MobiNet

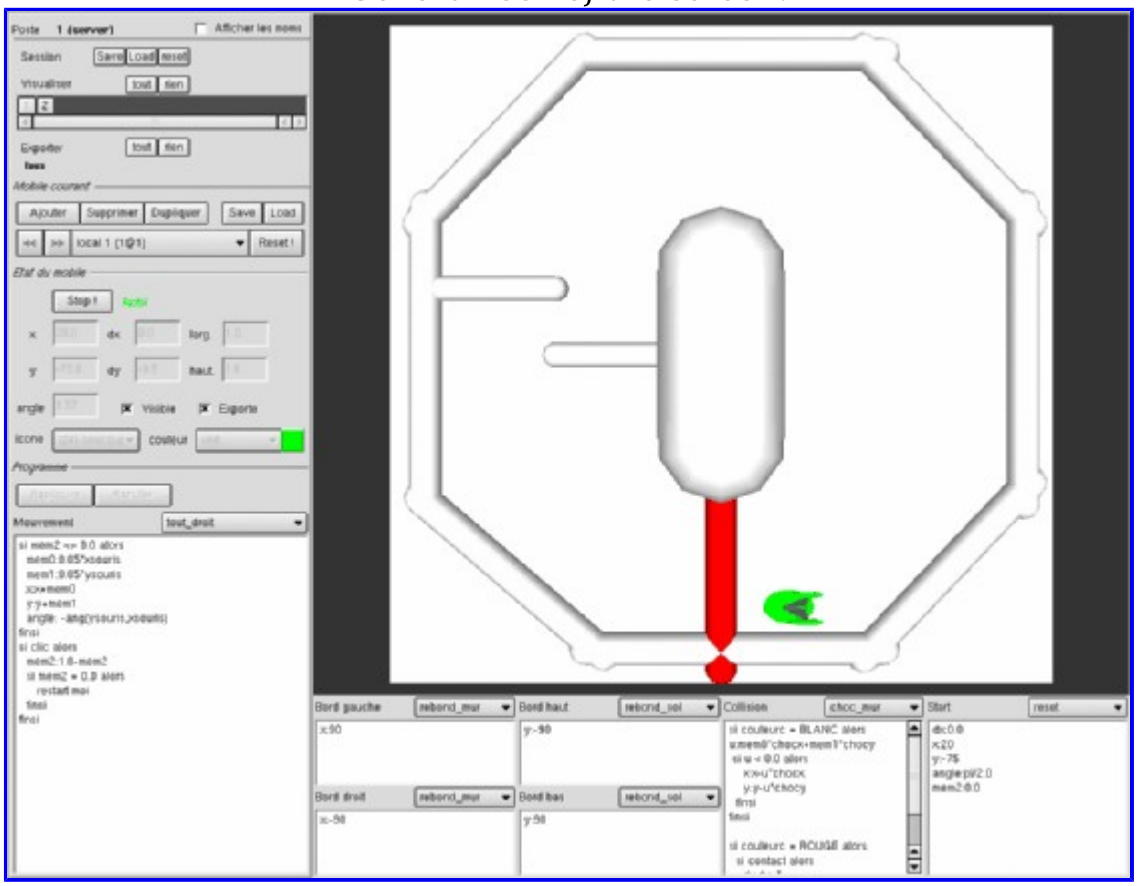
Networked platform for mobile objects programming. (simulation, games, graphics, maths-physics, ...)

