Protocole utilisé dans OcamlPilot

Ce poly décrit le protocole utilisé par les clients/serveurs du projet numéro 7 : développement d'un jeu de type Xpilot sous Ocaml.
Afin de simplifier vos programmes (à la fois client et serveur), le protocole utilisé est uniquement en *mode texte*. Ainsi, si le serveur doit envoyer le nombre 42 au client, il enverra précisément la chaîne de caractères « 42 ». Le mode texte, bien que très inefficace en terme de rapidité, a été retenu pour les deux raisons suivantes :

1. Il vous permet de débugger facilement vos programmes puisque vous pouvez demander à Ocaml d'afficher les chaînes de caractères transmises ou reçues du serveur (ou du client).
2. Il évite les problèmes de représentation des entiers en mémoire. En effet, la représentation d'un même nombre peut différer suivant les architectures (big-endian, little-endian) et, si l'on n'y prête attention, il se peut que les transmissions d'entiers soient erronées.

Dans le reste de ce poly, les conventions suivantes sont appliquées :
- Chaque ligne de texte envoyée du serveur vers le client (resp. du client vers le serveur) sera préfixée de la chaîne « **Serveur :** » (resp. « **Client :** »). Par exemple, le fait que le serveur envoie la chaîne « ERREUR SYNTAXE » sera écrit :
  Serveur : `ERREUR SYNTAXE`
- Les caractères des chaînes transmises par le réseau seront notés dans une police machine à écrire, comme `ERREUR SYNTAXE`, tandis que les nombres seront écrits en italique. Par exemple, la ligne :
  Serveur : `PUT TAILLE PLATEAU` *largeur hauteur*
  signifie que le serveur a envoyé la chaîne de caractères « PUT TAILLE PLATEAU » suivie de la chaîne correspondant au nombre largeur et de celle correspondant au nombre hauteur.
- Enfin, chaque ligne envoyée ou reçue par le serveur se termine par un caractère « `\r` ». Ce dernier correspond à un retour chariot (carriage return). Dans votre code ocaml, vous devrez donc écrire « `\r` » à la fin de chaque chaîne. Ces retours chariot vous permettront de tester votre client/serveur avec telnet.

1 Protocole à l'initialisation de la connexion

Lorsqu'un client se connecte au serveur, il doit commencer par indiquer la couleur du joueur qui veut se connecter de la manière suivante :

Client : `PUT COULEUR VAISSEAU` *couleur*`\r`

couleur est un `color` du module Graphics d'Ocaml, c'est-à-dire un nombre représentant une couleur de la forme 0xRRGGBB, où RR, GG et BB sont des nombres hexadécimaux (cf. la doc du module Graphics). Vous pouvez utiliser les couleurs prédéfinies par le module Graphics : `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan` et `magenta`. À l'issue de cette commande, le serveur répondra par :

Serveur : `ERREUR COULEUR UTILISEE\r`

si cette couleur est déjà utilisée, ou bien par :

Serveur : `VAISSEAU ENREGISTRE\r`

si tout s'est bien passé, ou bien encore par :

Serveur : `ERREUR SYNTAXE\r`

s'il n'a pas reçu correctement la chaîne `PUT COULEUR VAISSEAU` *couleur*`\r`.
Lorsque le client reçoit `ERREUR COULEUR UTILISEE\r`, il doit renvoyer une ligne `PUT COULEUR VAISSEAU` avec une autre couleur. Tant que le client n'a pas reçu la chaîne `VAISSEAU ENREGISTRE\r`, il n'a pas

accès au jeu. Une fois le vaisseau enregistré, l'initialisation est terminée et le jeu peut commencer pour le client.

2 Protocole en cours de jeu

En cours de jeu, le client peut effectuer auprès du serveur deux types d'actions différentes :

1. il peut demander des informations au serveur. Toutes ces demandes sont des chaînes de caractères commençant par le préfixe `GET`. Le serveur renvoie alors la réponse à l'aide d'une chaîne commençant par le préfixe `PUT`.
2. il peut envoyer une action au serveur. Dans ce cas, l'action commence par la chaîne `PUT`. Le serveur répond avec la chaîne `OK\r`.

