Bibliothèque de gestion d'agents 2D (Héritage multiple d'`Agent` et d'`Object2D`)

1 La classe `Agent2D`

La classe `Agent2D` décrite dans ce document correspond à une classe obtenue par héritage multiple entre la classe `Agent` (voir le répertoire `GestionAgents`) et la classe `Object2D` (voir le répertoire `GestionObjects2D`).

Le lecteur est donc encouragé à consulter les deux documents concernant la gestion d'`Agent` et la gestion d'`Object2D`:

- ▷ Gestion d'`Agent` : `GestionAgents/PDF/MoRis.pdf`
- ▷ Gestion d'`Object2D` : `GestionObject2D/PDF/jeuxInteractifs.pdf`

Ainsi, un `Agent2D` est un `Object2D` avec des fonctionnalités d'un `Agent`.

1.1 Rappels sur les `Object2D`

Un `Agent2D` est donc un `Object2D`, il a par conséquent une forme (décrivant un objet graphique 2D), une position et une orientation dans un repère global. Il est situé dans un plan (x, y, θ) structuré en couches (avant/arrière-plan).

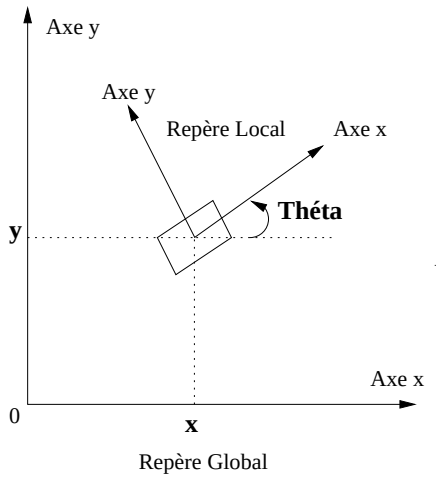

Figure 1: Description du repère global et du repère local associé à un objet graphique

1.2 Rappels sur les `Agent`

Un `Agent2D` est donc un `Agent`, il a par conséquent des possibilités de communication par `Message` et possède une méthode `live(double dt)` permettant de décrire son comportement.

Les `Agent2D` sont donc ordonnancés par un `Scheduler`.

2 Particularités de la classe `Agent2D`

Dans cette section, nous décrivons les fonctionnalités qui sont propres aux `Agent2D` (i.e. fonctionnalités ne se trouvant ni dans la classe `Agent`, ni dans la classe `Object2D`).

Rappels :

- ▷ Un `Agent2D` a donc une position et une orientation puisqu'un `Agent2D` est un `Object2D`.
- ▷ Un `Agent2D` a des capacités de détection/perception puisque c'est un `Object2D`. Mais attention, un `Agent2D` a ses propres méthodes de détection/perception...
- ▷ Les fonctionnalités concernant la gestion d'`Agent` permettent de connaître tous les `Agent` d'un certain type (et les types dérivés)... voir `getAllAgents`

  Cela peut-être utile pour qu'un `Agent2D` détecte/perçoive les autres `Agent2D`... et éventuellement seulement ceux d'un certain type.

2.1 Vitesses et accélérations

La classe `Agent2D` permet de gérer les notions de vitesses (linéaire et angulaire) et d'accélérations (linéaire et angulaire).

Ainsi, en fonction du temps qui passe (`dt`), il est possible de calculer la nouvelle position/orientation d'un `Agent2D` connaissant son ancienne position/orientation et ses vitesses (linéaires et angulaires). Les nouvelles vitesses (linéaire et angulaire) peuvent également être calculées grâce aux accélérations (linéaire et angulaire).

Remarques :

- ▷ Afin de connaître `dt` et que celui-ci ait un sens, l'ordonnanceur des `Agent` doit être en mode temps réel (voir `Scheduler::setRealTimeMode`).
- ▷ Les **vitesses et accélérations linéaires** sont considérées comme ayant un sens **dans le repère local** de l'`Agent2D`.

2.1.1 Gestion des vitesses (linéaire et angulaire)

Dans la classe `Agent2D`, on trouve les méthodes suivantes :

- ▷ `double getLinearVelocityX(void) const;`
- ▷ `double getLinearVelocityY(void) const;`
- ▷ `double getAngularVelocity(void) const;`
- ▷
```
void   getVelocity(double& xVelocity,double& yVelocity,
                   double& thetaVelocity) const;
```
- ▷ `void   setLinearVelocityX(double xVelocity);`
- ▷ `void   setLinearVelocityY(double yVelocity);`
- ▷ `void   setAngularVelocity(double thetaVelocity);`
- ▷
```
void   setVelocity(double xVelocity,double yVelocity,
                   double thetaVelocity);
```

2.1.2 Gestion des accélérations (linéaire et angulaire)

Dans la classe `Agent2D`, on trouve les méthodes suivantes :