http://www-evasion.imag.fr/mobinet

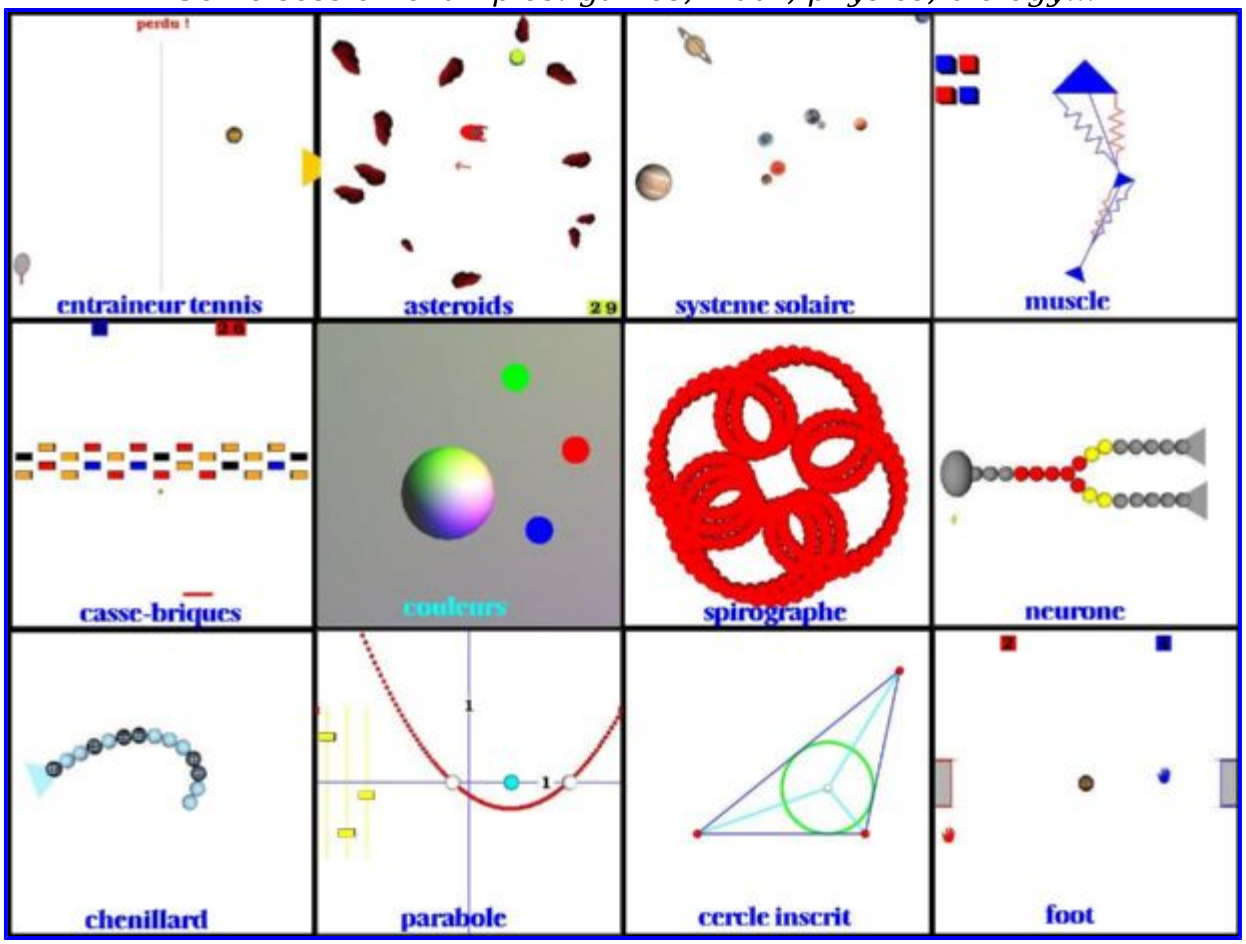
MobiNet is a software for programming the behavior of animated mobiles using an intuitive interface and language. MobiNet is totally interactive, easing learning and tuning of mobiles. Moreover, MobiNet is built to work in network (but it can also be used on a standalone computer).
MobiNet is especially interesting as a pedagogic platform, for initiating students (from high school to university) to games programming, or more generally to provide them with a concrete intuitive and fun version of the notions seen in math and physics course.

- **Chapter 1:** The MobiNet interface.
- **Chapter 2:** The MobiNet language (PDF , PS).
- **Chapter 3:** Tutorial : exercice sheets (in French) (PDF , PS).

General look of the screen.



>Some session examples: games, math, physics, biology...



- **Contact:**

  Do you want to register to our mailing-list ? (in French) Contact-us !

  Précisez si vous souhaitez:

  - la mailing-liste d'**annonce** des nouvelles versions et autres info techniques.
  - la mailing-liste du **forum** (discussion entre utilisateurs et developpeurs).
  - la mailing-liste d'expérimentation **pédagogique** (enseignants).

  E-mail : *mobinet@imag.fr*

  Adresse : *iMAGIS / GRAVIR - INRIA ZIRST, 655 avenue de l'Europe*
  38330 Montbonnot Saint Martin - FRANCE
  Fax : *+33 (0)4 76 61 54 40*

MobiNet - The interface (v1.1)

Networked platform for mobiles programming.

