MobiNet

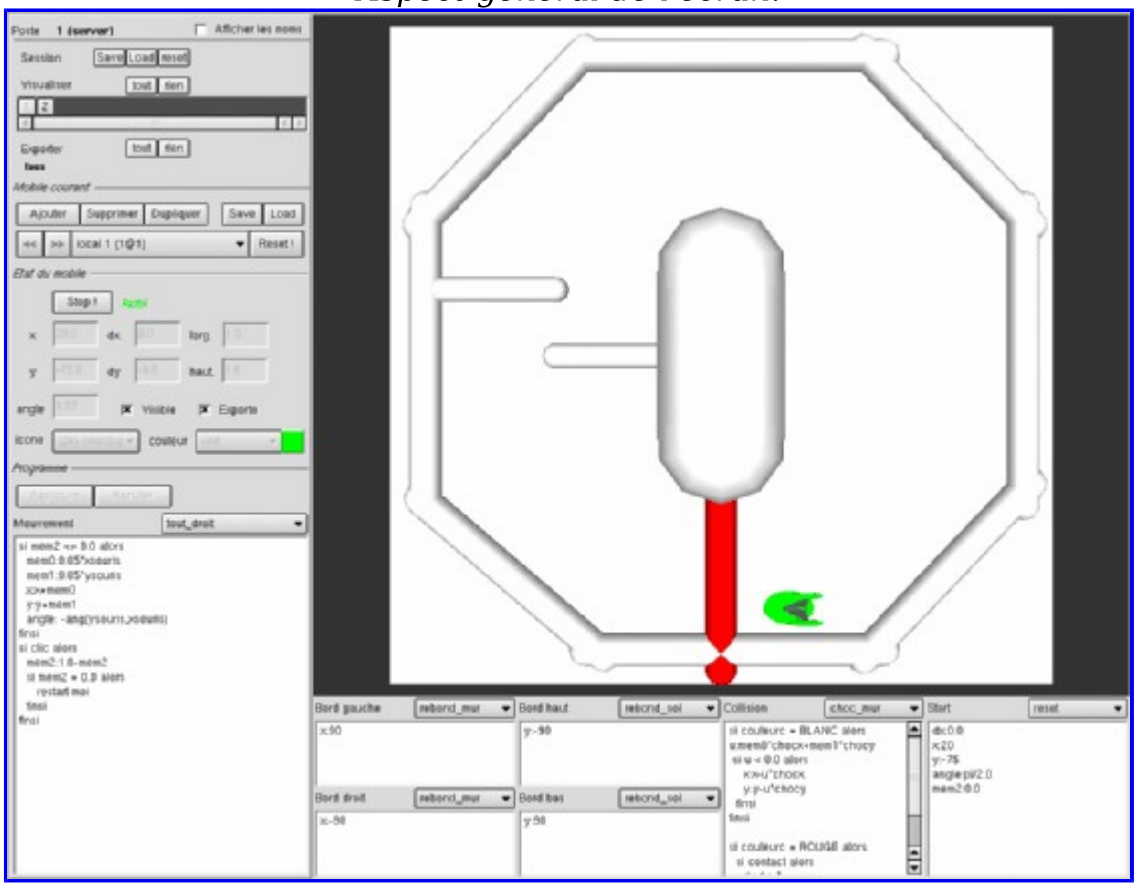
Plateforme de programmation de mobiles en réseau.

