Interface des icobjs

```
public interface Icobj
{
    String name();

    double x(),y();
    double width(),height();

    void registerIn(IcobjWorkspace workspace);
    ...

    void setExternal(String s);
    String external();

    void setIcon(String file);
    void setColor(Color color);

    void grow(int w, int h);
    void resize(int width, int height);

    boolean inside(double x, double y, double d);
    ...
    double dist(Icobj other);

    void whenRegistered(Program whenRegistered);
    Program whenRegistered();
    void setBehavior(Program behavior);
    Program behavior();

    void setImmediate();
    boolean immediate();

    void activate();
```

Interface des icobjs (suite)

```
    boolean mouseDown(java.awt.Event evt, int posX, int posY);
    boolean mouseDrag(java.awt.Event evt, int x, int y);
    ...

    boolean acceptFocus();
    void gainFocus();
    void looseFocus();
    boolean keyDown(java.awt.Event evt, int key);

    void paint(Graphics g,int originX,int originY);

    void translate(double dx, double dy);
}
```

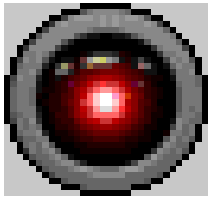
Workspace

```
public interface IcobjWorkspace
extends MouseListener,MouseMotionListener,FocusListener, KeyListener
{
    void add(Program prog);
    void generate(String event);

    void registerIcobj(Icobj icobj);
    void destroyIcobj(Icobj icobj);
    Stack icobjStack();
    Vector listOfOverlappingIcobj(Icobj icobj);

    void needToRepaint();
    void update(Graphics g);

    Image getImage(String fileName);

    void init();
    void start();
    void stop();
    void destroy();

    ...
}
```

Applet réactive

```
public class ReactiveApplet implements Runnable
{
    protected Applet applet;
    protected Machine machine;
    protected boolean running = false, stopped = false;

    public ReactiveApplet(Applet a){ machine = Jr.SyncMachine(); applet = a;  }

    public void add (Program prog){ machine.add(prog); }
    public void generate(Identifier id){ machine.generate(id); }

    protected Thread thread = new Thread(this);
    public void init(){ thread.start(); }
    public synchronized void start(){ running = true; notify(); }
    public void stop(){ running = false; }
    public void destroy(){ stopped = true; }
    protected int ips = 25; // fixed number of instants per second

    public synchronized void run(){
       while(!stopped){
           while (running){
              long time = System.currentTimeMillis();
              machine.react();
              applet.update(applet.getGraphics());
              try{
                 // This is the only sleep needed !
                 Thread.sleep((1000/ips)-(System.currentTimeMillis()-time));
              }catch(Exception e){}
           }
           try{ wait(); }catch(Exception e){}
       }
    }
}
```

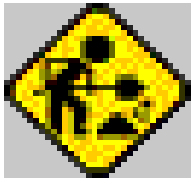
Icobj Detect



```
public class DetectIcobj extends IcobjImplem
{
   public DetectIcobj(int x, int y){
      super(x,y);
      String killName = name+"_kill";
       setBehavior(
           Jr.Local(killName,
             Jr.Until(killName,
              Jr.Loop(
                Jr.Seq(Jr.If(new Detect(),Jr.Generate(killName)),Jr.Stop()))))));

      setIcon("icones/HAL.gif");
      resize(32,32);
   }
}
```

```
public class Detect implements BooleanWrapper
{
    public boolean evaluate(Environment env){
       Icobj implicit = (Icobj)env.linkedObject();
       return implicit.workspace().listOfOverlappingIcobj(implicit).size() != 0;
    }
}
```

