Communiquer et jouer en réseau

Aurèle Barrière & Rémi Hutin

5 avril 2016

Table des matières

1 Retransmisison des matchs en directs

2.1

Nous avons utilisé le code issu de nos deux projets. Les fonctions de mise à jour du plateau et de calcul de score viennent d'un des projets, tandis que l'affichage côté client vient de l'autre.

Le premier à l'avantage de ne considérer les couleurs que comme de caractères, qui s'envoient donc facilement par les *sockets* TCP.

Le second permet d'avoir, grâce à la SDL2 (`www.libsdl.org/index.php`), un affichage élaboré pour le client (qui permet aussi de recevoir les choix du joueur en cliquant simplement sur la couleur souhaitée).

2.2

On modifie ainsi le serveur. On commence par attendre une connection en acceptant sur un port (par exemple 7777).

Ensuite, on garde l'identifiant de la *scoket* crée après acceptation et on lance une partie.

Au début, on envoie le plateau entier sur cette *socket*, puis on envoie le coup joué et le numéro du joueur à chaque tour.

Quand la partie est finie, on envoie à la place du prochain coup un signal de fin de partie.

2.3

Ainsi, il suffit de créer un client qui suive le schéma suivant :
— On se connecte sur le port demandé (par exemple 7777, qui est non réservé).
— Tant qu'on a pas reçu un signal de fin de partie (par exemple, le premier caractère du buffer est le caractère spécial '*'), on reçoit sur la *socket* utilisée pour la connection soit le plateau entier, soit le dernier coup joué et le numéro du joueur.
— Grâce aux fonctions de mise à jour et d'affichage, on retransmet la partie en cours.

2.4 (Bonus)

2.5 (Bonus)

2.6

2 Poste mono-client

2.7

On pourrait modifier le serveur ainsi :
— Acceptation des connections sur un certain port.
— Création du plateau et initialisation du jeu.
— Tant que la partie continue,
— On envoie au client et aux observateurs l'état de la partie.
— Si c'est au client de jouer, on attend son choix sur sa *socket*.
— Sinon, on joue comme d'habitude.
— On met à jour l'état du jeu.
— Quand la partie est finie, on envoie un signal de fin.

2.8

2.9 (Bonus)

2.10

Dans un premier temps, on se sert des valeurs de retour des fonctions `send()` et `recv()` pour vérifier qu'on a bien reçu un *buffer* de la taille escomptée. Quand on a bien reçu, on renvoie un signal ('ACK') pour garantir à l'envoyeur qu'il peut continuer.

Si ce n'est pas le cas lors de l'envoi ou de la réception avec un client, on consièrera au bout d'un certain nombre d'essais (défini dans une constante), que le joueur est déconnecté.

Le serveur attribuera alors automatiquement la victoire au second joueur.

2.11

3 Compétition équitable

2.12

2.13 (Bonus)

4 Bonus

Deux joueurs à distance

Nous avons souhaité séparer complètement les rôles des clients et du serveur.

Ainsi, le serveur se contentera d'attendre la connection de 2 joueurs, de recevoir leur choix, de vérifier si ces choix correspondent aux règles du jeu (sinon, on attribuera une couleur aléatoire), de mettre à jour, et de diffuser (aux clients comme aux observateurs).

Deux clients se chargeront donc de jouer à tour de rôle.

Il n'y a que peu de modifications à faire dans le protocole : au lieu de jouer le coup du serveur, on écoutera sur une autres *socket* le coup du second joueur.

5 Protocole