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

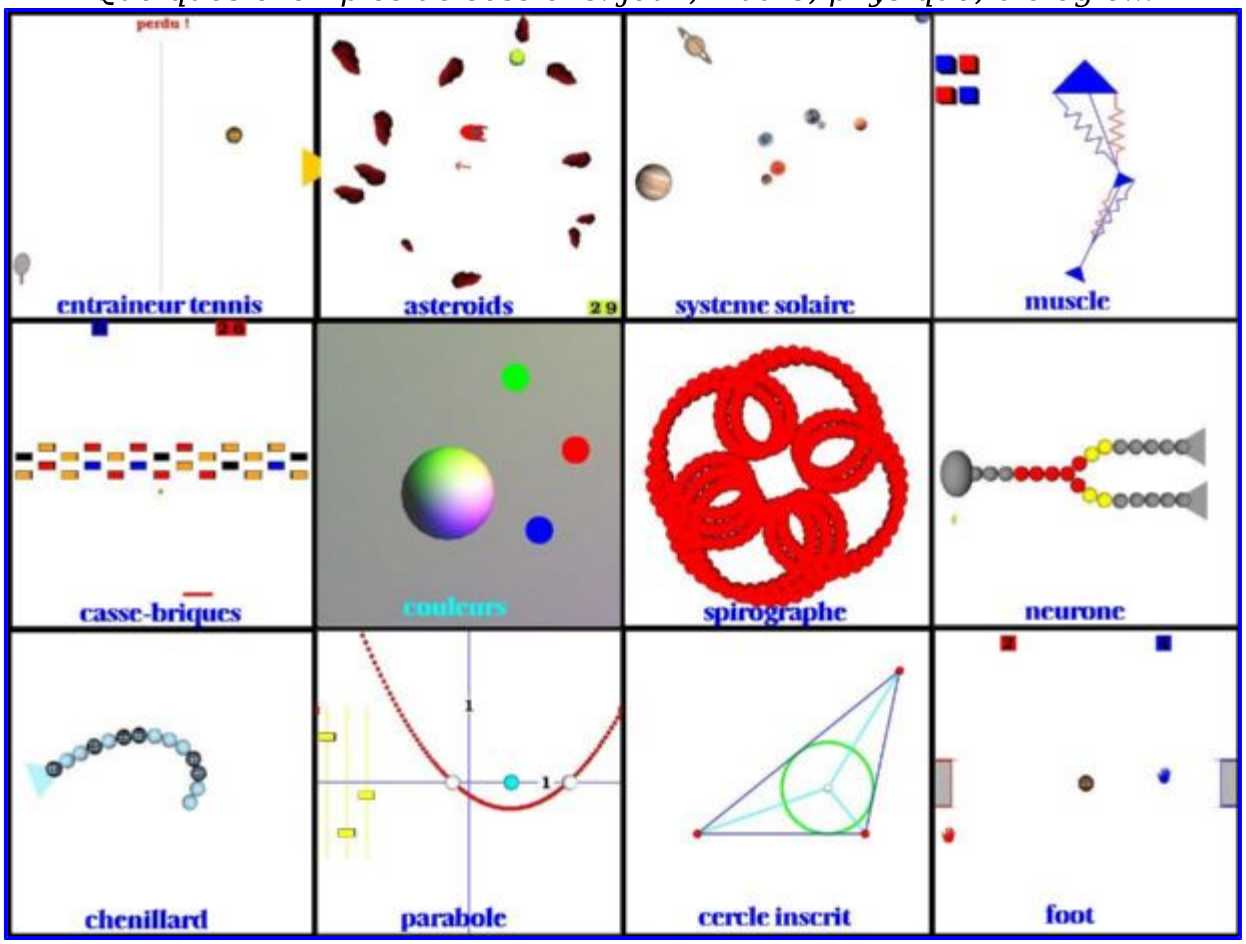
MobiNet est un logiciel permettant de programmer le comportement de mobiles animés à l'aide d'une interface et d'un langage intuitifs. MobiNet est totalement interactif, facilitant apprentissage et mise au point des mobiles. En outre, MobiNet est conçu pour fonctionner en réseau (mais il est également utilisable sur un poste isolé).
MobiNet est particulièrement intéressant en tant que plateforme pédagogique, dans le but d'initier les élèves (à partir du lycée) à la programmation des jeux, ou plus généralement de leur donner une version concrète, intuitive et ludique des notions vues en cours de math et de physique.

- **Chapitre 1:** L'interface de MobiNet.
- **Chapitre 2:** Le langage de MobiNet (PDF , PS).
- **Chapitre 3:** Tutoriel : feuilles d'exercices (PDF , PS).

Aspect général de l'écran.



