 unsigned int nbPoints,
                 const double * xPoints,
                 const double * yPoints,
                 int filled);
```

- ◇ Même principe qu'avec `Object2D_square()` mais pour un polygone.
- ◇ Les coordonnées (x, y) sont exprimées dans le repère local d'`obj2d`.
- ◇ Le dernier point et le premier point sont reliés.

▷
```
void
Object2D_circle(Object2D * obj2d,
                double radius,
                int filled);
```

- ◇ Même principe qu'avec `Object2D_square()` mais pour un cercle.

▷
```
int /* 0: failure    !=0: success */
Object2D_image(Object2D * obj2d,
               const char * fileName,
               double pixelScale);
```

- ◇ Même principe qu'avec `Object2D_square()` mais pour une image.
- ◇ Le fichier `fileName` doit être au format `bmp` ou `ras` et utiliser une pallette (pas d'image 24 bits).
- ◇ `pixelScale` indique la taille d'un pixel de l'image dans le plan.
- ◇ Retourne un résultat nul si le fichier n'est pas au format attendu.

▷
```
int /* 0: failure    =0: success */!
Object2D_getImagePixelAt(Object2D * obj2d,
                         double x,
                         double y,
                         int * redOut,
                         int * greenOut,
                         int * blueOut);
```

- ◇ Donne la couleur du pixel qui se trouve en (x, y) dans le plan (repère global).
- ◇ Retourne un résultat nul si `obj2d` n'a pas la forme d'une image ou si (x, y) n'est pas à l'intérieur d'`obj2d`.

2.5 Fonctions de détection de représentations graphiques

▷
```
int /* 0: outside    !=0: inside */
Object2D_isInside(const Object2D * obj2d,
                  double x, double y);
```

- ◇ Non nul si le point (x, y) est situé à l'intérieur de la représentation d'`obj2d`.
- ◇ `x` et `y` sont exprimés dans le repère global.

▷ `int /* 0: no intersection found  !=0: intersection found */`
`Object2D_intersectRay(const Object2D * obj2d,`
`double xRay, double yRay,`
`double thetaRay,`
`double * xOut,`
`double * yOut);`

- ◇ Non nul si la représentation d'`obj2d` est intersectée par la demi-droite issue de $(xRay, yRay)$ et orientée selon $thetaRay$.
- ◇ `xOut` et `yOut` reçoivent alors le point d'intersection.
- ◇ `xRay`, `yRay`, `thetaRay`, `xOut` et `yOut` sont exprimés dans le repère global.

▷ `Object2D * /* NULL: no intersection found, not NULL: intersection found */`
`Object2D_throwRay(const Object2D * obj2d,`
`double * xOut,`
`double * yOut,`
`Object2D *tabObject2D[], unsigned int nbObject2D);`

- ◇ Lance un rayon devant l'objet `obj2d` (via les coordonnees et l'axe de `obj2d`).
- ◇ Retourne l'objet le plus proche intersecté par ce rayon (object contenu dans le tableau `tabObject2D`, tableau de `nbObject2D` éléments).
- ◇ Retourne `NULL` si pas d'objet intersecté.
- ◇ `xOut` et `yOut` reçoivent alors le point d'intersection.
- ◇ `xOut` et `yOut` sont exprimés dans le repère global.

2.6 Fonctions de détection d'objets dans un cône de vision

Contrairement aux fonctions de détection de représentations graphiques qui travaillent sur les formes 2D des objets, les fonctions présentées dans cette sous-section travaillent sur les positions (`x`,`y`) des objets.
Ces fonctions utilisent la notion de cône de vision. Ainsi, pour un `Object2D obj2d`, un cône de vision est déterminé par 3 réels `vision`, `range` et `turn`.

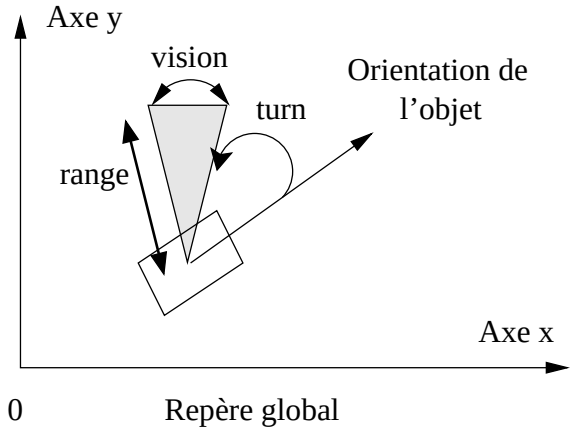

Figure 2: Cône de détection d'un object en fonction de son l'orientation et des paramètres `vision`, `range` et `turn`. Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞.

▷ `Object2D * /* NULL: no Object2D found, not NULL: an Object2D */`
`Object2D_viewFirstObject2D(const Object2D * obj2d,`
`double vision, double range, double turn);`

◇ Retourne l'`Object2D` le plus proche se trouvant dans le cône de vision de l'object `obj2d`, cône de vision déterminé par `vision`, `range` et `turn`.

◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞

▷ `int`
`Object2D_viewObject2D(const Object2D * obj2d,`
`Object2D*** tabObject2D,`
`double vision, double range, double turn);`

◇ Retourne le nombre d'`Object2D` se trouvant dans le cône de vision de l'object `obj2d`, cône de vision déterminé par `vision`, `range` et `turn`. A la sortie de la fonction, les objets se trouvant dans ce cône de vision sont disponibles dans un tableau alloué dynamiquement (avec `malloc`)... il faudra penser à faire un `free` !

◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞

◇ Exemple d'utilisation :

