Chapitre 1

Serveur OCaml - Mathieu Chailloux

Le serveur a été réalisé en OCaml (version 3.11.2, pas besoin de la 4.00). Il utilise les modules Thread, Unix, et Str (pour les expressions régulières). Les extensions réalisées sont la discussion instantanée, les comptes utilisateurs, et les spectateurs. Tout est centralisé dans le fichier my_serveur.ml, bien qu'il eût été possible de séparer les différents modules en fichiers distincts. Nous détaillerons par la suite les choix d'implémentation qui ont été faits.

1.1 Etablissement du serveur

L'établissement du serveur s'est fait en s'appuyant sur le chapitre 20 du livre Développement d'Applications en Objective Caml (Client/Serveur). Ainsi, pour permettre l'utilisation simultanée des modules Thread et Unix, on utilise le module ThreadUnix pour la communication (les fonctions de lecture/écriture sur les sockets sont donc redéfinies). Enfin, la socket associée au serveur est réutilisable, même en cas d'utilisation récente (ce qui est assez utile en phase de tests). Il est possible de fournir l'adresse sur laquelle on veut lancer le serveur par l'option "-adress", sinon il se lance sur le localhost.

1.2 Traitement d'une commande

Afin de traiter les commandes reçues, on les transforme en une liste dont chaque élément est délimité par "/" (non précédé par un "\" afin de respecter le protocle), il a donc fallu redéfinir une fonction quasi-équivalente à Str.split mais qui respectait le protocole. Ainsi, commande et arguments sont aisément identifiés. Le traitement des commandes se reposera par la suite sur la liste ainsi obtenue.

1.3 Connexion

La phase de connexion (module Connexion) est séparée de la boucle principale de jeu afin d'identifier les conditions de connexion (joueur ou spectateur) et lancer le traitement

adéquat. Si le pseudo reçu est déjà utilisé (joueur présent, ou utilisateur enregistré), alors un nouveau pseudo est généré en rajoutant un entier à la fin. Au bout de 2 joueurs connectés, un timer est lancé, implémenté par un thread qui, une fois les 30 secondes écoulées, lance la partie si c'est nécessaire. Sinon la partie est lancée au bout de 4 joueurs connectés.

1.4 Structure de données

Lors de la connexion, la structure représentant le client est alors initialisée. Elle comprend :

- son nom
- la socket grâce à laquelle il communique avec le serveur
- la liste de ses bateaux
- la phase de jeu courante
- la position de son drone

Les joueurs sont stockés dans une liste (de taille maximale 4 donc), ce qui permet d'utiliser les nombreuses fonctions du module List.

Les spectateurs sont eux stockés dans une autre liste, et ne sont représentés que par leur socket.

La phase de jeu est représentée par un type somme qui nous permettra ainsi de vérifier si une commande est licite à un moment de la partie donné.

Un bateau est une suite de cases et possède un état. Une case est une position munie d'un état.

Le type état est représenté par un type somme (Alive ou Dead). Il peut paraître ici assez inutile, mais il a été implémenté initialement dans l'objectif de réaliser d'extensions qui n'ont finalement jamais vu le jour (réparer un bateau, lancer une épidémie, ...).

Enfin, la grille de jeu est simplement représentée par une matrice de noms (à partir desquels on pourra retrouver la structure du client).

1.5 Extensions

1.5.1 Discussion instantanée

Cette extension a été plutôt aisée à implémenter, dès que le serveur reçoit la commande "TALK" correctement formée, il renvoie à tout le monde (clients et spectateurs) le message passé en argument précédé de son émetteur. Le split de la commande permet de respecter le protocole.

1.5.2 Comptes utilisateurs

Cette extension repose sur un simple fichier texte où sont stockés logins et mots de passe. Le fichier ("logins.txt") est créé s'il n'existe pas.

1.5.3 Spectateurs

Pour cette extension, il a fallu ajouter un mécanisme qui stocke les commandes voulues et les envoie aux spectateurs déjà présents. Dès qu'un spectateur se connecte, il reçoit alors la liste des commandes stockées. Grâce au split des commandes, il a suffi de changer le nom de la commande ("PUTSHIP" en "PLAYERSHIP", ...) qui correspond au premier élément de la liste résultante.

1.6 Architecture

L'architecture n'est pas forcément très rigoureuse car elle n'a pas été développé dans un esprit de réutilisabilité, mais en voici une brève description. Lors de la connexion d'une nouvelle socket, un thread exécutant "main_client" est lancé. Cette fonction lance la phase de connexion évoquée plus haut, et selon le résultat exécute "main_joueur" ou "main_spectator". La deuxième fonction se contente de réceptionner les commandes déjà effectuées et permet au spectateur de parler. La première effectue une boucle qui filtre à chaque tour la phase du joueur et la commande reçue afin de lancer le traitement adéquat.

Ci-suit une liste des modules :

- RegExp : stocke les expressions régulières, mais elles ont été beaucoup moins utilisées que pensé initialement
- Utils : toutes les fonctions utilitaires, ne se rapportant pas spécifiquement à un autre module
- Next : gère le correct enchaînement des "YOURTURN"
- Register : gère les comptes utilisateurs
- Stop : gère les déconnexions
- Connexion : cf plus haut
- Placement : gère la phase de placement des bateaux (et oui, quel nom adéquat !)
- End_of_game : gère la fin de partie (plus ou moins bien, des tests exhaustifs n'ont pas été effectués)
- Game : gère les phases d'actions

1.7 Remarques

En plus de l'option "-address" qui prend en argument l'adresse du serveur, est présente la possibilité d'afficher la trace par l'option "-debug". La plus grande difficulté a été, à mon sens, de tester le code, telnet ayant ses limites (notamment pour finir une partie...) et les versions finales s'étant développées sous la pression des délais. De plus, les commandes envoyées via telnet n'étaient pas forcément exactement les mêmes que celles reçues lors des phases de tests avec les clients des mes collègues. Un bug m'a été signalé par mes collègues lors de la fin du timer en phase de connexion, mais je n'ai jamais pu le constater chez moi ou à l'ARI, même lors des phases de tests finales. Si

cela se reproduit, une solution est de commenter le lancement du timer dans la fonction “treat_connexion” (affectation de la référence timer) et de décommenter la ligne “start_game ()”. En espérant que cela ne soit pas nécessaire :)

Bon jeu !