- ▷ `double getLinearAccelerationX(void) const;`
- ▷ `double getLinearAccelerationY(void) const;`
- ▷ `double getAngularAcceleration(void) const;`
- ▷ `void getAcceleration(double& xAcceleration,double& yAcceleration, double& thetaAcceleration) const;`
- ▷ `void setLinearAccelerationX(double xAcceleration);`
- ▷ `void setLinearAccelerationY(double yAcceleration);`
- ▷ `void setAngularAcceleration(double thetaAcceleration);`
- ▷ `void setAcceleration(double xAcceleration,double yAcceleration, double thetaAcceleration);`

2.1.3 Calcul cinématique : `void Kinematic(double dt);`

Dans la classe `Agent2D` a été définie une méthode prenant en charge la cinématique :

```
void Kinematic(double dt);
```

Le calcul cinématique effectué permet de gérer le déplacement d'un `Agent2D` en fonction de sa position, de ses vitesses (linéaire et angulaire) et de ses accélérations (linéaire et angulaire)... et bien sûr du temps qui passe `dt` !

Remarques :

- ▷ Afin de connaître `dt` et que celui-ci ait un sens, l'ordonnanceur des `Agent` doit être en mode temps réel (voir `Scheduler::setRealTimeMode`).
- ▷ La méthode `live(double dt);` des `Agent2D` fait appel à cette méthode `Kinematic`.

```
void Agent2D::live(double dt)
{
 (void)dt; // Pour eviter un warning si pas utilise

 // "Comportement" d'un Agent de la classe Agent2D

 if (dt!=0.0) Kinematic(dt);
}
```

- ▷ Si par héritage vous dérivez de la classe `Agent2D`, il faudra penser à faire appel à cette méthode `Kinematic` dans méthode `live` de la classe dérivée.

  Et c'est presque magique! En effet, le fait d'appeler la méthode `Kinematic` permet de voir un `Agent2D` se déplacer "tout seul"... en fonction du temps qui passe `dt`, de ses vitesses (linéaire et angulaire) et de ses accélérations (linéaire et angulaire).

2.2 Détections/Perceptions propres aux `Agent2D`

Un `Agent2D` a des capacités de détection/perception puisque c'est un `Object2D`. Mais attention, un `Agent2D` a ses propres méthodes de détection/perception...

2.2.1 Méthodes de détection/perceptions de représentations graphiques d'`Agent2D`

Dans la classe `Agent2D`, on trouve les méthodes suivantes :

▷
```
bool /* true: inside, false: outside */
isInside(double x, double y) const;
```

- ◇ Retourne `true` si le point (x, y) est situé à l'intérieur de la représentation de l'`Agent2D *this`. Retourne `false` sinon.
- ◇ `x` et `y` sont exprimés dans le repère global.
- ◇ Cette méthode `isInside` **correspond** à celle définie dans la classe `Object2D`.

▷
```
bool /* true: intersection found, false: no intersection found */
intersectRay(double xRay, double yRay, double thetaRay,
             double& xOut, double& yOut) const;
```

- ◇ Retourne `true` si la représentation de l'`Agent2D *this` est intersectée par la demi-droite issue de $(xRay, yRay)$ et orientée selon $thetaRay$. Retourne `false` sinon.
- ◇ `xOut` et `yOut` reçoivent alors le point d'intersection.
- ◇ `xRay`, `yRay`, `thetaRay`, `xOut` et `yOut` sont exprimés dans le repère global.
- ◇ Cette méthode `intersectRay` **correspond** à celle définie dans la classe `Object2D`.

▷
```
Agent2D * /* NULL: no intersection found, not NULL: intersection found */
throwRay(double& xOut, double& yOut,
         const vector<Agent2D*>& vectAgent2D) const;
```

- ◇ Lance un rayon devant l'`Agent2D *this` (via les coordonnées et l'axe de `*this`).
- ◇ Retourne l'`Agent2D` le plus proche intersecté par ce rayon (`Agent2D` contenu dans le vecteur `vectAgent2D`, vecteur d'`Agent2D*`).
- ◇ 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.
- ◇ Cette méthode `intersectRay` **ne correspond pas** à celle définie dans la classe `Object2D`... Elle est propre aux `Agent2D`.

2.2.2 Méthodes de détection/perception d'`Agent2D` dans un cône de vision

Contrairement aux méthodes de détection de représentations graphiques qui travaillent sur les formes 2D des `Agent2D`, les méthodes présentées dans cette sous-section travaillent sur les positions (`x`,`y`) des `Agent2D`.

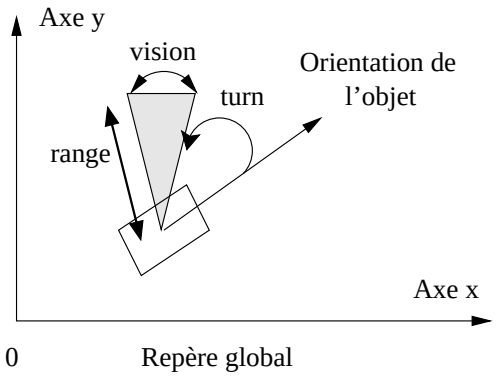

Figure 2: Cône de détection d'un `Agent2D` en fonction de son l'orientation et des paramètres `vision`, `range` et `turn`. Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞.