Quelques exemples de sessions: jeux, maths, physique, biologie...



- **Contact:**

  Vous souhaitez vous inscrire sur notre mailing-liste ? Contactez-nous !

  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 - l'interface (v1.1)

Plateforme de programmation de mobiles en réseau.

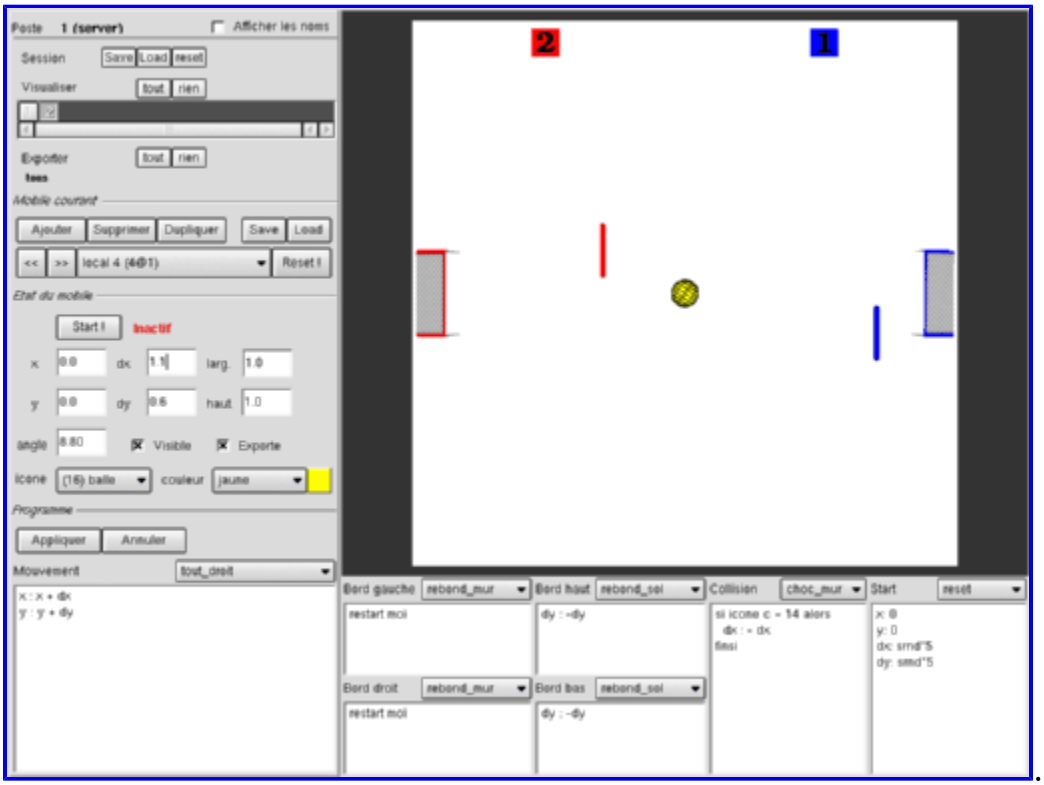
Retour à la page de départ de Mobinet.

*Dans MobiNet, on programme le comportement de différents **mobiles**: ce sont les différents objets visibles (qu'ils bougent ou non!). L'interface permet de régler tous les comportements du mobile courant: aspect, mouvement, règles de collision avec les bords ou les autres mobiles... (on commute ensuite d'un mobile à l'autre).*

Aspect général de l'écran (cliquer pour agrandir). Les différentes zones sont détaillées un peu plus loin.