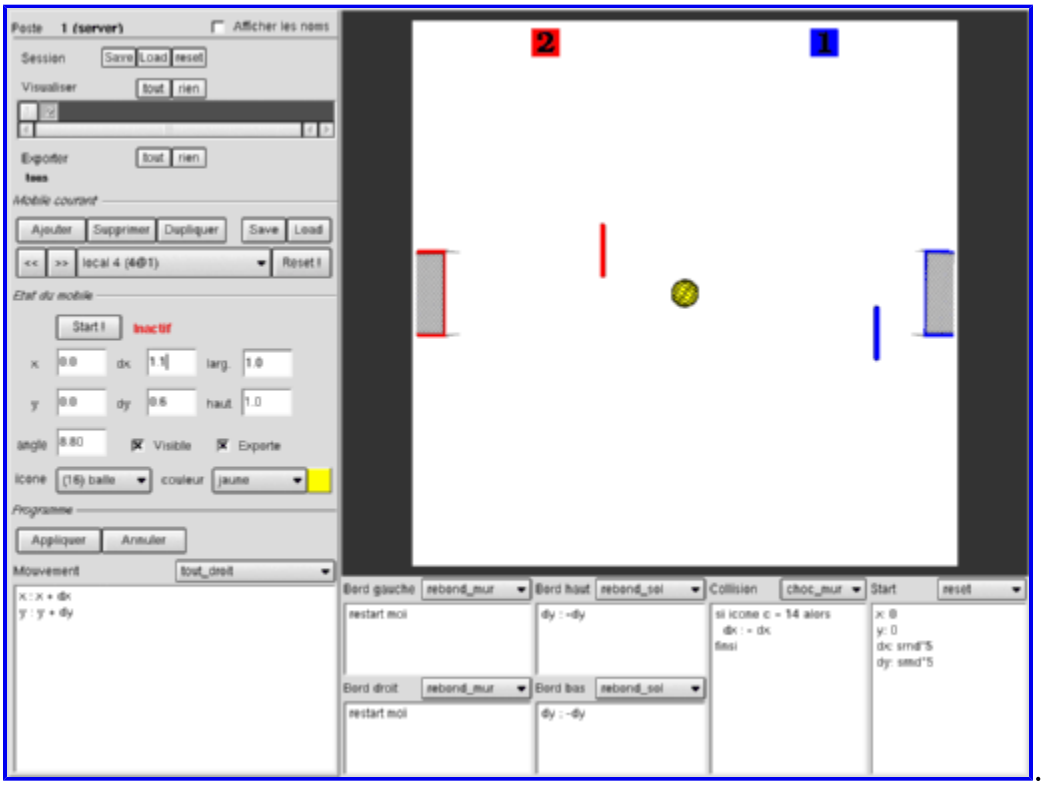
Back to MobiNet home page.

*In MobiNet, one programs the behavior of various **mobiles**: These are the various visible objects (whether they move or not!). The interface allow the tuning of every behaviors of the current mobile: aspect, motion, colliding rules with borders or other mobiles... (then one switch for a mobile to the other).*

General aspect of the screen (click for zooming). The various areas are described below.

Here, we programmed 7 mobiles: the ball, the racket, the goals, the counters.

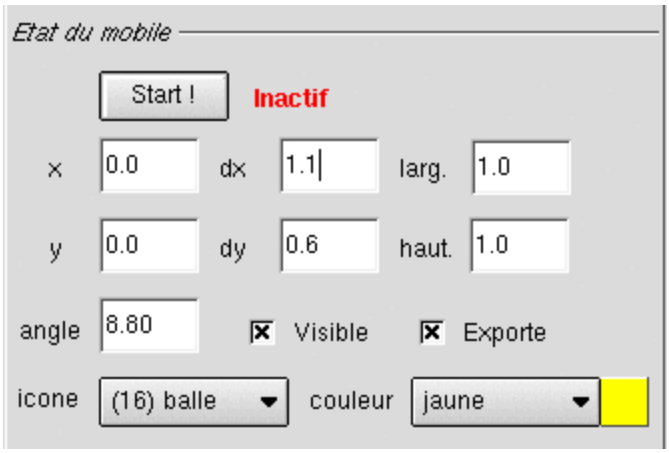
- The ball follows a straight line trajectory, and bounce on borders and on the rackets.
*- The rackets follow the mouse vertically. In facts, one of the rackets (and the corresponding goal and counter) is managed by **another computer**: here, two users share their mobiles on the network. One manage the reds and the ball, the other manage the blues.*
- The goals account for collisions with the ball, and trigger the opposite counter.
- The counters increment when triggered by the goals.