Ces méthodes utilisent la notion de cône de vision. Ainsi, pour un `Agent2D`, un cône de vision est déterminé par 3 réels `vision`, `range` et `turn`.

Dans la classe `Agent2D`, on trouve les méthodes suivantes :

▷
```
Agent2D * /* NULL: no Agent2D found, not NULL: an Agent2D */
viewFirst(string aClass,
          double vision, double range, double turn) const;
```

- ◇ Retourne l'`Agent2D` le plus proche se trouvant dans le cône de vision de l'`Agent2D *this`, cône de vision déterminé par `vision`, `range` et `turn`.
- ◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞
- ◇ Seuls les `Agent2D` de la classe `aClass` sont perçus...
  En interne, il y a un `getAllAgents` !

▷
```
int
view(string aClass,
     vector<Agent2D*>& vectAgent2D,
     double vision, double range, double turn) const;
```

- ◇ Retourne le nombre d'`Agent2D` se trouvant dans le cône de vision de l'`Agent2D *this`, cône de vision déterminé par `vision`, `range` et `turn`. A la sortie de la méthode, les `Agent2D` se trouvant dans ce cône de vision sont disponibles dans le vecteur passé par référence (`vectAgent2D`), vecteur d'`Agent2D*`.
- ◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞
- ◇ Seuls les `Agent2D` de la classe `aClass` sont perçus...
  En interne, il y a un `getAllAgents` !

▷
```
Agent2D * /* NULL: no Agent2D found, not NULL: an Agent2D */
viewFirstAgent2D(double vision, double range, double turn) const;
```

- ◇ Revient à faire `viewFirst("Agent2D",vision,range,turn);`

▷
```
int
viewAgent2D(vector<Agent2D*>& vectAgent2D,
            double vision, double range, double turn) const;
```

- ◇ Revient à faire `view("Agent2D",vectAgent2D,vision,range,turn);`

2.3 Pour mémoire : accès aux méthodes de détection/perception de la classe `Object2D`

Normalement, vous n'avez pas à utiliser ces méthodes puisqu'elles sont disponibles (et spécialisées) dans la classe `Agent2D`.

2.3.1 Méthodes de détection/perception de représentations graphiques d'`Object2D`

Pour mémoire, dans la classe `Object2D`, on trouve les méthodes suivantes :

▷
```
bool /* true: inside, false: outside */
isInside(double x, double y) const;
```

- ◇ Retourne `true` si le point (x, y) est situé à l'intérieur de la représentation de l'`Object2D *this`. Retourne `false` sinon.
- ◇ `x` et `y` sont exprimés dans le repère global.
- ◇ Cette méthode `isInside` **correspond** à celle définie dans la classe `Agent2D`.

▷
```
bool /* true: intersection found, false: no intersection found */
intersectRay(double xRay, double yRay, double thetaRay,
             double& xOut, double& yOut) const;
```

- ◇ Retourne `true` si la représentation de l'`Object2D *this` est intersectée par la demi-droite issue de $(xRay, yRay)$ et orientée selon $thetaRay$. Retourne `false` sinon.
- ◇ `xOut` et `yOut` reçoivent alors le point d'intersection.
- ◇ `xRay`, `yRay`, `thetaRay`, `xOut` et `yOut` sont exprimés dans le repère global.
- ◇ Cette méthode `intersectRay` **correspond** à celle définie dans la classe `Agent2D`.

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

- ◇ Lance un rayon devant l'`Object2D *this` (via les coordonnées et l'axe de `*this`).
- ◇ Retourne l'`Object2D` le plus proche intersecté par ce rayon (`Object2D` contenu dans le tableau `tabObject2D`, tableau de `nbObject2D` éléments de type `Object2D*`).
- ◇ 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.
- ◇ Cette méthode `intersectRay` **ne correspond pas** à celle définie dans la classe `Agent2D`... Elle est propre aux `Object2D`.

2.3.2 Méthodes de détection/perception d'`Object2D` 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`, un cône de vision est déterminé par 3 réels `vision`, `range` et `turn`.

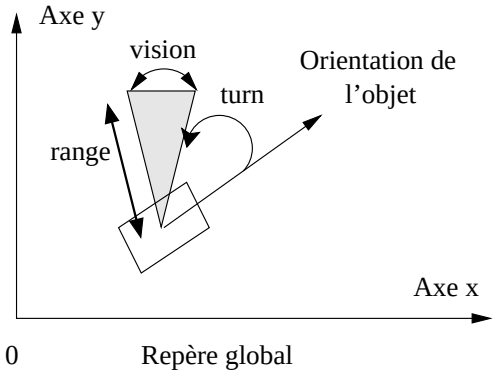

Figure 3: Cône de détection d'un object en fonction de son l'orientation et des paramètres `vision`, `range` et `turn`. Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞.