```
void testViewObject2D(Object2D* obj2d)
{
 double range = 10;

 Object2D** v;
 int i, nbObj2D;

 nbObj2D = Object2D_viewObject2D(obj2d,&v,2*M_PI,range,0.0);

 for(i=0;i<nbObj2D;i++)
 {
  v[i]->setColor("yellow");
 }

 free(v);  /* Important (autrement, il y a une fuite memoire !) */
}
```

Dans cet exemple, tous les objets se trouvant à une distance inférieure à `10` de `obj2d` sont détectés... Et deviennent jaunes.

2.7 Fonction d'interaction "clavier"/"souris"

▷ `void`
`Object2D_onKeyPress(Object2D * obj2d, const char * key);`

◇ Exécute le comportement par defaut d'un Object2D lorsque celui-ci est sélectionné et qu'une touche est appuyée.
Par exemple, affichage de `Object 0x100fe080 key <f>`.

▷
```
void
Object2D_onMouseDrag(Object2D * obj2d, double dx, double dy);
```

◇ Exécute le comportement par defaut d'un Object2D lorsque celui-ci est déplacé à la souris de (`dx`,`dy`).

◇ Cela revient à :

```
void
Object2D_onMouseDrag(Object2D * obj2d, double dx, double dy)
{
 double x,y,theta;