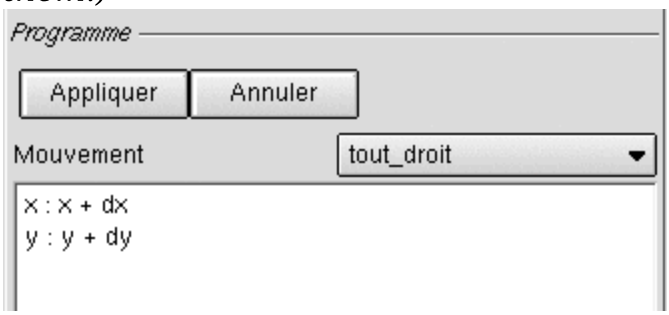
By clicking on `display names', one can see the mobiles number (`local n` *when managed on the computer,* `n@m` *when managed by computer m*) and the

coordinate system (*-100 to 100 for x and y*).

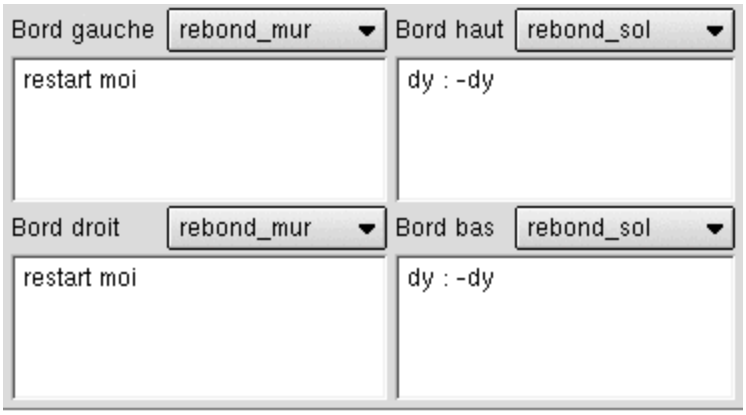
- Parameters describing the current mobile (*here, the ball*): position, dimension, icon, color, orientation... (*moreover, we will use dx,dy to encode the motion direction.*)
NB: in fact all is number (*e.g. the ball icon is number 16*), which allows calculation.



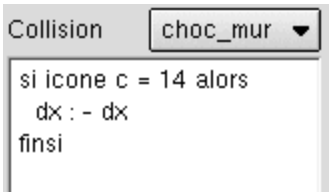
- Programming the motion: it consists in describing the changes to be done between 2 images (about every 25th second) by "`such_parameter: new_value`". See the language manual for the set of usable **commands** in the program areas.
One can set small iterated steps (`dynamics'), describe a function of time (`cinematics'), or of the mouse, or of the position of the other mobiles (e.g. chasing), or any combination programmed according to once fantasy.
*(NB: predefined sets of behaviors are also available to pick up ingredients if wanted. These **presets** are stored in a text file, so an teacher can easily modify them.)*



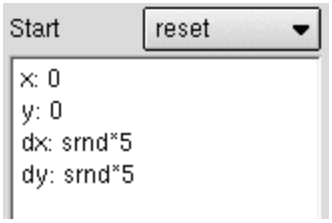
- What to do when a mobile touch a border.



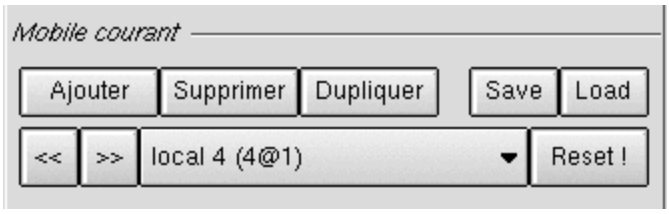
- What to do when a mobile touch another mobile.
If necessary, one can settle a different behavior, depending on the collider identity.
(e.g.: *if collider's icon is a racket then...*)



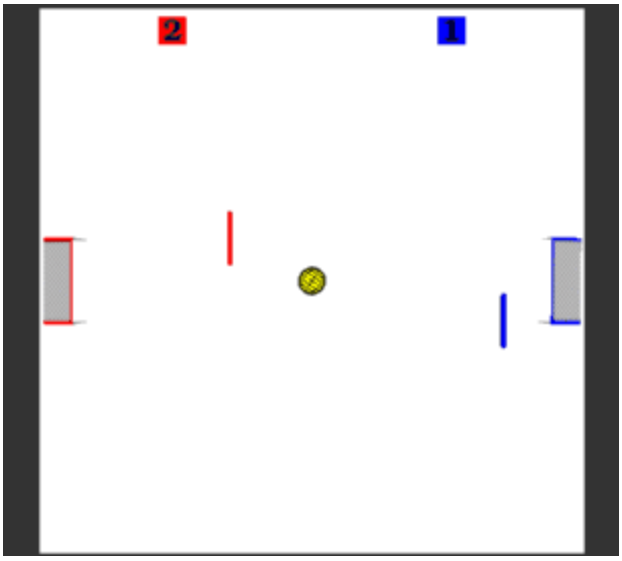
- Parameters value when (re)starting.
NB: a mobile can ask to restart itself (e.g. in case of collision) or to restart another mobile (e.g. to trigger a counter, a shot, to kick-off the ball...) using the command `restart num`. It is also a way to send **messages** between mobiles.
(`srnd` *gives a random number between -1 and 1.*)