Pour mémoire, dans la classe `Object2D`, on trouve les méthodes suivantes :

▷
```
Object2D * /* NULL: no Object2D found, not NULL: an Object2D */
viewFirstObject2D(double vision, double range, double turn) const;
```

- ◇ Retourne l'`Object2D` le plus proche se trouvant dans le cône de vision de l'`Object2D *this`, cône de vision déterminé par `vision`, `range` et `turn`.
- ◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞

▷
```
int
viewObject2D(vector<Object2D*>& vectObject2D,
             double vision, double range, double turn) const;
```

- ◇ Retourne le nombre d'`Object2D` se trouvant dans le cône de vision de l'`Object2D *this`, cône de vision déterminé par `vision`, `range` et `turn`. A la sortie de la méthode, les `Object2D` se trouvant dans ce cône de vision sont disponibles dans le vecteur passé par référence (`vectObject2D`), vecteur d'`Object2D*`.
- ◇ Si `range` est égal à `0.0` $\Longrightarrow$ range : ∞

2.3.3 Utilisation : attention à l'héritage multiple !

Si vous décidez d'utiliser ces méthodes de la classe `Object2D`, il faut faire attention à l'héritage multiple ayant permis d'obtenir la classe `Agent2D` à partir de la classe `Agent` et de la classe `Object2D`.

Soit l'exemple suivant :

```
// La classe Petit heritant de la classe Agent2D
// La classe Poisson heritant de la classe Agent2D
.....
vector<Agent*> v;
getAllAgents("Petit",v);
int nbAgents=v.size();

Object2D* unObject2D = (Agent2D*)getAgent("Poisson.1");

Object2D** tab = (Object2D**)malloc(nbAgents*sizeof(Object2D*));

// v : vecteur d'Agent*
// tab : tableau d'Object2D*
for(int i=0;i<nbAgents;i++)
{
 tab[i]=(Agent2D*)v[i]; // Ecrire tab[i]=(Object2D*)v[i]; est une erreur !
}

double x,y;
Object2D* objFound=unObject2D->throwRay(x,y,tab,nbAgents);

if (objFound!=NULL) objFound->setColor("yellow");

free(tab);
.....
```

Sur cet exemple, nous constatons qu'il n'est pas possible, à cause de l'héritage multiple, de passer d'un `Agent*` à un `Object2D*` directement... Il faut passer par `Agent2D*` !

$\Longrightarrow$ Le mieux est certainement d'utiliser les méthodes de détection/perception de la classe `Agent2D` !

3 Initialisation, activation et paramétrage de l'application graphique

En plus de l'interface de la classe `Agent2D`, le fichier

`GestionAgents2D/LibAgents2D/include/Agent2D.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 `Agent2D`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 `Agent2D`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 le cœur de l'application.

◇ **Vous devez définir cette fonction !**

◇ Le paramètre `userData` est celui qui a été transmis à `graphic_run()`.

▷ `void graphic_keyPressCallback(Agent2D * agt2d, const char * key, void * userData);`

- ◇ Cette fonction est appelée lorsqu'une touche du clavier est enfoncée.
- ◇ **Vous devez définir cette fonction !**
- ◇ Si `agt2d` est non nul, il s'agit d'un objet graphique qui était sélectionné lors de l'appui sur la touche.
- ◇ Si `agt2d` 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 * agt2d, double dx, double dy, void * userData);`

- ◇ Cette fonction est appelée lorsqu'un mouvement de souris est appliqué à un `Agent2D`.
- ◇ **Vous devez définir cette fonction !**
- ◇ `agt2d` 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 `Agent2D` 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 Programme minimal