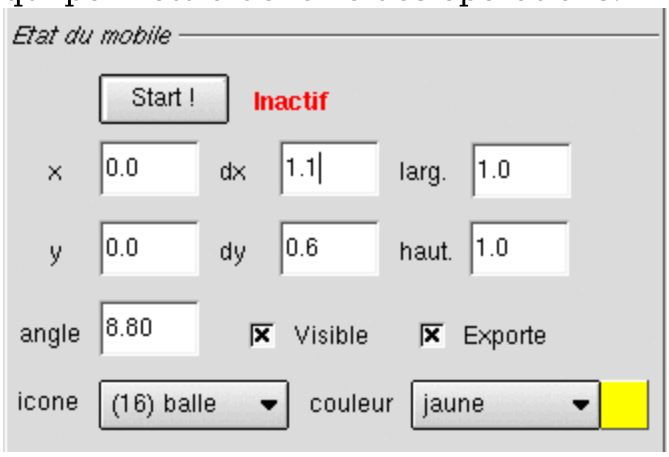
Ici on a programmé 7 mobiles: la balle, les raquettes, les buts, les compteurs.

- La balle suit une trajectoire rectiligne, et rebondit sur les bords et les raquettes.
*- Les raquettes suivent la souris verticalement. L'une des raquettes (ainsi que le but et le compteur correspondant) est en fait gérée sur une **autre machine**: deux utilisateurs partagent ici leurs mobiles en réseaux. L'un gère les rouges et la balle, l'autre les bleus.*
- Les buts enregistrent les collisions avec la balle, et déclenchent les compteurs adverses.
- Les compteurs s'incrémentent quand les buts le leur signalent.



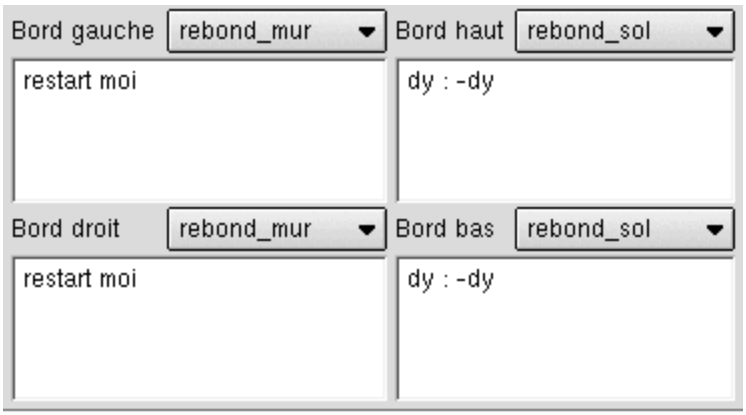
- En cliquant sur `afficher les noms', on voit les numéros des mobiles (`local n` *s'il est sur la machine,* `n@m` *s'il est géré par la machine m*), ainsi que le système de coordonnées (*de -100 à 100 pour les x et les y*).

- Paramètres représentant le mobile courant (*ici, la balle*): position, dimension, icône, couleur, orientation... (*de plus on va utiliser dx,dy pour indiquer la direction du mouvement.*)
NB: en réalité tout est numérique (*e.g. l'icône de la balle est le numéro 16*), ce qui permettra de faire des opérations.



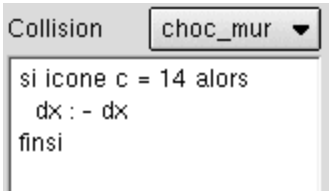
- Programmation du mouvement: consiste à décrire les changements à faire entre 2 images successives (env. tous les 25eme de seconde), par "`tel_paramètre: nouvelle_valeur`". Voir le manuel du langage pour le choix des **commandes** utilisables dans les zones de programmation.
On peut faire des petits mouvement successifs (`dynamique'), décrire une fonction du temps (`cinématique'), ou de la souris, voire dépendant de la position des autres mobiles (e.g. poursuite), ou toute combinaison programmée selon l'inspiration.
(NB: on dispose d'un choix de comportements prédéfinis où piocher si on le souhaite. Ces ***presets*** *sont stockés dans un fichier texte, un animateur peut donc facilement les modifier.)*

- Quoi faire quand on touche un bord.



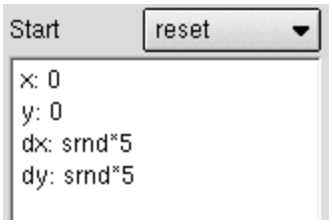
- Quoi faire quand on collisionne un autre mobile.

