MobiNet - mobiles language (v1.1)

http://mobinet.imag.fr/

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	luminosity of its color
dx	(free; e.g.: x motion)
dy	(free; e.g.: y motion)
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
```

Remarks: 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 brief summary of available commands (detailled later):

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

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, like this (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 = -1, 0 or 1
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 (excepted for some special mobiles). 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 (,2,3)`
`m@n`	`next`	`collider`	`camera`	`light (,2,3)`	`(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 :

```
x : 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 current mobile take the color of mobile 3 on station 15.
this writes: `color : color3@15`

m`@`n is the mobile number m on computer ("poste", i.e. "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 3 components of the RGB color, derived attributes `red` , `green` , `blue` (and also `grey`) are available.

Collisions between objects

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 time step. Testing `contact` allows to avoid treating twice the same 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 `m`. But one can displace it using the command `move_to(m,x,y)`, and 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 a reference mobile `m`, using the command `like m` (where `m` is a mobile 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 (or click on `mobinet_client` icon). (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.