Cette section présente un programme minimal permettant de gérer des `Agent2D` : gestion d'un ordonnanceur + application graphique (voir `GestionAgents2D/Exemples/prgMinimal`)

```
#ifndef _APPLIDATA_H           /* Fichier : AppliData.h */
#define _APPLIDATA_H

#include "Agent2D.h"

typedef struct
{
Scheduler* sched;
Agent2D*   obj;
int        autoscale;
int        pause;
} AppliData;

#endif // _APPLIDATA_H
```

```
#include "Agent2D.h"            /* Fichier : prg.cpp */

/*------------------------------ Application --------------------------------*/
/*---- application specific data ----*/

#include "AppliData.h"

void deleteAppliData(AppliData * data)
{
 delete(data->obj);
 delete(data->sched);
 delete(data);
}

void help(void)
{
 cout << "*******************************************"<< endl;
 cout << "help"                                       << endl;
 cout << "----"                                       << endl;
 cout << "Actions clavier AVEC objet selectionne:"    << endl;
 cout << "- p : obtenir le nom d'un Agent2D"          << endl;
 cout << "Actions clavier SANS objet selectionne:"    << endl;
 cout << "- h : help"                                 << endl;
 cout << "- a : autoscale (oui/non)"                  << endl;
 cout << "-' ': pause     (oui/non)"                  << endl;
 cout << "- q : quitter"                              << endl;
 cout << "*******************************************"<< endl;
}

int main(void)
{
AppliData * data;

/*---- initialize graphic window ----*/

graphic_init("My Window","-*-helvetica-*-r-normal--14-*");
graphic_setWidth(640);
graphic_setHeight(480);
graphic_setBackground("cyan4");

/*---- initialize application specific data ----*/

data=(AppliData *)malloc(sizeof(AppliData));
data->sched=new Scheduler;
data->sched->setRealTimeMode(true);

/*---- initialize Agent2D  ----*/

data->obj=new Agent2D();
data->obj->square(10,0);
data->obj->setAngularVelocity(1); // Vitesse angulaire

/*----- initialize specific data -----*/

data->autoscale=1;
data->pause=0;
```

```
/*---- run the graphic application ----*/

help();

graphic_run(data);
return(0);
}

void
graphic_mainLoop(void * userData)
{
AppliData * data=(AppliData *)userData;
if(!data->pause)
  {
   data->sched->cycle();   // Lancement d'un cycle de l'ordonnanceur
  }
if(data->autoscale)
  {
   graphic_autoscale();
  }
}

void
graphic_keyPressCallback(Agent2D * agt2d,
                         const char * key,
                         void * userData)
{
AppliData * data=(AppliData *)userData;
(void)data;

if (agt2d==NULL)         // Interaction clavier SANS objet selectionne
{
 if(!strcmp(key,"Left"))
  {
  graphic_mouseDragCallback(data->obj,
                            -0.5,0.0,userData); // simulate mouse drag
  }
 else if(!strcmp(key,"Right"))
  {
  graphic_mouseDragCallback(data->obj,
                            0.5,0.0,userData); // simulate mouse drag
  }
 else if(!strcmp(key,"h")||!strcmp(key,"H"))
  {
  help();
  }
 else if(!strcmp(key,"a")||!strcmp(key,"A"))
  {
  data->autoscale=1-data->autoscale;
  }
 else if(!strcmp(key," "))
  {
  data->pause=1-data->pause;
  }
```

```
 else if(!strcmp(key,"q")||!strcmp(key,"Q")||!strcmp(key,"\x1b")) // Esc
  {
  deleteAppliData(data);
  exit(0);
  }
 else { fprintf(stderr,"Viewer key <%s>\n",key); }
 return;
}
        //  ET donc ici, interaction clavier AVEC un objet selectionne
agt2d->onKeyPress(key);  // Par defaut : si p affiche le nom de l'Agent2D
}
void
graphic_mouseDragCallback(Agent2D * agt2d,
                          double dx,
                          double dy,
                          void * userData)
{
AppliData * data=(AppliData *)userData;
(void)data;
agt2d->onMouseDrag(dx,dy); // Par defaut : deplacement de l'Agent2D a la souris
}
```

5 Co–existence Agent2D/Object2D

Les `Agent2D` étant des objets graphiques avec un comportement, il est préférable de décrire les éléments de décor (i.e. sans comportement) à l'aide d'`Object2D`.

Cela peut poser quelques problèmes vis-à-vis des interactions avec la souris ou le clavier.

Reprenons le programme minimal décrit lors de la section précédente. Celui-ci doit être modifié de façon à gérer la co–existence entre les `Agent2D` et les `Object2D`.

Nous introduisons pour cela la notion de décor où, tous les `Object2D` doivent être ajouter afin de gérer correctement l'interaction souris/clavier.

Le fichier `decor.h` ci–après décrit le type `Decor`. Attention : vous n'avez pas a déclarer d'objet de ce type. Il en existe déjà un déclaré dans la bibliothèque !... et le fichier `decor.h` décrit les opérations possibles sur cet objet.

```
#ifndef  _DECOR_H_                      /* Fichier : decor.h */
#define  _DECOR_H_

#include <set>
#include "Object2D.h"

using namespace std;

typedef set<Object2D*> Decor;

extern Decor decor;       // Contient uniquement des Object2D pas des Agent2D !

extern void addDecor(Object2D* obj2d);    // Ajouter un element du decor
extern void removeDecor(Object2D* obj2d); // Enlever un element du decor
extern bool isInDecor(Object2D* obj2d); // Tester si un objet est dans le decor

#endif //_DECOR_H_
```

L'idée est ici d'ajouter des éléments au décor (tous les `Object2D` de l'environnement). Ensuite, lors d'une interaction, il suffit de tester si l'objet ayant subit l'interaction est dans le décor. S'il est dans le décor, c'est un `Object2D`. Sinon, c'est un `Agent2D`. Ce test doit être fait dans les fonctions `graphic_keyPressCallback` et `graphic_mouseDragCallback`.