Si nécessaire, on peut choisir un comportement différent selon l'identité du collisionneur.
(traduction: *si l'icône du collisionneur est une raquette, alors...*)

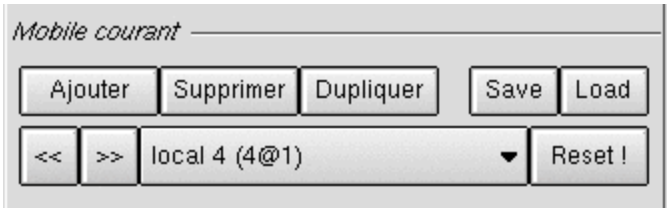


- Valeur des paramètres au (re)démarrage.

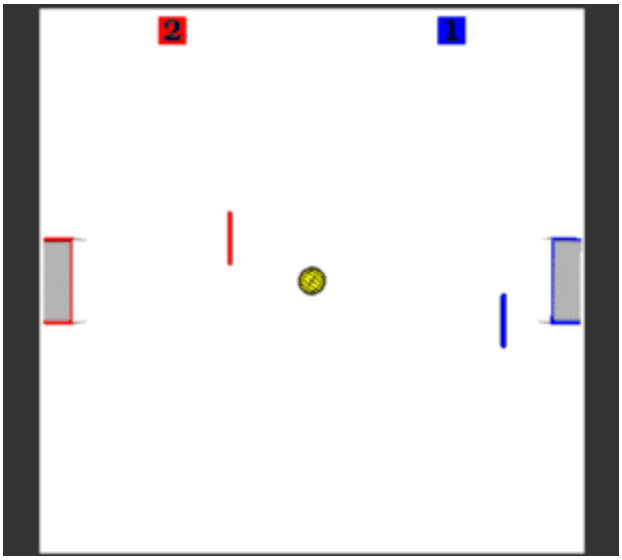
NB: on peut dire de se redémarrer (e.g. en cas de collision) ou de redémarrer un autre mobile (e.g. pour déclencher un compteur, un tir, remettre la balle en jeu...) avec l'instruction `restart num`. C'est une façon d'envoyer des **messages** entres mobiles.
(`srnd` *donne un nombre au hasard entre -1 et 1.*)



- Créer un nouveau mobile, choisir sur lequel on travaille, etc.

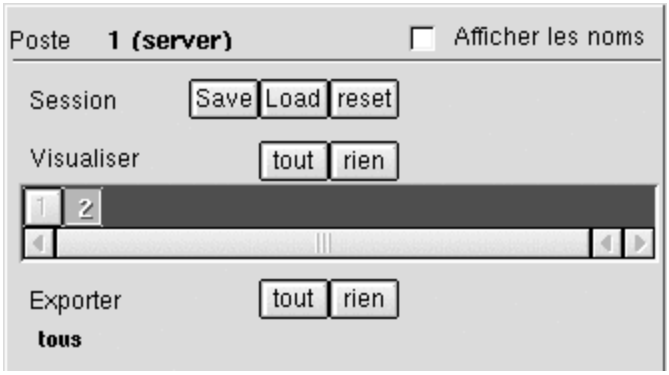


- Affichage du résultat.



- Lire, écrire, interagir avec d'autres machines en réseau...
On choisit d'une part les mobiles que l'on **exporte** (i.e. que l'on rend visible sur le réseau), et d'autre part les machines dont on importe les mobiles.

On dispose donc de plusieurs modes de travail:
- travail indépendant sur chaque poste.
- visionnage de n'importe quel poste depuis le poste maître.
- visionnage superposé de tous les postes (e.g. pour projection murale).
- travail en binômes ou en trinômes (ou autre).
- travail collectif sur N postes (pas forcément sur le réseau local).



- Et voilà ! plutôt facile, non ?