Icobj Constructor



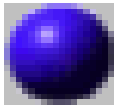
```
public class IcobjConstructor extends IcobjImplem
{
    protected boolean looping = false;
    protected int counter = -1;
    public Icobj result;
    protected Vector globalSeq = new Vector();
    public void createResult(){ result = new IcobjImplem(0,0); }

    public IcobjConstructor(int x, int y){
        super(x,y);
        setBehavior(
           Jr.Loop(Jr.Seq(Jr.Await(name),Jr.Seq(Jr.Atom(new Build()),Jr.Stop()))));
    }
    public void build(){
        Enumeration overlapped =(workspace.listOfOverlappingIcobj(this)).elements();
        int numberOfComponents = 0;
        if(!overlapped.hasMoreElements()){  terminateConstruction(); return; }
        Program newComponent = Jr.Nothing();
        while(overlapped.hasMoreElements()){
              Icobj icobj = (Icobj)overlapped.nextElement();
              // tag processing
              if (icobj instanceof LoopTag){ looping = true; continue; }
              ...
              numberOfComponents++;
              Program body = icobj.behavior().copy(); // get a copy of the behaviour
              newComponent = Jr.Par(newComponent,body);
        }
        globalSeq.addElement(newComponent);
    }
```

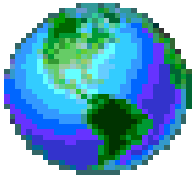
Icobj Constructor (suite)

```
    public void terminateConstruction(){
        createResult();
        Program newInst = Jr.Nothing();
        Enumeration list = globalSeq.elements();
        while(list.hasMoreElements()){
           newInst = Jr.Seq(newInst,(Program)list.nextElement());
        }
        if (looping){
            if (counter == -1) newInst = Jr.Loop(Jr.Seq(newInst,Jr.Stop()));
            else newInst = Jr.Repeat(counter,newInst);
        }
        result.whenRegistered(Jr.Link(result,newInst));
        result.setBehavior(newInst);
        result.registerIn(workspace);
        ...
    }
}
```

Icobj inertiel

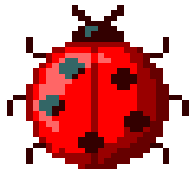
```
public interface InertialIcobj extends Icobj
{
   double speedx(),speedy();
   double oldx(),oldy();
   double mass();
   double absorbtion();

   void setSpeed(double speedx, double speedy);
   void setMass(double m);
   void setAbsorbtion(double d);

   void setInertia(boolean b);
   void inertiaAction();

   void setAttraction(boolean b);
   void attractAction();
}
```





```
 public void inertiaAction(){
    if (inertia == false) return;
    translate(speedx*(1-absorbtion),speedy*(1-absorbtion));
    speedx = x-oldx; speedy = y-oldy;
    oldx = x; oldy = y;
 }
```

Icobj de poursuite



```
protected Icobj closest(){
   double min = 100000;
   Icobj target = null;
   Enumeration list = workspace.icobjStack().elements();
   while(list.hasMoreElements()){
     Icobj other = (Icobj)list.nextElement();
     if (other == this) continue;
     if (!tobeconsidered(other) || notTobeconsidered(other)) continue;
     double d = dist(other);
     if (d<min){ min = d; target = other; }
   }
   return target;
}
```

```
public void inertiaAway(){
    Icobj target = closest();
    if (target != null && dist(target) < visibleDist){ moveAway(target); }
    super.inertia();
}
```