 Object2D_getLocation(&x,&y,&theta);
 x+=dx;
 y+=dy;
 Object2D_setLocation(x,y,theta);
}
```

2.8 Fonction de gestion de l'attachement d'objets ensemble

▷
```
void
Object2D_attachTo(Object2D * obj2d, const Object2D * anObj2d);
```

◇ L'Object2D `obj2d` est attaché à l'Object2D `anObj2d`.
$\Longrightarrow$ Lors d'un déplacement de `anObj2d`, `obj2d` subit le même déplacement.
$\Longrightarrow$ L'inverse n'est pas vrai...

▷
```
int /* 0: no    1: yes */
Object2D_isAttachedTo(const Object2D * obj2d,
                      const Object2D * anObj2d);
```

◇ Permet de savoir si l'Object2D `obj2d` est attaché à l'Object2D `anObj2d`.

◇ ... permet de savoir si on a fait un `Object2D_attachTo(obj2d,anObj2d);`

▷
```
void
Object2D_detachFrom(Object2D * obj2d, const Object2D * anObj2d);
```

◇ Supprime l'attachement de l'Object2D `obj2d` à l'Object2D `anObj2d`.

◇ ... opération inverse de `Object2D_attachTo(obj2d,anObj2d);`

▷
```
void
Object2D_detachFromAll(Object2D * obj2d);
```

◇ Supprime tous les attachements réalisé par l'Object2D `obj2d`.

3 Initialisation, activation et paramétrage de l'application graphique

Le fichier d'interface `LibObjects2D/include/CObject2D.h` donne également accès à des fonctions permettant d'initialiser et d'activer l'application graphique :

▷
```
void
graphic_init(const char * windowName,
             const char * fontName);
```

- ◇ Créer la fenêtre graphique en lui affectant le titre `windowName`.
- ◇ Les `Object2D`s ayant une représentation sous forme de texte utiliseront la police `fontName`.
- ◇ Cette fonction doit être appelée avant toutes les autres fonctions de la bibliothèque.

▷
```
void
graphic_setWidth(int width);
```

- ◇ Fixer la largeur en pixels de la fenêtre graphique.

▷
```
void
graphic_setHeight(int height);
```

- ◇ Fixer la hauteur en pixels de la fenêtre graphique.

▷
```
void
graphic_setBackground(const char * colorName);
```

- ◇ Changer la couleur du fond de la fenêtre graphique.

▷
```
void
graphic_setViewPoint(double x,
                     double y,
                     double scale);
```

- ◇ Modifier le point de vue de la fenêtre
- ◇ Le point (x, y) représente le point du plan qui sera situé au centre de la fenêtre.
- ◇ `scale` représente le facteur qui permet de passer des grandeurs sans dimension du plan aux *pixels* de l'écran.

▷
```
void
graphic_getViewPoint(double * xOut,
                     double * yOut,
                     double * scaleOut);
```

- ◇ Obtenir le point de vue courant de la fenêtre.
- ◇ Voir `graphic_setViewPoint()` .

▷
```
void
graphic_autoscale(void);
```

- ◇ Recadrer la fenêtre pour qu'elle montre l'ensemble des `Object2D`s.

▷ `void`
`graphic_run(void * userData);`

- ◇ Lancer la partie active (événementielle) du programme.
- ◇ `userData` désigne généralement une structure créée par vos soins, permettant d'accéder à l'ensemble des données de l'application.

▷ `void`
`graphic_mainLoop(void * userData);`

- ◇ Cette fonction est appelée perpétuellement à partir de `graphic_run()` ; c'est la cinématique de l'application.
- ◇ **Vous devez définir cette fonction !**
- ◇ Le paramètre `userData` est celui qui a été transmis à `graphic_run()`.

▷ `void`
```
graphic_keyPressCallback(Object2D * obj2d,
                         const char * key,
                         void * userData);
```

- ◇ Cette fonction est appelée lorsqu'une touche du clavier est enfoncée.
- ◇ **Vous devez définir cette fonction !**
- ◇ Si `obj2d` est non nul, il s'agit d'un objet graphique qui était sélectionné lors de l'appui sur la touche.
- ◇ Si `obj2d` est nul, aucun objet graphique n'était sélectionné au moment de l'appui sur la touche.
- ◇ La touche enfoncée est décrite de manière lisible par `key`.
- ◇ Le paramètre `userData` est celui qui a été transmis à `graphic_run()`.

▷ `void`
```
graphic_mouseDragCallback(Object2D * obj2d,
                          double dx,
                          double dy,
                          void * userData);
```

