Devoir de programmation PC2R : iSketch

Clara Muller & Théo Lebourg

1 Côté serveur

Pour développer la partie serveur de iSketch, nous avons opté pour le langage OCaml. Nous allons dans un premier temps expliquer quels ont été nos choix pour organiser notre serveur avant de s'attarder un peu plus en détail sur le code.

1.1 Lancement du serveur

Lorsque l'on lance l'exécutable, la première étape du programme est la création du serveur (pour qu'il soit prêt à recevoir les connexions des futurs clients) ainsi que l'initialisation de certaines variables qui seront utilisées une fois le jeu lancé.

La création de notre objet serveur reprend les étapes classiques vues en cours : on crée une **une prise** (en utilisant le domaine Internet, le flot d'octets et le protocole de communication par défaut) et on récupère **une adresse dans le domaine internet** (à l'aide de l'adresse Internet de la machine locale et d'un numéro de port défini par l'utilisateur) afin de pouvoir associer les deux, puis on rend la prise capable d'accepter les connexions et enfin, on met en place une boucle infinie qui accepte les demandes de connexions et qui va créer un thread joueur à chaque nouvelle demande (nous verrons plus tard comment nous gérons le nombre limite de joueurs par partie).

Notons qu'avant de commencer à accepter les demandes de connexion, nous lisons le fichier dictionnaire et nous stockons dans une liste chacun des mots présent dans le fichier. Nous avons opté pour cette solution pour éviter de devoir ouvrir et fermer le dictionnaire à chaque tour de round pour piocher un mot.

En parallèle des demandes de connexions des joueurs, on lance un thread qui va gérer une partie de la manière suivante :

1.2 Connexion d'un joueur

Plusieurs possibilités sont offertes au joueur pour accéder au jeu :

- Avec la commande `CONNECT/user/` : le serveur vérifie alors que le nom *user* n'a pas déjà été choisi (en vérifiant la base de données des comptes utilisateurs ainsi que les joueurs déjà connectés à la partie en cours) et crée un nouveau nom si ce n'est pas le cas (en concaténant un nombre à la fin de *user*).
- Avec la commande `REGISTER/user/password/` : le serveur vérifie alors que le nom *user* n'a pas déjà été choisi et l'ajoute à la base de données si c'est le cas.
- Avec la commande `LOGIN/user/password/` : le serveur vérifie alors que les noms *user* et *password* sont corrects et s'ils le sont, le serveur vérifie également que le joueur n'est pas déjà en train de jouer.

Enfin, notons qu'une commande `ACCESSDENIED/` est envoyée ou bien si le nombre maximum de joueurs pour une partie est atteint ou bien si les noms et mots de passe ne correspondent pas ou bien si un joueur tente de se connecter alors qu'il est déjà connecté.

1.2.1 Quand un mot est trouvé par un joueur

Voici le déroulement général des étapes effectuées par le serveur lorsqu'un joueur a trouvé le mot que le dessinateur était en train de dessiner. Notons que puisque les joueurs évoluent chacun sur un thread, nous avons protégé cette série d'opérations par un **mutex**.

1. Le serveur se charge d'envoyer la commande `WORD_FOUND/joueur/` à tous les joueurs de la partie.
2. Le serveur attribue le nombre de points qui va bien au joueur
3.

1.3 Extensions

1.3.1 Comptes utilisateurs

Côté serveur, nous avons opté pour le stockage des noms et mots de passe des joueurs dans un simple fichier texte. Cependant, avant d'être stocké sur ce fichier, le mot de passe est salé (dynamiquement pour éviter les attaques par tables arc-en-ciel) puis hashé avec la fonction de hashage MD5 (cette fonction de hachage n'est certes plus fiable depuis longtemps mais nous n'avons pas trouvé dans la librairie standard d'OCaml d'autres fonctions plus performantes telle SHA-256). Ainsi, à chaque inscription, on ajoute une ligne contenant le nom du joueur, le mot de passé hashé et salé et le sel.

Par ailleurs, notons que pour l'instant, les mots de passe circulent en clair lorsqu'il est envoyé depuis le client.

2 Côté client