Les modifications a apporter au programme minimal (décrit dans la section précédente) sont indiquées via des commentaires `// *** AJOUT ..!  *** .`

```
#ifndef _APPLIDATA_H          /* Fichier AppliData.h */
#define _APPLIDATA_H

#include "Agent2D.h"
#include "Object2D.h"

typedef struct
{
Scheduler* sched;
Agent2D*   agt;
Object2D*  obj;               // *** AJOUT ..! ***
int        autoscale;
int        pause;
} AppliData;
#endif // _APPLIDATA_H
```

Et maintenant le fichier `prg.cpp`

```
/*-------------------------------------------------------------------------
  prg.cpp
-------------------------------------------------------------------------*/

#include "Agent2D.h"

#include "Decor.h"            // *** AJOUT ..! ***
#include "Object2D.h"         // *** AJOUT ..! ***

/*-------- Application ----------------------------------------------------*/

/*---- application specific data ----*/

#include "AppliData.h"

void deleteAppliData(AppliData * data)
{
 removeDecor(data->obj);      // *** AJOUT ..! ***
 delete(data->obj);
 delete(data->agt);
 delete(data->sched);
 delete(data);
}
```

```
void help(void)
{
 cout << "*****************************************"<< endl;
 cout << "help"                                     << endl;
 cout << "----"                                     << endl;
 cout << "Actions clavier AVEC objet selectionne:"  << endl;
 cout << "- p : obtenir le nom d'un Agent2D"        << endl;
 cout << "Actions clavier SANS objet selectionne:"  << endl;
 cout << "- h : help"                               << endl;
 cout << "- a : autoscale (oui/non)"                << endl;
 cout << "-' ': pause     (oui/non)"                << endl;
 cout << "- q : quitter"                            << endl;
 cout << "*****************************************"<< endl;
}

int
main(void)
{
AppliData * data;

/*---- initialize graphic window ----*/

graphic_init("My Window","-*-helvetica-*-r-normal--14-*");
graphic_setWidth(640);
graphic_setHeight(480);
graphic_setBackground("cyan4");

/*---- initialize application specific data ----*/

data=(AppliData *)malloc(sizeof(AppliData));
data->sched=new Scheduler;
data->sched->setRealTimeMode(true);

/*---- initialize Agent2D  ----*/

data->agt=new Agent2D();
data->agt->square(10,0);
data->agt->setAngularVelocity(1);

data->obj=new Object2D();          // *** AJOUT ..! ***
addDecor(data->obj);               // *** AJOUT ..! ***
data->obj->square(15,0);           // *** AJOUT ..! ***
data->agt->attachTo(*data->obj);   // *** AJOUT ..! ***

/*----- initialize specific data -----*/

data->autoscale=1;
data->pause=0;

/*---- run the graphic application ----*/

help();

graphic_run(data);
return(0);
}
```

```
void
graphic_mainLoop(void * userData)
{
AppliData * data=(AppliData *)userData;
if(!data->pause)
  {
   data->sched->cycle(); // Lancement d'un cycle de l'ordonnanceur
  }
if(data->autoscale)
  {
   graphic_autoscale();
  }
}

void
graphic_keyPressCallback(Agent2D * agt2d,
                         const char * key,
                         void * userData)
{
AppliData * data=(AppliData *)userData;
(void)data;

if (agt2d==NULL)         // Interaction clavier SANS objet selectionne
{
 if(!strcmp(key,"Left"))
  {
  graphic_mouseDragCallback(data->agt,
                            -0.5,0.0,userData); // simulate mouse drag
  }
 else if(!strcmp(key,"Right"))
  {
  graphic_mouseDragCallback(data->agt,
                            0.5,0.0,userData); // simulate mouse drag
  }
 else if(!strcmp(key,"h")||!strcmp(key,"H"))
  {
  help();
  }
 else if(!strcmp(key,"a")||!strcmp(key,"A"))
  {
  data->autoscale=1-data->autoscale;
  }
 else if(!strcmp(key," "))
  {
  data->pause=1-data->pause;
  }
 else if(!strcmp(key,"q")||!strcmp(key,"Q")||!strcmp(key,"\x1b")) // Esc
  {
  deleteAppliData(data); exit(0);
  }
 else { fprintf(stderr,"Viewer key <%s>\n",key); }
 return;
}
```

```
        //  ET donc ici, interaction clavier AVEC un objet selectionne

if (isInDecor((Object2D*)agt2d))                // *** AJOUT ..! ***
{// Interaction avec les Object2D du decor      // *** AJOUT ..! ***
 Object2D* obj2d=(Object2D*)agt2d;              // *** AJOUT ..! ***
 obj2d->onKeyPress(key);                        // *** AJOUT ..! ***
 return;                                        // *** AJOUT ..! ***
}                                               // *** AJOUT ..! ***

agt2d->onKeyPress(key); // Par defaut : si p affiche le nom de l'Agent2D
}

void
graphic_mouseDragCallback(Agent2D * agt2d,
                          double dx,
                          double dy,
                          void * userData)
{
AppliData * data=(AppliData *)userData;
(void)data;

if (isInDecor((Object2D*)agt2d))                // *** AJOUT ..! ***
{// Interaction avec les Object2D du decor      // *** AJOUT ..! ***
 Object2D* obj2d=(Object2D*)agt2d;              // *** AJOUT ..! ***
 obj2d->onMouseDrag(dx,dy);                     // *** AJOUT ..! ***
 return;                                        // *** AJOUT ..! ***
}                                               // *** AJOUT ..! ***

agt2d->onMouseDrag(dx,dy); // Par defaut : deplacement de l'Agent2D a la souris
}

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

6 Interface public/protected de la classe Agent2D

Fichier : GestionAgents2D/LibAgents2D/include/Agent2D.h