- Creating a new mobile, choosing the current one, etc.

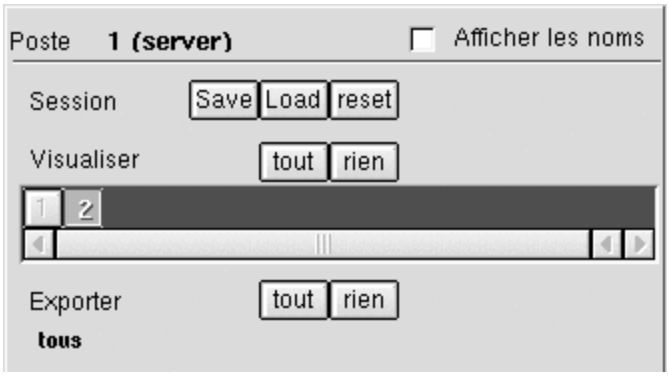


- Resulting display.



- Reading, writing, interacting with other computers through the network... One chooses the mobiles to be **exported** (i.e. that one make visible on the network) and the computers whose we **import** the mobiles.

Thus, several working modalities can be used:
- independent work on each station.
- displaying any station from a master station.
- displaying every station superimposed (e.g. pour wall projection).
- collaborative work by groups if 2 or 3 stations (or more).
- collaborative work between N stations (not necessarily on the local network).



- That's it ! Quite easy, isn't it ?

MobiNet - mobiles language (v1.1)

http://www-evasion.imag.fr/mobinet

State variables (or attributes) for a mobile:

x	horizontal position
y	vertical position
visible	is it visible
angle	its orientation (in radians)
icon	its icon
width	its width
height	its height
zoom	(to tune both values simultaneously)
color	its color
red	red component of its color
green	green component of its color
blue	blue component of its color
grey	grey component of its color
dx	(free; e.g.: vx)
dy	(free; e.g.: vy)
mem0..4	(free)

Program

A mobile program is a set of *instructions* allowing the modification of attributes. These instructions can be typed in various areas (see the *interface guide*), which will be executed depending the circumstances (every time, in case of collision with borders or another mobile, or only at start). The program is accounted for once "Apply" have been clicked. If it contains error the area display in red, and a message appears at the bottom of the window. (Note that the mobile will execute the program only if it has been switch on, by clicking on "Start"). Fore instance, here are 4 instructions which modify different attributes of the current mobile :

```
x :  50
color :  RED
dy :  cos(t)
y :  y + dy
```

The last instruction `y : y + dy` tells that the vertical position of the current mobile must be increased by the amount of the value of `dy` . In the 3rd instruction, `t` represent time.

Here is a summary of available commands :

Functions:

+	-	*	/	sqr	sqrt	log
sin	cos	tan	asn	acs	ang	exp
norm	dist	rnd	srnd	min	max	
abs	int	frac	sgn	mod	and	or
=	<	>	<=	>=	<>	

Variables and constants:

t	dt	PI	click	clicked	unclick	contact
me	next	prev	mouse	collider	camera	chockx
BLACK	WHITE	RED	GREEN	BLUE	light (,2,3)	chocky
GREY	CYAN	ORANGE	YELLOW	PURPLE		EMPTY

nearest...	,2,3	_indir(,2,3)	_left	_right	_up	_down
key...		_space	_left	_right	_up	_down

Commands:

:	stop	start	restart	like	move_to	if	then	else	endif	trace:

Example:

```
y :  80*cos(t)
mem1 :  80*sin(t + PI/3)
if y > 0 then
   color:  GREEN
   x :  mem1
else
   color:  RED
   x :  0
endif
```

Note the instruction `if :` between `if` and `then` settles a *test*. If the test is right, then the following instructions up to `else` are executed, otherwise these are the ones between `else` and `endif`. The `else` can be omitted if there is nothing to do.
Everything can also be written on a single line (note the '`;`'):

```
if x>0 then color:  RED; else color:  BLUE; endif
```

