Communiquer et jouer en réseau

Aurèle Barrière & Rémi Hutin

5 avril 2016

Table des matières

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`), 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 connexion en acceptant sur un port (par exemple 7777).

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

Nous avons choisi d'envoyer la totalité du plateau à chaque tour à tous les observateurs. Cela présente les avantages suivants :

- Simplification des calculs côté client. Le client se contente d'effectuer un affichage.
- Le calcul est effectué à un seul endroit ("single point of truth").
- Le code est plus clair côté observateur et côté joueur client. La connexion d'un observateur en cours de partie est également simplifiée.

De plus, le problème de la bande passante est négligeable car la taille des données envoyées est dans tous les cas très faible.

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 connexion le plateau entier 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

Ce programme a été testé sur un réseau local (sur la même machine et sur des machines différentes).

Un problème s'est manifesté : vu que nous avons pris des morceaux de nos 2 projets, nous n'avions pas les mêmes conventions. Par exemple, l'un considérait des couleurs de 0 à

6 et l'autre de 97 à 103 (le code ASCII de 'a' à 'g'). La première version est utile pour la génération aléaoire, mais la seonde plus pratique pour l'affichage.

Pour résoudre ce problème, il a fallu décaler les indices de couleurs en ajoutant ou retirant 'a' à certaines variables.

Poste mono-client

2.7

Une première solution serait d'envoyer le plateau au client joueur, que celui choisisse son coup, qu'il calcule le résultat puis l'envoie au serveur en envoyant le plateau entier mis à jour.

On pourrait également modifier le serveur ainsi :

— Acceptation des connexions 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.

La première solution, bien que facilement implémentable, soulève plusieurs problèmes :

— Pour les observateurs, nous avions laissé le calcul au serveur. Pourquoi ne pas faire de même pour les joueurs ?
— On ne peut pas être sûr du code exécuté par le client sur une machine distante. Un joueur pourrait très bien avoir légèrement modifié le code tout en respectant le protocole pour tricher sur la mise à jour du plateau. Le contrôle des règles doit se faire côté serveur.

2.8

Nous avons donc choisi la deuxième solution pour des raisons de simplicité et de sécurité.

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 considèrera 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

Compétition équitable

2.12

2.13 (Bonus)

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 connexion 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 autre *socket* le coup du second joueur.

Protocole