# TME numéro 2

## (première communication avec OcamlPilotd)

**Exercice 1** Écrivez une fonction qui déplace votre vaisseau spatial de 3 pixels vers la droite et vers le haut toutes les demi secondes en utilisant le `gettimeofday` (comme nous l'avons vu en cours) et regardez ce qu'affiche votre fonction `affiche` du TME précédent. Utilisez le `synchronize` pour éviter les scintillements.

**Exercice 2** Sur la page ouèbe, vous trouverez des fichiers `aide_client.mli`, `aide_client.cmi` et `aide_client.cmo`. Ce dernier contient des fonctions que vous pouvez utiliser pour converser avec le serveur `ocamlpilotd`. Le fichier `.cmi` contient les prototypes (compilés) des fonctions fournies par le fichier `.cmo`, ces prototypes sont en bytecode et ne sont donc pas lisibles. En revanche, vous pouvez les regardez dans le fichier `aide_client.mli`. En ce qui concerne ce TME, voici les prototypes qui vont nous être utiles :

| *Fichier aide_client* | |
|---|---|
| nom de fonction | type |
| `tme2_get_balles_a_afficher` | `unit -> Jeu.eltJeu list` |
| `tme2_get_vaisseaux_a_afficher` | `unit -> Jeu.eltJeu list` |
| `tme2_get_plateau_a_afficher` | `unit -> Jeu.eltJeu list` |
| `tme2_get_obstacles_a_afficher` | `unit -> Jeu.eltJeu list` |
| `tme2_recupere_action_joueur` | `unit -> unit` |
| `tme2_connexion_serveur` | `string -> string -> Graphics.color` |

Voici comment vous pouvez vous servir de tous ces fichiers : afin qu'ocaml puisse faire ses vérifications de type, il faut charger les types de toutes les fonctions en mémoire. Ceci peut être réalisé en utilisant la commande suivante dans votre toplevel :

```
open Aide_client;;
```

Ensuite, il faut charger le code des fonctions du tableau ci-dessus. Là, vous n'avez rien à faire si vous compilez en utilisant le `mytop` du répertoire `/Infos/lmd/2004/licence/ue/li260-2005fev/g7/bin`. En effet, lorsque ce dernier a été généré, l'équipe enseignante lui a rajouté toutes les fonctions `tme2_*`. En revanche, si vous n'utilisez pas le `mytop`, il vous faudra écrire en dessous du « `open Aide_client;;` » la ligne suivante :

```
#load "aide_client.cmo";;
```

et les fonctions fournies par le fichier `aide_client.cmo` seront dans l'environnement d'ocaml et pourront donc être utilisées dans votre programme.
Maintenant, examinons comment on peut se servir de ces fonctions. La première fonction à lancer est `tme2_connexion_serveur` : cette fonction réalise la connexion avec un serveur que vous aurez préalablement lancé (je rappelle que des exécutables du client/serveur se trouvent sur la page ouèbe du module ainsi que dans le répertoire `/Infos/lmd/2004/licence/ue/li260-2005fev/g7/bin`). Pour que la connexion s'établisse, vous devez passer à la fonction `aide_connexion_serveur` l'adresse de la machine sur laquelle s'exécute le serveur `ocamlpilotd` et le numéro de port sur lequel se trouve ce dernier. Par défaut, `ocamlpilotd` se lance sur le port 12345. En ce qui concerne l'adresse de la machine, vous pouvez soit spécifier le nom de la machine (i.e., `shalmaneser.lip6.fr`) si celle-ci est enregistrée dans le DNS, soit par son numéro IP (i.e., 132.227.204.42). Si le serveur se trouve sur la

même machine que votre toplevel, vous pouvez passer l'adresse 127.0.0.1 (c'est l'adresse du loopback et elle fonctionne pour toutes les machines UFR et hors UFR). Ainsi, en salle de TME, vous devriez probablement lancer la commande suivante :

```
let couleur_vaisseau = tme2_connexion_serveur "127.0.0.1" "12345";;
```

La fonction vous demandera la couleur de votre vaisseau spatial. Vous donnerez alors la couleur qui vous intéresse et **vous validerez cette couleur en appuyant sur la touche « entrée » du pavé numérique**. En effet, dans le « `caml-toplevel` » de xemacs, chaque fois que vous tapez sur le retour chariot, cela indique à xemacs de compiler l'expression en question sous ocaml. Or, lorsque vous tapez la couleur de votre vaisseau, vous ne voulez rien compiler, vous voulez juste fournir une chaîne de caractère lue au clavier à une fonction. Dans le « `caml-toplevel` » de xemacs, c'est la touche « entrée » du pavé numérique qui remplit cet office. Une fois la touche appuyée, vous êtes en relation avec le serveur `ocamlpilotd` et la fonction vous renvoie votre couleur sous forme d'un `color` du module `Graphics`.
Puisque vous êtes en relation avec le serveur, vous pouvez maintenant lui envoyer des ordres : essayez de récupérer la liste des vaisseaux spatiaux du jeu en utilisant la fonction :

```
tme2_get_vaisseaux_a_afficher ();;
```

Familiarisez-vous avec les autres fonctions du tableau ci-dessus.

**Exercice 3** Créez une fonction `affiche_jeu : Graphics.color -> unit` qui réalise l'affichage des objets du plateau de jeu (la couleur passée en argument est celle de votre vaisseau spatial). Pour cela, cette fonction commencera par récupérer auprès du serveur la liste des vaisseaux, balles, etc, à afficher grâce aux fonctions :
– `tme2_get_balles_a_afficher ()`
– `tme2_get_vaisseaux_a_afficher ()`
– `tme2_get_plateau_a_afficher ()`
– `tme2_get_obstacles_a_afficher ()`
puis elle appellera la fonction d'affichage que vous aviez programmée au TME précédent. Testez votre fonction (je vous suggère de lancer un ocamlpilot dans une console et de lui faire tirer quelques balles près de votre vaisseau spatial, simplement pour voir si votre fonction affiche correctement ce vaisseau ainsi que les balles). Pour finir, n'oubliez pas les commentaires OcamlDoc, il me semble que vous aviez oublié de les écrire (rappelez-vous la touche `M-o`).

**Exercice 4** Écrivez une boucle qui réalise 30 fois par seconde :

1. la récupération de ce que l'utilisateur a tapé au clavier et l'envoi de l'instruction correspondante au serveur : la fonction `tme2_recupere_action_joueur ()` fait justement tout cela.
2. l'affichage de tous les éléments du plateau de jeu.

Testez votre réalisation intensivement. Tout fonctionne correctement ? Alors vous avez obtenu la partie cliente de votre application.
Pour achever cet exercice et ce TME, vous allez modifier la partie de votre code qui réalise l'attente du 30ème de seconde : vous avez programmé une boucle pour faire cela lors du premier exercice. En soit, c'est correct mais vous effectuez tout le temps des `gettimeofday ()`. Or cela a pour conséquence que, bien que votre but soit de dormir pendant 1/30ème de seconde, vous monopolisez tout le temps votre processeur. Vous allez donc remplacer votre boucle d'attente par un simple appel à la fonction `nanosleep : float -> unit` qui a pour but de réellement dormir le nombre de secondes indiqué par le réel passé en paramètre.
Maintenant, je peux vous donner la ligne qui a permis de générer le `mytop` que vous avez utilisé :

```
ocamlmktop -custom -thread -o mytop unix.cma graphics.cma threads.cma \
ocamlpilot_stub.o aide_client.cmo
```

le fichier `ocamlpilot_stub.o` contient le programme C réalisant le `nanosleep`.