TME numéro 7

(premiers pas vers le serveur)

Le TME d'aujourd'hui vous permettra de commencer à écrire votre serveur OcamlPilotd. Vous trouverez sur la page web d'OcamlPilot dans la section « ressources » un fichier `types_serveur.mli` qui vous fournira des types permettant de gérer les vaisseaux et balles présents dans le jeu, à savoir :

```
type vaisseau = {
  mutable vaiss_x       : float;          (* abscisse du vaisseau *)
  mutable vaiss_y       : float;          (* ordonnee du vaisseau *)
  mutable vaiss_angle   : float;          (* direction de deplacement en radians *)
  mutable vaiss_vitesse : float;          (* vitesse du vaisseau *)
  mutable vaiss_pt_vie  : int;            (* nombre de points de vie restant *)
  vaiss_couleur         : Graphics.color; (* couleur du vaisseau *)
};;
type balle = {
  mutable balle_x     : float;          (* abscisse de la balle *)
  mutable balle_y     : float;          (* ordonnee de la balle *)
  mutable balle_angle : float;          (* direction de deplacement en radians *)
  balle_couleur       : Graphics.color; (* couleur de la balle *)
};;
type cube = {
  left   : float; (* abscisse du coin inférieur gauche *)
  bottom : float; (* ordonnée du coin inférieur gauche *)
  right  : float; (* abscisse du coin supérieur droit  *)
  up     : float  (* ordonnée du coin supérieur droit  *)
};;
```

Utilisez ce fichier, ces types vous seront particulièrement utiles pour les TME suivants.

Exercice 1 Écrivez une fonction `cree_cubes :int -> Types_serveur.cube list` qui, étant donné un nombre x, renvoie une liste de x obstacles placés « presque » aléatoirement sur le plateau de jeu. Par « presque », on entend qu'une grille dont le maillage a la taille d'un obstacle est plaquée sur le plateau de jeu et que les cubes sont placés aléatoirement à l'intérieur de ce maillage (comme le montre la figure 1). Ceci permet d'assurer que les vaisseaux spatiaux peuvent toujours se faufiler entre 2 obstacles non contigus.

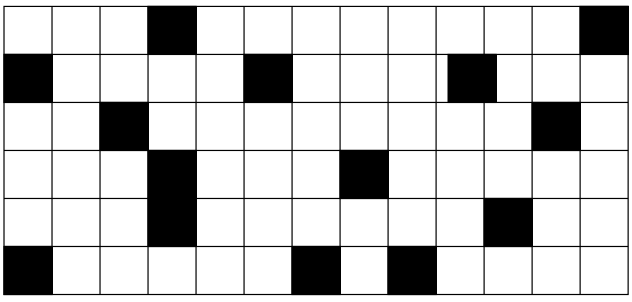

FIG. 1 – Le maillage sur lequel sont posés les obstacles.

Exercice 2 (utilisation de `Aide_serveur`) Comme pour l'écriture du client, vous avez à votre disposition un fichier d'aide nommé `aide_serveur`. Ce fichier contient toute l'implémentation du serveur que vous avez utilisé lors de l'écriture de votre client, à savoir :

```
aide_main : unit -> unit
registered_between : (float * float -> float * float -> float * float -> bool) ref
registered_client : (Unix.file_descr -> unit) ref
registered_cree_cubes : (int -> Types_serveur.cube list) ref
registered_cree_vaisseau : (Graphics.color -> bool) ref
registered_detruit_vaisseau : (Graphics.color -> unit) ref
registered_get_cube_inter_segments : (float -> float -> ((float * float) * (float * float)) list) ref
registered_init_client : (in_channel -> out_channel -> Graphics.color ref -> string) ref
registered_inter_cubes : (float -> float -> ((float * float) * (float * float)) list -> bool) ref
registered_inter_plateau : (float -> float -> bool) ref
registered_inter_vaisseaux : (float -> float -> Graphics.color -> bool) ref
registered_intersect_segment : (float * float -> float * float -> float * float -> float * float -> bool) ref
registered_left : (float * float -> float * float -> float * float -> int) ref
registered_mise_a_jour : (unit -> unit) ref
registered_mise_a_jour_balles : (unit -> unit) ref
registered_mise_a_jour_pos_balles : (unit -> unit) ref
registered_mise_a_jour_vaisseaux : (unit -> unit) ref
registered_traite_action_joueur : (Graphics.color -> Jeu.action -> unit) ref
```

Pour l'utiliser, il suffit de faire un `open Aide_serveur` puis de faire un appel à `aide_main()`. Comme vous le voyez, toutes les fonctions « `registered_xxx` » sont des références sur des fonctions. Vous pouvez donc remplacer leurs valeurs par celles de vos propres fonctions. Ainsi, pour tester votre `cree_cubes`, il suffit d'évaluer l'expression :

```
registered_cree_cubes := cree_cubes;;
```

puis d'appeler `aide_main()`. Cela lance alors le serveur OcamlPilotd. Essayez et vérifiez le résultat avec votre propre client.

Exercice 3 Programmez une fonction `inter_plateau : float -> float -> bool` qui, étant donné les coordonnées du centre d'un vaisseau spatial, renvoie un booléen indiquant si le vaisseau se trouve à l'intérieur de l'arène de jeu. Enregistrez votre fonction dans le serveur en utilisant l'expression :

```
registered_inter_plateau := inter_plateau;;
```

Exercice 4 Écrivez une fonction `inter_vaisseaux : float -> float -> Graphics.color -> bool` qui, étant donné les coordonnées du centre d'un vaisseau spatial ainsi que sa couleur, renvoie un booléen indiquant si ce vaisseau a une intersection non vide avec un autre vaisseau. Vous supposerez qu'il existe une variable globale `vaisseaux : Types_serveur.vaisseau list ref` contenant la liste de tous les vaisseaux spatiaux du jeu. Cette variable a, bien entendu, été déclarée dans le fichier `aide_serveur`. Enregistrez votre fonction `inter_vaisseaux` et testez votre serveur.

Exercice 5 Écrivez une fonction `traite_action_joueur : Graphics.color -> Jeu.action -> unit` qui, étant donné la couleur d'un vaisseau spatial et une action (GAUCHE, ACCELERE, etc), met à jour l'angle du vaisseau concerné et sa vitesse. Si l'action consiste à tirer une balle, il faut rajouter celle-ci dans la liste des balles du jeu. Cette dernière est stockée dans une variable globale déclarée dans le fichier `aide_serveur` et répondant au doux nom de `balles : Types_serveur.balle liste ref`.