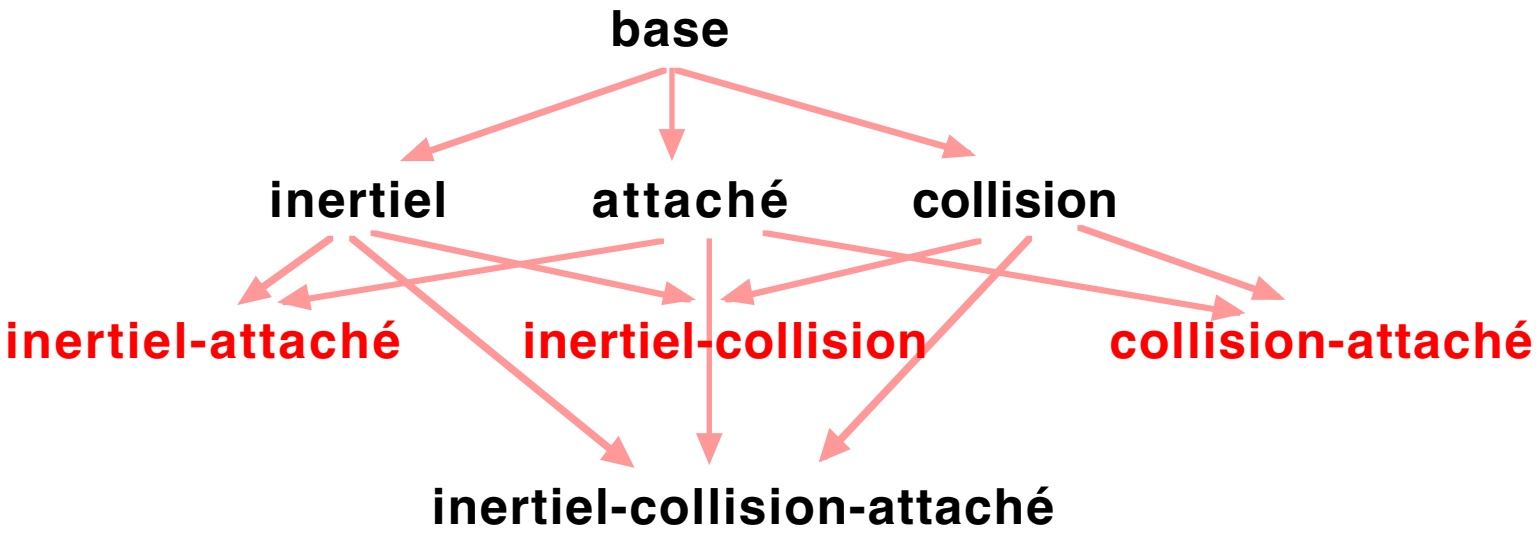
Hiérarchie d'héritage

lcobj de base
lcobj inertiel
lcobj pouvant être attaché
lcobj traitant les collisions
lcobj de poursuite
...

un constructeur nécessaire pour chaque !

- lcobj de poursuite ne pouvant pas être attaché ?
- lcobj de base traitant les collisions avec les autres ?
- Approche orientée-objets ... ?

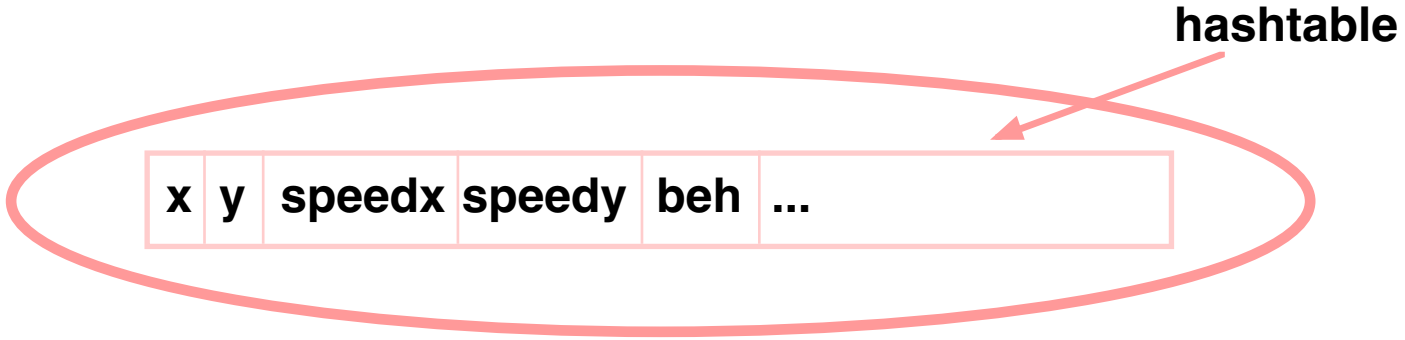
Héritage multiple ?



+ poursuite !

x,y dans inertiel et x,y dans collision ; quel système de coordonnées utiliser dans inertiel-collision ?

Autre solution



Création dynamique de champs

Atoms : inertia, collide, goAway, goCloser, ...

Toutes les interactions (souris, destruction,...) traitées par des événements avec valeurs

Primitive pour traiter toutes les valeurs générées dans l'instant (gravité,...)

Nouvelles implémentations en cours des icobjs