```
class Agent2D : public Agent,
                public Object2D
{
  DEFCLASS(Agent2D)

  friend ostream& operator<<(ostream& os, const Agent2D& anA);

 public :

   // Allocateurs/Desallocateurs

            Agent2D(void);
            Agent2D(const Agent2D& anA);
            Agent2D& operator=(const Agent2D& anA);
   virtual ~Agent2D(void);
```

```
virtual  void live(double dt);
                                                  // Par defaut :
virtual  void onKeyPress(const char * key);       // si p affiche le nom
virtual  void onMouseDrag(double dx, double dy); // deplacement souris

// Comparaisons

friend  bool operator==(const Agent2D& anA1, const Agent2D& anA2);
friend  bool operator!=(const Agent2D& anA1, const Agent2D& anA2);

// Inspecteurs/modificateurs

   ////////////////////////////////////////////
   // En plus, ... Voir Agent.h et Object2D.h ...!
   ////////////////////////////////////////////

   // Gestion de la cinematique :
   // ---------------------------

   void   Kinematic(double dt);  // Modification de la position/orientation

   // Velocity: xVelocity, yVelocity, thetaVelocity
   // ---------------------------------------------

   double getLinearVelocityX(void) const;
   double getLinearVelocityY(void) const;
   double getAngularVelocity(void) const;
   void   getVelocity(double& xVelocity,double& yVelocity,
                      double& thetaVelocity) const;

   void   setLinearVelocityX(double xVelocity);
   void   setLinearVelocityY(double yVelocity);
   void   setAngularVelocity(double thetaVelocity);
   void   setVelocity(double xVelocity,double yVelocity,
                      double thetaVelocity);

   // Acceleration xAcceleration, yAcceleration, thetaAcceleration
   // ------------------------------------------------------------

   double getLinearAccelerationX(void) const;
   double getLinearAccelerationY(void) const;
   double getAngularAcceleration(void) const;
   void   getAcceleration(double& xAcceleration,double& yAcceleration,
                          double& thetaAcceleration) const;

   void   setLinearAccelerationX(double xAcceleration);
   void   setLinearAccelerationY(double yAcceleration);
   void   setAngularAcceleration(double thetaAcceleration);
   void   setAcceleration(double xAcceleration,double yAcceleration,
                          double thetaAcceleration);
```

```
   // Perception/Detection

   bool   isInside(double x, double y) const;
   bool   intersectRay(double xRay, double yRay, double thetaRay,
                       double& xOut, double& yOut) const;
   Agent2D * throwRay(double& xOut, double& yOut,
                      const vector<Agent2D*>& vectAgent2D) const;

   Agent2D * viewFirst(string aClass,
                       double vision, double range,
                       double turn=0.0) const;

   int       view(string aClass, vector<Agent2D*>& vectAgent2D,
                  double vision, double range,
                  double turn=0.0) const;

   Agent2D * viewFirstAgent2D(double vision, double range,
                              double turn=0.0) const;

   int       viewAgent2D(vector<Agent2D*>& vectAgent2D,
                         double vision, double range,
                         double turn=0.0) const;

 protected :

   // Methodes a appeler par une classe derivee

   // display: a appeler dans une classe derivee      // display est une
   virtual void display(ostream& os) const;           // methode appelee
                                                      // dans operator<<

   // isEqualTo: a appeler dans une classe derivee (dans operator==)
   virtual bool isEqualTo(const Agent2D& anA) 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(Agent2D * agt2d,
                                     const char * key,
                                     void * userData);

extern void graphic_mouseDragCallback(Agent2D * agt2d,
                                      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(Agent2D * agt2d,
                            const char * key,
                            void * userData)
   {
   ...
   }

   void
   graphic_mouseDragCallback(Agent2D * agt2d,
                             double dx,
                             double dy,
                             void * userData)
   {
   ...
   }
*/
```

7 Pour mémoire : Interface public/protected de la classe Agent

Fichier : GestionAgents/LibMoRis/include/Agent.h

Dans GestionAgents/LibMoRis/include,
il faut voir également la classe Scheduler et la classe Message !