The main functions:

`abs(`f`)`	computes the absolute value of f
`int(`f`)`	computes the integer part of f
`frac(`f`)`	computes the fractionary part of f
`sgn(`f`)`	computes the sign of f
a `mod` b	computes a modulo b
`sqrt(`f`)`	computes the square root of $\|f\|$
`sqr(`f`)`	computes f^2
`sin(`f`)`	computes the sinus of f (in radians)
`cos(`f`)`	computes the cosine of f (in radians)
`tan(`f`)`	computes the tangent of f (in radians)
`ang(`a,b`)`	computes the angle (orientation) of vector (a,b)
`dist(`m,n`)`	computes the distance between mobiles m and n
`norm(`a,b`)`	computes $\sqrt{a^2+b^2}$, norm of vector (a,b)
`rnd`	get a random number between 0 and 1
`srnd`	get a random number between -1 and 1
`min(`a,b`)`	computes the smallest between a and b
`max(`a,b`)`	computes the largest between a and b

The other mobiles

A mobile's program can modify the attributes of his mobile, but not the ones of another mobile. Conversely, it can read them. Let supposes that the current mobile is mobile 1. We want mobile 1 to have the same horizontal position than mobile 2. We can write this program for mobile 1 :

```
x :  x2
```

- The numbers of mobiles that are before and after the current mobile in the list are obtained by `prev` and `next`.
- The number of the current mobile is obtained by `me`.
- The number of the mobile closest (on screen) to the current mobile is obtained by `nearest` (one can also use the shortcut `pp`).
- In case of collision, the number of the collided mobile is obtained by `collider`.

The various kinds of mobiles:

`m`	`prev`	`me`	`mouse`	`EMPTY`	`nearest`
`m@n`	`next`	`collider`	`camera`	`light`	`(nearest variations)`

The mouse

It is also a mobile. One cannot modify its attributes. But one can easily program a mobile to follow the mouse :

```
y : xmouse          or          m: mouse
y : ymouse                      x : xm
                                y : ym
```

One can also known its direction and speed using `dxmouse` et `dymouse`.
Moreover, one can know if the mouse button have been clicked by `if click then`, if it has been released by `if unclick then`, if it is still pressed by `if clicked then`.

The camera

It is also a mobile ! One can change its position (e.g.: `xcamera: xmouse`), its zoom (`zoom camera: 2`), its angle (to rotate the view)...
Moreover, clear screen can be disabled by `visible camera: 1` to let objects make a visible trace along their trajectory, which allows tracing curves (shortcut: `trace: 1`).
Remark: the camera is reseted when "reset" is clicked.

Lights

Similarly, one can change the light color (e.g. : `color light: RED` or `grey light: (1+sin(t))/2`) and its direction (e.g.: `x light: x1; y light: y1; height light: 10`). Initially the light height is infinity. Two extra lights `light2` and `light3` are available, initially off (i.e. black color).

Reading the attributes of a mobile managed on a distant computer

For instance we want that our mobile take the color of mobile 3 on station 15.
this writes: `color : color3@15`

m`@`n is the mobile number m on computer (or "station") number n (quite like an email address). NB: for this mobile m`@`n to be reachable, its computer n must have *exported* it (by clicking on "Export"), and that you *import* its mobiles by clicking on its station number n in the strip "Visualize". It is then also accounted for by the commands like `nearest`.

Local variables

One can use any word to store a value (intermediate variable):

```
object : mouse          // object is now a synonym for mouse
d : dist(me,object)     // d contains the distance to the mouse
x : x + 3*(x-xobject)/d // move in the mouse direction with length 3
y : y + 3*(y-yobject)/d
```

Remarks on attributes

Icons:
Note that icons 0 to 9 figure digits 0 to 9, which eases the construction of counters.

Width, height:
Attention, it is a *zoom factor* relative to the initial size of the icon. Most icons have an initial size of about 10.

Angle:
Here also, it is the rotation angle relative to initial orientation, which can be either horizontal or vertical depending on icons. NB: all angles in MobiNet are in radians.

Colors:
`color` only allows to provide a color name. To tune or obtain precisely the RGB color, derived attributes `red` , `green` , `blue` (and also `grey`) are available.

Collisions

They are managed in the "Collision" area. Various extra variable are available: `collider` gives the number of the collided mobile. So one can test this variable if reaction should depend on the mobile. E.g.: `if collider=2`, or `if collider=2@3`, or `if color collider = BLUE` . Note that it is often simpler to test the icon or the color of the mobile - corresponding to its category - rather than is number.