MobiNet - langage des mobiles (v1.1)

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

Variables d'état (ou attributs) d'un mobile :

x	l'abscisse de sa position
y	l'ordonnée de sa position
visible	indique s'il est visible
angle	son orientation (en radians)
icone	son icône
largeur	sa largeur
hauteur	sa hauteur
zoom	(pour régler les deux à la fois)
couleur	sa couleur
rouge	la composante rouge de la couleur
vert	la composante verte de la couleur
bleu	la composante bleu de la couleur
gris	la composante grise de la couleur
dx	(libre ; par ex : vx)
dy	(libre ; par ex : vy)
mem0..4	(libre)

Programme

Un programme de mobile est une suite d'*instructions* permettant de modifier les attributs. Ces instructions peuvent être entrées dans diverses zones (voir le *guide de l'interface*), qui seront exécutées selon les circonstances (tout le temps, en cas de collision aux bords ou avec un autre mobile, ou juste au démarrage). Le programme est pris en compte une fois que l'on a cliqué sur "Appliquer". S'il comporte des erreurs, la zone s'affiche en rouge, et un message apparaît en bas d'écran. (Notez que le mobile n'exécutera le programme que si on l'a mis en marche, en cliquant sur "Start"). Voici par exemple 4 instructions qui modifient différents attributs du mobile courant :

```
x : 50
couleur : ROUGE
dy : cos(t)
y : y + dy
```

Remarquons que la dernière instruction, `y : y + dy`, indique que l'abscisse du mobile courant doit être augmentée de la valeur `dy` . Dans la 3ème instruction, `t` représente le temps.

Voici un récapitulatif des instructions que l'on peut utiliser :

Fonctions :

+	-	*	/	carre	racine	log
sin	cos	tan	asn	acs	ang	exp
norme	dist	rnd	srnd	min	max	
abs	ent	frac	sgn	mod	et	ou
=	<	>	<=	>=	<>	

Variables et constantes :

t	dt	PI	clic	cliquee	declic	contact
moi	suiv	prec	souris	lui	camera	chocx
NOIR	BLANC	ROUGE	VERT	BLEU	lampe (,2,3)	chocy
GRIS	CYAN	ORANGE	JAUNE	VIOLET		VIDE

plusproche...	,2,3	_vers(,2,3)	_gauche	_droite	_haut	_bas
touche...		_espace	_gauche	_droite	_haut	_bas

Commandes :

:	stop	start	restart	comme	met_en	si	alors	sinon	finsi	trace :

Exemple :

```
y : 80*cos(t)
mem1 : 80*sin(t + PI/3)
si y > 0 alors
   couleur : VERT
   x : mem1
sinon
   couleur : ROUGE
   x : 0
finsi
```

Remarquer l'instruction `si` : entre `si` et `alors` se trouve un *test*. Si le test est vérifié, alors les instructions qui suivent jusqu'au `sinon` sont exécutées, sinon ce sont celles entre `sinon` et `finsi` qui le seront. On peut omettre le `sinon` s'il n'y a rien à y mettre.
On peut également tout écrire sur une seule ligne (noter les '`;`') :

```
si x>0 alors couleur : ROUGE; sinon couleur : BLEU; finsi
```

Les principales fonctions :

`abs(`f`)`	calcule la valeur absolue de f
`ent(`f`)`	calcule la partie entière de f
`frac(`f`)`	calcule la partie fractionnaire de f
`sgn(`f`)`	calcule le signe de f
a `mod` b	calcule a modulo b
`racine(`f`)`	calcule la racine carrée de $\|f\|$
`carre(`f`)`	calcule f^2
`sin(`f`)`	calcule le sinus de f (en radians)
`cos(`f`)`	calcule le cosinus de f (en radians)
`tan(`f`)`	calcule la tangente de f (en radians)
`ang(`a,b`)`	calcule l'angle (orientation) du vecteur (a,b)
`dist(`m,n`)`	calcule la distance entre les mobiles m et n
`norme(`a,b`)`	calcule $\sqrt{a^2+b^2}$, norme du vecteur (a,b)
`rnd`	tire un nombre aléatoire entre 0 et 1
`srnd`	tire un nombre aléatoire entre -1 et 1
`min(`a,b`)`	calcule le plus petit nombre entre a et b
`max(`a,b`)`	calcule le plus grand nombre entre a et b