2.1 Les demandes d'information du client

Afin d'harmoniser l'aspect du plateau de jeu entre les différents clients, chacun d'entre eux doit demander au serveur les caractéristiques de ce plateau. La première des caractéristiques est la taille (en pixels) de l'espace de jeu (ce dernier est un rectangle de taille *largeur*×*hauteur*) :

Client : `GET TAILLE PLATEAU\r`

ce à quoi le serveur répond par :

Serveur : `PUT TAILLE PLATEAU` *largeur hauteur*`\r`

Les vaisseaux spatiaux que devra afficher le client sont des cercles. Le rayon de ceux-ci peut être demandé au serveur de la manière suivante :

Client : `GET RAYON VAISSEAU\r`

et le serveur répondra par :

Serveur : `PUT RAYON VAISSEAU` *taille*`\r`

Les balles lancées par ces derniers sont des cercles dont le rayon peut, là encore, être demandé au serveur avec la commande :

Client : `GET RAYON BALLE\r`

et le serveur répondra par :

Serveur : `PUT RAYON BALLE` *rayon*`\r`

Le nombre de points de vie des vaisseaux au début du jeu est déterminé par le serveur, aussi le client doit-il s'en informer s'il veut représenter les points de vie comme le fait le client qui vous est fourni en démonstration. Pour cela, le client doit envoyer la chaîne :

Client : `GET MAX PT VIE\r`

et le serveur répondra par :

Serveur : `PUT MAX PT VIE` *nombre_de_points*`\r`

Pour se rassurer, le client peut aussi demander au serveur si le vaisseau du joueur est mort ou bien vivant, de la manière suivante :

Client : `GET STATUT\r`

et le serveur répondra par :

Serveur : `PUT STATUT mort\r`

si le joueur est mort, ou bien par :

Serveur : `PUT STATUT vivant\r`

dans le cas contraire.

Pour complexifier un peu le jeu, des obstacles sont placés sur le plateau en début de jeu. Ceux-ci resteront aux mêmes endroits pendant toute la partie. Les obstacles sont des cubes dont le client peut demander la liste avec la commande :

Client : `GET CUBES\r`

Le serveur renverra alors la liste des coordonnées des coins inférieurs gauches et supérieurs droits des cubes. Pour cela, il commencera par envoyer une chaîne indiquant le nombre de cubes dans le jeu :

Serveur : `PUT CUBES` *nombre_de_cubes*`\r`

puis, pour chaque cube, il enverra une ligne :

Serveur : *x_gauche y_bas x_droit y_haut*`\r`

Ainsi, s'il existe deux cubes dont les coordonnées des coins extrêmes sont respectivement cube 1 : (12,10) (22,33) et cube 2 : (100,20) (150,800), le serveur enverra successivement les lignes suivantes :

Serveur : `PUT CUBES` *2*`\r`
Serveur : *12 10 22 33*`\r`
Serveur : *100 20 150 800*`\r`

Lorsque le client désire afficher l'état du jeu à un moment donné, il doit demander au serveur les positions des vaisseaux et des balles. Dans le premier cas, il demande au serveur :

Client : `GET VAISSEAUX\r`

et le serveur répond par une première ligne indiquant le nombre de vaisseaux :

Serveur : `PUT VAISSEAUX` *nombre_de_vaisseaux*`\r`

suivi, pour chaque vaisseau, par ses coordonnées, son angle de direction (en radians, c'est donc un nombre réel), son nombre de points de vie et sa couleur :

Serveur : *x y angle points_de_vie couleur*`\r`

De la même manière, si le client veut s'enquérir des positions des balles, il peut envoyer la ligne suivante au serveur :

Client : `GET BALLES\r`

et le serveur répondra en donnant le nombre de balles :

Serveur : `PUT BALLES` *nombre_de_balles*`\r`

suivi, s'il en existe, par la liste des coordonnées et de la couleur de chaque balle :

Serveur : *x y couleur*`\r`

2.2 Les actions transmises au serveur

Le client peut réaliser 6 actions différentes : effectuer une rotation vers la gauche, une rotation vers la droite, accélérer, décélérer, tirer, ou bien se retirer de la partie. Celles-ci peuvent être transmises au serveur de la manière suivante :

Client : `PUT ACTION GAUCHE\r`
Client : `PUT ACTION DROITE\r`
Client : `PUT ACTION ACCELERE\r`
Client : `PUT ACTION DECELERE\r`
Client : `PUT ACTION TIRE\r`
Client : `PUT ACTION QUITTE\r`

Le serveur répondra alors par :

Serveur : `OK\r`

Si le client envoie une chaîne au serveur qui n'est pas répertoriée dans le protocole décrit ci-dessus, le serveur n'effectue aucune opération mais renvoie un message :

Serveur : `ERREUR SYNTAXE\r`