TP Rust UDP: Implémentation d'un jeu pong en ligne

Dans ce TP nous allons implémenter le jeu pong avec des sockets UDP en language Rust. Les objectifs sont divers: dans un premier temps, il s'aggira de comprendre les bases du language Rust, son intérêt en terme de gestion de la mémoire et les avantages que cela apporte en terme de cyber-sécurité. Dans un second temps, nous essaierons de comprendre les enjeux de la programmation réseau et des jeux-vidéo en ligne en général.

Pour aller plus loin nous pourrons nous poser les questions suivantes : comment réduire la latence ? Comment éviter que les joueurs ne trichent ?

Introduction au Rust (1h, ou plus ?)

Commencez par installer rust sur votre machine : https://rust-lang.org/tools/install/

Il existe de nombreuses ressources pour apprendre ce language, ici, il est conseillé de suivre le livre rust et de faire au minimu les parties 1, 3 et 4 : https://doc.rust-lang.org/stable/book/title-page.html

Les rustlings sont aussi une bonne manière d'apprendre le language : https://rustlings.rust-lang.org

Quoiqu'il en soit, la partie la plus importante pour votre apprentissage du language est l'`ownership`. En effet Rust gère la mémoire d'une manière singulière : il n'utilise ni garbage collector[1] (comme en python, java, go...), ni vous demande de gèrer la mémoire à la main (comme en C, C++...). La mémoire est en fait gérée dans la syntaxe du language, ce qui permet une performance optimale à l'exécution, et évite les problèmes de sécurité que poserait une mauvaise gestion de la mémoire en C par exemple.

Cependant, au début il pourrait sembler que le compilateur ne soit pas votre ami. Apprenez à le connaître, prennez le temps de comprendre les concepts, et vous verrez qu'il est en fait très utile !

Aide: tester les différents concepts du livre dans un projet à part :

```
# Créer un nouveau projet
cargo new trying-rust
cd trying-rust
# Lancer le programme
cargo run
```

Pong (le reste)

Contexte

Il est maintenant temps de coder Pong (https://en.wikipedia.org/wiki/Pong).

Dans l'archive vous trouverez un squelette du projet nommé `pong-rust`. Les différents fichiers de code sont agencés de la façon suivante :

```
src
├── client
│   └── bin
│       └── main.rs <- exécutable du client
├── common
│   └── lib.rs <- librairie commune au client/serveur pour gérer le jeu
└── server
    └── bin
        └── main.rs <- exécutable du seveur
```

[1] Le garbage collector est une solution pratique pour le développeur, mais inefficace en terme de performances, voir: https://fr.wikipedia.org/wiki/Ramasse-miettes_(informatique)

La logique du jeu est déjà implementée pour vous faciliter la vie (cependant libre à vous de supprimer le contenu du fichier lib.rs pour plus de fun 🙃).

Le but est donc de gérer toute la "logique réseau". Vous utiliserez donc des sockets UDP pour envoyer et recevoir des informations entre le serveur et les deux clients (les joueurs).

La documentation rust pour les sockets UDP: https://doc.rust-lang.org/std/net/struct.UdpSocket.html
L'envoie d'information se fera en Json, pour cela nous utiliserons la librairie serde qui permet de Serializer/Dezerializer nos structures rust. Des exemples vous sont donnés dans le code.

Pour chaque fichier main il faudra donc implémenter une boucle de jeu.

`client/mains.rs` :

```rust
loop {
    // You probably want to:
    // 1. get user input
    // 2. send it to the server
    // 3. fetch game state from the server and draw it
}
```

`server/mains.rs` :

```rust
loop {
    // You probably want to:
    // 1. receive potential user input
    // 2. refresh game state
    // 3. send new game state to the players
}
```

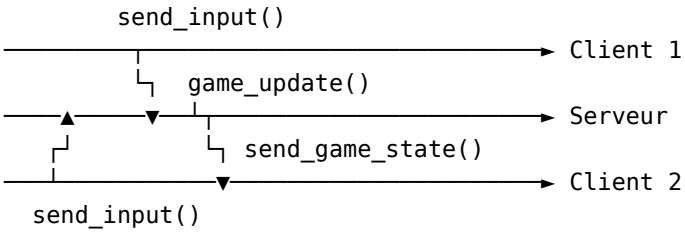
Il faudra également penser à différencier le joueur de gauche et le joueur de droite.

Pour exécuter le client `cargo run --bin client`, et le serveur `cargo run --bin server`.

Avant de commencer

1. Pourrions-nous également implémenter le jeu avec le protocole TCP ?
2. Quels avantages présente UDP dans notre cas ?
3. Dessinez un diagramme des communications entre le serveur et vos deux clients afin de réfléchir à la logique d'exécution et de communication.

Exemple de diagramme avec des pseudo-fonctions :



C'est à vous !

Pour aller plus loin

4. Comment pourrions-nous tricher sur ce jeu ?
5. Les clients pourraient-ils appeller eux-mêmes la fonction update et envoyer le game state a serveur ? Quel serait l'impact sur le jeu ?
6. Est-ce une bonne idée d'envoyer le game state en entier ? Comment cela pourrait être problématique sur d'autres jeux ? Comment Valorant, LoL, cs-go gèrent cela ?

7. Avez-vous constaté de la latence sur vos inputs ? Comment amméliorer ce point ?

Pour aller encore plus loin sur ce dernier, voici un excellent blog post par Gariel Gambetta: https://www.gabrielgambetta.com/client-server-game-architecture.html

Et une démo montrant les différentes solutions pour réduire la latence: https://www.gabrielgambetta.com/client-side-prediction-live-demo.html