- ◇ Cette fonction est appelée lorsqu'un mouvement de souris est appliqué à un `Object2D`.
- ◇ **Vous devez définir cette fonction !**
- ◇ `obj2d` désigne l'objet sélectionné lors du mouvement de souris.
- ◇ Le mouvement est décrit par le vecteur (dx, dy) dans le repère global.
- ◇ Le paramètre `userData` est celui qui a été transmis à `graphic_run()`.

Tous les mouvements et les dimensions des `Object2D` sont exprimés dans une grandeur sans dimention ; ce ne sont pas des pixels. L'axe $\vec{x}$ croît de gauche à droite et l'axe $\vec{y}$ de bas en haut. Le point de vue de la fenêtre peut être modifié selon des translations (`Ctrl`+Click Gauche) et selon un facteur de grossissement (`Ctrl`+Click Droit).

4 A savoir: quelques fonctions utilitaires

La bibliothèque de gestion d'`Object2D` fournit également quelques fonctions utilitaires concernant le passage de coordonnées cartésiennes en coordonnées polaires (et inversement), ... et d'autres fonctions décrites ci-après.

voir le fichier `LibObjects2D/include/UtilObject2D.h`

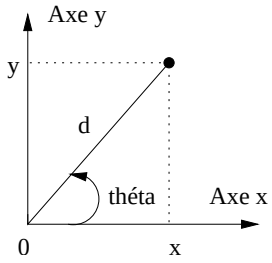

Figure 3: Rappel : coordonnées cartésiennes (`x`,`y`) – coordonnées polaires (`d`,`théta`)

▷
```
void cartesianToPolar(double x,
                      double y,
           double * distanceOut,
           double * angleOut);
```

◇ Cette fonction permet de passer des coordonnées cartésiennes (`x`,`y`) aux coordonnées polaires (`distanceOut`,`angleOut`).

▷
```
void polarToCartesian(double distance,
                      double angle,
                      double * xOut,
                      double * yOut);
```

◇ Cette fonction permet de passer des coordonnées polaires (`distance`,`angle`) aux coordonnées cartésiennes (`xOut`,`yOut`)

▷ `double cartToDist(double x,double y);`

◇ Cette fonction permet de passer des coordonnées cartésiennes (`x`,`y`) aux coordonnées polaires.

◇ Dans cette fonction, il n'y a que le calcul de la distance.

◇ Cette fonction retourne la distance ainsi calculée.

▷ `double cartToAngle(double x,double y);`

◇ Cette fonction permet de passer des coordonnées cartésiennes (`x`,`y`) aux coordonnées polaires.

◇ Dans cette fonction, il n'y a que le calcul de l'angle.

◇ Cette fonction retourne l'angle ainsi calculé.

▷ `double betweenPi(double a);`

◇ Cette retourne, à partir d'un angle `a`, une valeur comprise entre $-\pi$ et π.

◇ Exemple : si `a` est égal à $\frac{5}{4}\pi$, la valeur retournée sera : $-\frac{3}{4}\pi$.

5 L'exemple fourni

Le fichier `graphTestC.c` est un programme qui utilise de nombreuses fonctionnalités de la bibliothèque graphique. On y trouve traité un certain nombre de points qui pourront vous guider dans votre réalisation :

- ▷ Squelette général de l'application.
- ▷ Fonctions utilitaires (temps, aléatoire ...).
- ▷ Utilisation des formes, des couleurs ...
- ▷ Changement de repère, suivi de cible.
- ▷ Cadencement de l'application (mesure du temps).
- ▷ Lancers de rayons.
- ▷ Interactions clavier/souris.
- ▷ Lecture des couleurs d'une image.
- ▷ ...

Le `Makefile` qui l'accompagne vous sera également utile pour vos développements. Vous pourrez constater sur certaines machines peu puissantes que l'utilisation d'images est beaucoup plus pénalisante que l'utilisation d'autres représentations. Évitez donc autant que possible d'y avoir recours et limitez vous à des images de petite taille (peu de pixels).

6 Travail demandé

Après avoir pris en main et manipulé l'exemple fourni, choisissez le programme graphique que vous souhaitez réaliser. Procédez à une analyse descendante des fonctionnalités en identifiant pour chaque traitement les entrés/sorties et les effets de bord. Après concertation avec l'enseignant, la réalisation pourra débuter. Elle donnera lieu à plusieurs versions successives dans des répertoires distincts (`V1`, `V2` ...). Vous devrez rendre un rapport relatant l'analyse et les détails de réalisation du projet.

7 Et pour ceux qui connaissent le C++

7.1 Contenu du fichier LibObjects2D/include/Object2D.h