```
typedef void (Agent::*liveMethodType)(double dt); // Pour get/setLiveMethod...

class Agent {
 DEFCLASS(Agent)
  friend ostream& operator<<(ostream& os, const Agent& anAgent);
 public:
   // Allocateurs/Desallocateurs
          Agent(void);
          Agent(const Agent& anAgent);
          Agent& operator=(const Agent& anAgent);
  virtual ~Agent(void);
  virtual  void live(double dt)              // dt en seconde : temps depuis
          {                                  // la derniere activation
           // Rien pour un Agent de base
           (void)dt;  // Pour eviter un warning
          }
          void suspend(void);
          void restart(void);
          bool isSuspended(void) const;
          void setLiveMethod(liveMethodType newLiveMethod); // Progr. avertis!
          liveMethodType getLiveMethod(void);               // Progr. avertis!
   // Comparaisons
  friend   bool operator==(const Agent& anAgent1, const Agent& anAgent2);
  friend   bool operator!=(const Agent& anAgent1, const Agent& anAgent2);
   // Inspecteurs
          string getName(void) const;  // N'a pas de sens dans le constructeur
          unsigned long getSuffix(void) const; // Idem : n'a pas de sens ...
          string getClass(void) const; // DEFCLASS: virtual getClassName ...
          bool   isA(string aClass) const;
   // Gestion des messages
          size_t   getNbMessages(void) const;
          Message* getNextMessage(void); // Le suivant
          void     clearMessageBox(void);

          void     setSensitivity(string aClass,bool yesNo);
                                                              // retourne
   virtual size_t   sendMessageTo(Message& aM,Agent *dest) const;// 1(ok),0(ko)
   virtual void     broadcastMessage(Message& aM) const;
```

```
 protected:

   // Methodes a appeler par une classe derivee

                               // Methode qui doit etre appelee dans le cons-
           void newAgent(void); // tructeur d'une classe derivee
                               // => Arbre d'heritage

           void newAgent(Agent* This); // Idem mais pour l'heritage multiple
                                       // ... pour programmeurs avertis !

   //  display a appeler dans une classe derivee       // display est une
   virtual void display(ostream& os) const;            // methode appelee
                                                       // dans operator<<

   //  isEqualTo a appeler dans une classe derivee     // isEqualTo est une
   virtual bool isEqualTo(const Agent& anAgent) const; // methode appelee
                                                       // dans operator==
};
```

8 Pour mémoire : Interface publique de la classe `Object2D`

Fichier : `GestionObjects2D/LibObjects2D/include/Object2D.h`

```
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);

 protected:

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

};
```

9 Installation

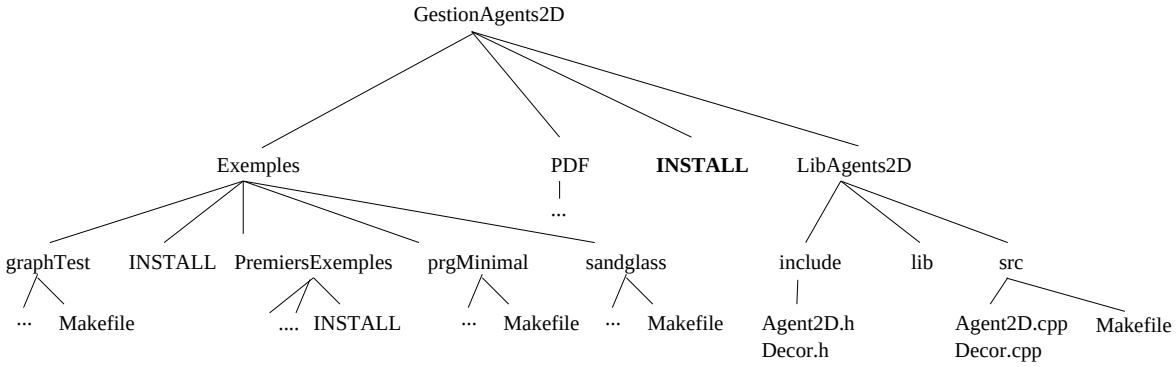

Figure 4: Description de l'arborescence de la bibliothèque

Aller dans le répertoire `LibAgents2D/src` et faire `$ make`
Une bibliothèque `libAgents2D.a` est alors placée dans le répertoire `LibAgents2D/lib`
Des exemples de programmes sont disponibles dans le répertoire `Exemples`.

... mais on peut faire plus simple !

En étant dans le répertoire `GestionAgents2D`, faire tout simplement `$ ./INSTALL`
Il faut ensuite aller dans les répertoires avec les divers exemples et faire `$ make`
...mais on peut aussi faire `$ ./INSTALL`