For advance use, also available are: the variable `contact` which tells whether we are treating this collision for the first time, and the *collision vector* `chockx,chocky` which encodes the angle of the mobiles contact. See for instance how the **preset** "choc" uses it.

Remark: there is always a risk that collision occurs again at the next step. Testing `contact` allows to avoid treating twice the collision (e.g. bouncing twice would produce a wrong direction). Another technique, also usable for collisions with borders, consist in *uncolliding* before treating, doing for instance `x: x-dx ; y : y-dy`. Remind that collisions are a difficult task in computer sciences, and that it is difficult to produce a perfect behavior (especially with fast motions).

Interaction between mobiles

Acting on another mobile:

One cannot modify the attributes of another mobile (it can only be displaced using the command `move_to(m,x,y)`). But one can act on its state using the commands `stop m` to stop it, `start m` to start it, or `restart m` to restart it even if it was already on.

When this mobile m restart, it first executes its "Start" program (if you entered a program in this area). this is useful to kick-off a ball which would have gone out of the terrain, for instance. But this also allows to *send signals* to mobiles: for instance, a counter can be incremented remotely using `restart`, by putting `icon: icon+1` in its "Start" area.

Imitating the behavior of another mobile:

When several mobiles share the same behavior, on can tell for each necessary area to refer to the corresponding area of the reference mobile, using the command `like m` (where `m` is a mobile running on the same station).

Chasing, running away:

`nearest` allows to 'see' which is the nearest mobile, in order to get closer or further for instance. Numerous variants of this function allow to get a smarter behavior: `nearest2` and `3` allow to 'see' what comes in second or third. `nearest_left` allows to look only in the west quadrant, etc. `nearest_indir(vx,vy,ang)` looks only in direction (vx, vy) with a field of view aperture of ang. (Shortcuts: `pp, pp2, pp3, ppg, ppd, pph, ppb, ppv, ppv2, ppv3`).
ATTENTION: in this case, there might have no mobile in sight. So care should be taken to test whether the command really found a mobile:

```
m:  nearest_indir(dx,dy,Pi/2)
if m <> EMPTY then
    (run away or attack)
endif
```

Launching MobiNet

If the network is not used (personal use, or non-collaborative use in lab without master station), simply launch MobiNet.

In the other cases, first launch MobiNet on the master station, **then** launch `mobile -client master_station_name` on the others. (For a distant computer, the name is the full Internet address).
NB: in the case of a tutorial with students, the teacher should probably add (first) the option `-nosave` to forbid saving files.

MOBINET - Fiche de TP

`http://www-evasion.imag.fr/mobinet`

Introduction

N'hésitez pas à essayer, à recommencer autrement, à tester les exemples :
ça ne gronde pas, ça n'explose pas !
Par contre, si ça fait autre chose que ce que vous attendiez, c'est bien d'essayer de comprendre pourquoi.

1 Les variables états

Créez un mobile ; choisissez-lui un icône. Modifiez la valeur de x, y, angle, etc...

2 Mouvements simples

Les instructions de la zone *mouvement* sont exécutées à chaque instant, entre deux affichages d'écran (tous les 25^{eme} de seconde).
Exemples :
`x : x+1` , ou `x : x+10` , ou `x : x-1` , ou `angle : angle + 0.1` ,
ou `x : x+dx` (mettre une valeur dans la variable d'état dx !).

Exercice :
Faites un mobile qui va du point (100,0) au point (-100,100).
Remarques :
- plutôt que re-régler chaque fois à la main la position initiale, faites le une fois pour toute dans la zone *start*.

3 Évènements : les bords

Les instructions des zones *bords...* sont exécutées quand on touche les bords, si l'on veut que quelque chose de particulier se passe (rebondir, coller, cycler, s'arrêter...).
Exemple :
dans *bord droit*, mettre `dx : -dx` (rebond), ou `dx : 0` (colle),
ou `x : 0` (cycle) , ou `start` (redémarre le mobile).

Exercice :
Créez un mouvement quelconque (pas juste horizontal). Faites qu'il se passe quelque chose d'intéressant aux bords (pas forcément pareil sur les quatre bords).