```
#ifndef   _OBJECT2D_H_
#define   _OBJECT2D_H_

#include <vector>
#include <set>
#include <string>
#include <iostream>

#include "guiTrans.h"

#include "UtilObject2D.h"

using namespace std;

class Object2D
{
 friend ostream& operator<<(ostream& os, const Object2D& object2D);

 public :

/*-------- Constructor/Destructor ----------------------------------------*/

         Object2D(void);
         Object2D(const Object2D& object2D);
virtual ~Object2D(void);

         Object2D& operator=(const Object2D& object2D);

 // Test d'egalite uniquement sur la couleur et le type de forme:
 // noShape, point, text,
 friend   bool operator==(const Object2D& obj1, const Object2D& obj2);
 friend   bool operator!=(const Object2D& obj1, const Object2D& obj2);

/*-------- Location ------------------------------------------------------*/

      void   setLocation(double x, double y, double theta);
      void   getLocation(double& xOut, double& yOut, double& thetaOut) const;

      void   setX(double x);
      double getX(void) const;
      void   setY(double y);
      double getY(void) const;
      void   setTheta(double theta);
      double getTheta(void) const;

/*-------- Motion --------------------------------------------------------*/

      void   translate(double dx, double dy);
      void   rotate(double dTheta);
```

```
/*-------- Interaction ----------------------------------------------------*/

virtual void onKeyPress(const char * key);
virtual void onMouseDrag(double dx, double dy);

/*-------- Attachment -----------------------------------------------------*/

        void  attachTo(Object2D& object2D);
        bool  isAttachedTo(const Object2D& object2D) const;
        void  detachFrom(Object2D& object2D);
        void  detachFromAll(void);

/*-------- Detection ------------------------------------------------------*/

      bool   isInside(double x, double y) const;
      bool   intersectRay(double xRay, double yRay, double thetaRay,
                          double& xOut, double& yOut) const;
      Object2D * throwRay(double& xOut, double& yOut,
                          Object2D *tabObject2D[],
                          unsigned int nbObject2D) const;

      Object2D * viewFirstObject2D(double vision,
                                   double range,
                                   double turn=0.0) const;
      int        viewObject2D(Object2D*** tabObject2D,         // Version C
                              double vision,
                              double range,
                              double turn=0.0) const;
      int        viewObject2D(vector<Object2D*>& vectObject2D, // Version C++
                              double vision,
                              double range,
                              double turn=0.0) const;

/*-------- Transformation -------------------------------------------------*/

      void   globalToLocalPosition(double& xInOut, double& yInOut) const;
      void   localToGlobalPosition(double& xInOut, double& yInOut) const;

      double globalToLocalOrientation(double orientation) const;
      double localToGlobalOrientation(double orientation) const;

/*-------- Representation -------------------------------------------------*/

      void   setColor(const char * colorName);
      const  char * getColor(void) const;// Retourne NULL si invisible(noShape)

      void   setLayer(int layer);
      int    getLayer(void) const;

      void   noShape(void);
      void   point(void);
      void   text(const char * text);
      void   line(double length);
```

```
      void   square(double side, int filled);
      void   rectangle(double length, double width, int filled);
      void   polyline(unsigned int nbPoints, const double * xPoints,
                                             const double * yPoints);
      void   polygon(unsigned int nbPoints,  const double * xPoints,
                                             const double * yPoints,
                                             int filled);
      void   circle(double radius, int filled);

      int    image(const char * fileName,        // 0: failure, !=0: success
                   double pixelScale);
      int    getImagePixelAt(double x, double y, // 0: failure, !=0: success
                             int& redOut,
                             int& greenOut,
                             int& blueOut);
      int    getImagePixelNumberAt(double x,     //  >=0: pixel number
                                   double y);    //  <0: failure
      int    setImagePixelNumberAt(int pixel,    // 0: failure, !=0: success
                                   double x,
                                   double y);
      int    getImageNbColors(void);             // number of colors
      int    getImageRGB(int pixel,              // 0: failure, !=0: success
                         int& redOut,
                         int& greenOut,
                         int& blueOut);

/*-------- Data types ----------------------------------------------------*/

 private : // ...

 protected:

  virtual void display(ostream& os) const;
  virtual bool isEqualTo(const Object2D& object2D) const;

          // ...
};