Les autres mobiles

Un programme de mobile peut modifier les attributs de son mobile, mais ne peut pas modifier les attributs d'un autre mobile. En revanche, il peut les lire. Supposons que le mobile courant soit le numéro 1. Nous voulons que le mobile 1 ait la même abscisse que le mobile 2. Nous pouvons écrire ce programme pour le mobile 1 :

```
x : x2
```

- Les numéros des mobiles situés avant et après le mobile courant dans la liste sont donnés par `prec` et `suiv`.
- Le numéro du mobile courant est donné par `moi`.
- Le numéro du mobile le plus proche (à l'écran) du mobile courant est donné par `plusproche` (on peut aussi utiliser l'abréviation `pp`).
- En cas de collision, le numéro du mobile collisionné est donné par `lui`.

Les différents types de mobiles :

m	prec	moi	souris	VIDE	plusproche
m@n	suiv	lui	camera	lampe	(variantes de pp)

La souris

C'est aussi un mobile. On ne peut cependant pas modifier ses attributs. Mais on peut facilement programmer un mobile pour qu'il suive la souris :

```
y : xsouris                ou encore        s : souris
y : ysouris                                 x : xs
                                            y : ys
```

On peut également connaître sa direction et sa vitesse avec `dxsouris` et `dysouris`.
D'autre part, on peut savoir si le bouton de la souris a été cliqué par `si clic alors`, s'il a été relâché par `si declic alors`, s'il est maintenu appuyé par `si cliquee alors`.

La caméra

C'est aussi un mobile ! On peut changer sa position (ex : `xcamera : xsouris`), son zoom (`zoom camera : 2`), son angle (pour tourner la vue)...
En outre, on peut désactiver l'effacement d'écran par `visible camera : 1` pour que les objets laissent une trace visible de leur trajectoire, ce qui permet de dessiner des courbes (raccourcit : `trace : 1`).
Remarque : la caméra est remise à zéro quand on clique sur "reset".

Les lampes

De même, on peut changer la couleur de l'éclairage (ex : `couleur lampe : ROUGE` ou `gris lampe : (1+sin(t))/2`) et sa direction (ex : `x lampe : x1; y lampe : y1; hauteur lampe : 10`). Initialement la hauteur de la lampe est à l'infini. On dispose en fait de deux lampes supplémentaires `lampe2` et `lampe3`, initialement éteintes (i.e. couleur noire).

Lire les attributs d'un mobile sur une machine distante

Nous voulons par exemple que notre mobile prenne la couleur du mobile 3 de la machine 15.
Ceci s'écrit : `couleur : couleur3@15`

m`@`n désigne le mobile numéro m de la machine (ou "poste") numéro n (un peu comme pour une adresse mail). NB : pour que ce mobile m`@`n soit accessible, il faut que la machine n l'ait *exportée* (en cliquant sur "Exporter"), et que vous *importiez* ses mobiles en cliquant sur son numéro n dans le bandeau "Visualiser". Il est alors également pris en compte par les commandes comme `plusproche`.

Variables locales

On peut utiliser des mots quelconques pour stocker une valeur (variable intermédiaire) :

```
objet : souris          // objet devient un raccourci pour souris
d : dist(moi,objet)     // d contient la distance à la souris
x : x + 3*(x-xobjet)/d // avance de 3 dans la direction de la souris
y : y + 3*(y-yobjet)/d
```

Remarques sur les attributs

Icônes :
Noter que les icônes 0 à 9 représentent les chiffres de 0 à 9, ce qui facilite la construction de compteurs.

Largeur, hauteur :
Attention, il s'agit d'un *facteur de zoom* par rapport à la taille initiale de l'icône. La plupart des icônes ont une taille initiale d'environ 10.

Angle :
Ici aussi, il s'agit de l'angle de rotation par rapport à l'orientation initiale, qui peut être horizontale ou verticale selon les icônes. NB : tous les angles dans MobiNet sont en radians.