4 Autres mouvements

Exemple :
`dy : dy-1` (gravité)
`dx : dx+10*srnd; dy : dy+10*srnd` (papillon)
`x : 10*cos(t); y : 10*sin(t)` (cercle autour de (0,0))

5 La souris

la position de la souris est `xsouris, ysouris`.
Exemple :
`x : xsouris-10; y : ysouris-10`

Exercices :
- **Faites que le mobile soit décalé de quelques cm à droite de la souris.**
- **Faites bouger le mobile 'en miroir' (i.e. symétrique) de la souris.**
- **Faites tourner le mobile en rond autour de la souris.**

Exemple :
faites *load mobile* et prenez 'aiguille.mobile' :
On peut interpréter la direction de la souris comme une direction à suivre. On peut se servir de ça comme volant, pour diriger le mouvement.

Exercices : (si on a le temps)
- **Faites un mobile piloté à la souris, c'est à dire qui avance dans la direction de l'aiguille.**
- **Faites que le mobile (avion, soucoupe) s'oriente dans la direction où il va.** (indice : regardez comment est fait le mobile de l'aiguille !).

6 Plusieurs mobiles

On a actuellement à l'écran deux mobiles à la fois : l'aiguille et votre véhicule. En fait on peut en mettre autant qu'on veut, et même les faire interagir :
Faites `ajouter`, et entrez le mouvement `x : x1-1`

Exercices :
Faites *reset* pour tout effacer.
- **Faites un mobile simple (qui suit la souris, ou qui avance en rebondissant sur les bords). Faites un second mobile qui tourne autour du premier.**
- **Faites des planètes qui tournent autour du soleil. Ajouter la lune qui tourne autour de la terre.**
- (si on a le temps). **Faites un mobile simple. Faites qu'un second mobile soit attiré par le premier (indice : pensez aux forces en physique, comme s'il y avait un ressort invisible !).**

Remarques :
- Regardez le *preset* 'ressort2' : il simule des ressorts qui ont une longueur à vide.
- On peut utiliser `prec` (signifiant 'mobile précédant') au lieu de 1 : `x : xprec+1`

Exercice :
Reprenez le mobile attiré par le mobile simple (ou refaites le avec 'ressort2'). Faites un serpent en dupliquant dix fois le dernier mobile.

7 Évènements : collisions

Comme pour les bords, les instructions de la zone *collision* sont exécutées quand le mobile en touche un autre. Le numéro du mobile touché est dans `c` (ses coordonnées sont donc `xc, yc`).
C'est ce qu'on utilise pour faire qu'un mobile raquette tape dans un mobile balle. (Astuce : faire `stop c` pour arrêter la balle quand elle sort du terrain. Cliquez sur *start* pour redémarrer la balle. Ou alors entrez `restart c` à la place de `stop` pour faire redémarrer directement la balle !).

Exercices :

- **Faites un entraîneur de tennis : la balle part (par exemple du point (100,0)) dans une direction aléatoire. Le mobile raquette, dirigé par la souris, la renvoie. Quand elle sort du terrain on en relance une autre.**
- **Cage de buts : c'est aussi une collision ! Mettez un mobile (immobile !) de buts à droite de l'écran. La balle doit s'arrêter quand elle tombe dedans.**
- **ajoutez un compteur pour le score.**
  Indices : si on fait `icone : 3` le dessin de l'icône est le chiffre 3.
  Si dans la zone *start* on met `icone : icone+1`, ce sera exécuté chaque fois qu'on démarre ou redémarre ce mobile, par exemple avec l'instruction `restart`.

8 Réseau

Tout ce qu'on a vu marche aussi en faisant intervenir les mobiles qui sont sur des machines différentes. Mettez vous par groupe de deux machines, faites *exporter tout* pour laisser voir vos mobiles, faites *importer* le numéro de poste de l'autre pour voir ses mobiles.
On voit le mobile 3 du poste 5 sous le nom 3@5 (un peu comme une adresse mail).

Exercices :

- **Pong : un poste fait une raquette et une balle. L'autre fait juste une raquette. La balle rebondi sur les bords haut et bas, et sort à gauche et à droite.**
- **Score : un compteur par camp. Ajouter 1 au compteur gauche quand on sort à droite et réciproquement. Chaque poste s'occupe de son compteur.**
- **Billard à 4 ou 6 trous. Un trou, c'est un mobile rond, noir, immobile, qui fait disparaître les balles qui le collisionnent... Ajoutez le score.**

Exercice final :

Foot : Faites `reset` et chargez le terrain de foot 'foot.session'. Créez un mobile de joueur différent des autres (exportez le pour qu'on le voit). Une moitié de la salle se met en rouge, l'autre en bleu. Faites *importer tout* pour voir les autres. Go !