/*-------- Graphical application template --------------------------------*/

extern void graphic_init(const char * windowName, const char * fontName);

extern void graphic_setWidth(int width);

extern void graphic_setHeight(int height);

extern void graphic_setBackground(const char * colorName);

extern void graphic_setViewPoint(double x, double y, double scale);

extern void graphic_getViewPoint(double * xOut, double * yOut,
                                 double *scaleOut);
```

```
extern void graphic_autoscale(void);

extern void graphic_run(void * userData);

extern void graphic_mainLoop(void * userData);

extern void graphic_keyPressCallback(Object2D * obj2d,
                                     const char * key,
                                     void * userData);

extern void graphic_mouseDragCallback(Object2D * obj2d,
                                      double dx, double dy,
                                      void * userData);

/*-------- Graphical application template --------*/
/*
   int
   main(void)
   {
   graphic_init("My Window","-*-helvetica-*-r-normal--14-*");
   graphic_setWidth(640);
   graphic_setHeight(480);
   ... application specific initializations ...
   graphic_run(myDataPointer);
   return(0);
   }

   void
   graphic_mainLoop(void * userData)
   {
   ...
   }

   void
   graphic_keyPressCallback(Object2D * obj2d,
                            const char * key,
                            void * userData)
   {
   ...
   }

   void
   graphic_mouseDragCallback(Object2D * obj2d,
                             double dx,
                             double dy,
                             void * userData)
   {
   ...
   }
*/
#endif // _OBJECT2D_H_
```

7.2 Contenu du fichier `LibObjects2D/include/UtilObject2D.h`

```
#ifndef _UTILOBJECT2D_H_
#define _UTILOBJECT2D_H_

#include <math.h>

void cartesianToPolar(double x,
                      double y,
                      double * distanceOut,
                      double * angleOut);

void polarToCartesian(double distance,
                      double angle,
                      double * xOut,
                      double * yOut);

double cartToDist(double x,double y);

double cartToAngle(double x,double y);

double betweenPi(double a);

#endif // _UTILOBJECT2D_H_
```

8 Installation

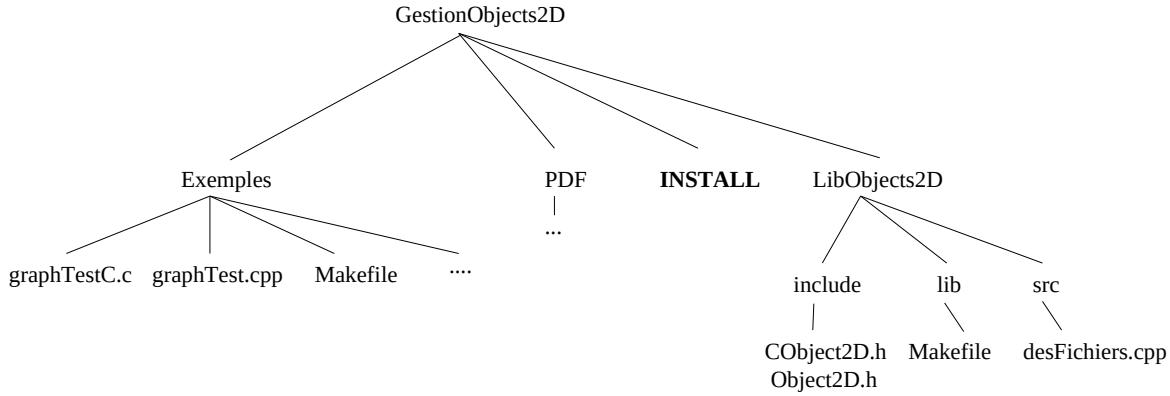

Figure 4: Description de l'arborescence de la bibliothèque

Aller dans le répertoire `LibObjects2D/lib` et faire `$ make`
Une bibliothèque `libObjects2D.a` est alors placée dans le répertoire `LibObjects2D/lib`
Des exemples de programmes sont disponibles dans le répertoire `Exemples`.

... mais on peut faire plus simple !

En étant dans le répertoire `GestionObjects2D`, faire tout simplement `$ ./INSTALL`
Il faut ensuite aller dans les répertoires avec les divers exemples et faire `$ make`