Les couleurs :
`couleur` permet juste d'indiquer un nom de couleur. Pour régler ou connaître précisément la teinte en RGB, on dispose des attributs dérivés `rouge` , `vert` , `bleu` (ainsi que `gris`).

Les collisions

On les traite dans la zone "Collision". On dispose alors de divers variable supplémentaires : `lui` indique le mobile avec qui on est entré en collision. On peut ainsi tester cette variable s'il faut agir différemment selon le mobile. Exemple : `si lui=2`, ou `si lui=2@3`, ou `si couleur lui = BLEU` . À noter qu'il est souvent plus commode de tester l'icône ou la couleur du mobile, qui caractérisent généralement sa catégorie, plutôt que son numéro.

Pour une utilisation avancée, on dispose en outre de la variable `contact` qui indique si c'est la première fois que l'on traite cette collision, et du *vecteur de collision* `chocx,chocy` qui indique sous quel angle les mobiles se sont touchés. Voir par exemple comment le **preset** "choc" en fait usage.

Remarque : il y a toujours un risque que l'on soit à nouveau en état de collision au pas de temps suivant. Tester `contact` permet d'éviter de traiter 2 fois la collision (on risquerait alors de re-rebondir dans la mauvaise direction). Une autre technique, également utilisable pour les collisions au bords, consiste à *décollisionner* avant toute chose, en faisait par exemple `x : x-dx; y : y-dy`. Sachez cependant que les collisions sont une tâche délicate en informatique, et qu'il est difficile d'obtenir un comportement parfait (notamment si on bouge vite).

Interaction entre mobiles

Agir sur un autre mobile :

On ne peut modifier les attributs d'un autre mobile (on peut juste le déplacer avec la commande `met_en(m,x,y)`). Par contre on peut agir sur son état avec commandes `stop m` pour l'arrêter, `start m` pour le démarrer, ou `restart m` pour le remettre en route même s'il était déjà en marche.

Quand ce mobile m redémarre, il commence par exécuter son programme "Start" (si vous en avez entré un dans cette zone). C'est utile par exemple pour remettre en jeu une balle qui est sortie, mais cela permet également d'*envoyer des signaux* à des mobiles : un compteur pourra par exemple être incrémenté à distance par `restart`, en mettant `icone : icone+1` dans sa zone "Start".

Imiter le comportement d'un autre mobile :

Quand plusieurs mobiles partagent le même comportement, on peut dire, pour chaque champs souhaité, de se 'reporter' au programme du champs correspondant du module de référence, à l'aide de la commande `comme m` (`m` étant un mobile tournant sur le même poste).

Poursuite, fuite :

`plusproche` permet de 'voir' quel est le mobile le plus proche, ce qui permet par exemple de s'en approcher ou de s'en éloigner. De nombreuses variantes de cette fonction permettent d'affiner ce comportement : `plusproche2` et `3` permettent de 'voir' qui arrive en second ou en troisième, `plusproche_gauche` permet de ne regarder que dans le cadran Ouest du champs de vision, etc. `plusproche_vers(vx,vy,ang)` ne regarde que dans la direction (vx, vy) avec un champ de vision d'ouverture ang. (Abréviations : `pp, pp2, pp3, ppg, ppd, pph, ppb, ppv, ppv2, ppv3`). ATTENTION : dans ces conditions, il peut n'y avoir aucun mobile en vue. Il faut donc prendre garde à tester si la commande a bien trouvé un mobile :

```
m : plusproche_vers(dx,dy,Pi/2)
si m <> VIDE alors
    (fuir ou attaquer)
finsi
```

Démarrage de l'application

Si l'on n'utilise pas le réseau (usage individuel, ou usage non collaboratif et sans poste maître en salle de TP), lancer simplement l'application.

Dans les autres cas, lancer simplement l'application sur la machine maître, puis lancer **ensuite** `mobile -client nom_machine_maitre` sur les autres. (Pour une machine distante, le nom est l'adresse internet complète). NB : dans le cas de TP avec des élèves, on aura sans doute intérêt à ajouter (en premier) l'option `-nosave` pour interdire la sauvegarde.

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 !