Lycée de l'Hyrôme

24h du code : le puissance IV

Introduction

Vous connaissez le principe du puissance 4 : deux joueurs s'affrontent sur une grille de 7 colonnes et 6 lignes. Chaque joueur positionne tour après tour un jeton de couleur qui tombe dans une colonne. La partie est gagnée lorsqu'un des deux joueurs arrive à aligner 4 jetons de sa couleur.

Vous avez à votre disposition un serveur qui fait tourner le jeu. Ce dernier est déjà réalisé, vous n'avez donc pas à le refaire. Vous devez en revanche développer le client pour interagir avec le serveur afin de jouer.

Comment faire communiquer le client et le serveur ?

Le client et le serveur devront communiquer en utilisant des sockets TCP. Il s'agit simplement d'un tuyau entre deux machines sur un port bien précis. Dans notre cas, le serveur écoute sur le port 2023. Les messages que vous lui envoyez devront donc partir sur ce même port. L'adresse IP du serveur vous est donnée à l'oral. Avec ces deux informations vous êtes capables d'envoyer des messages au serveur et d'en recevoir.

L'objectif de ce sujet est entre autres de vous laisser la possibilité de choisir le langage utilisé pour créer le client. Nous vous laissons donc vous renseigner sur la manière dont on peut envoyer et recevoir des messages via une socket TCP dans le langage que vous aurez choisi.

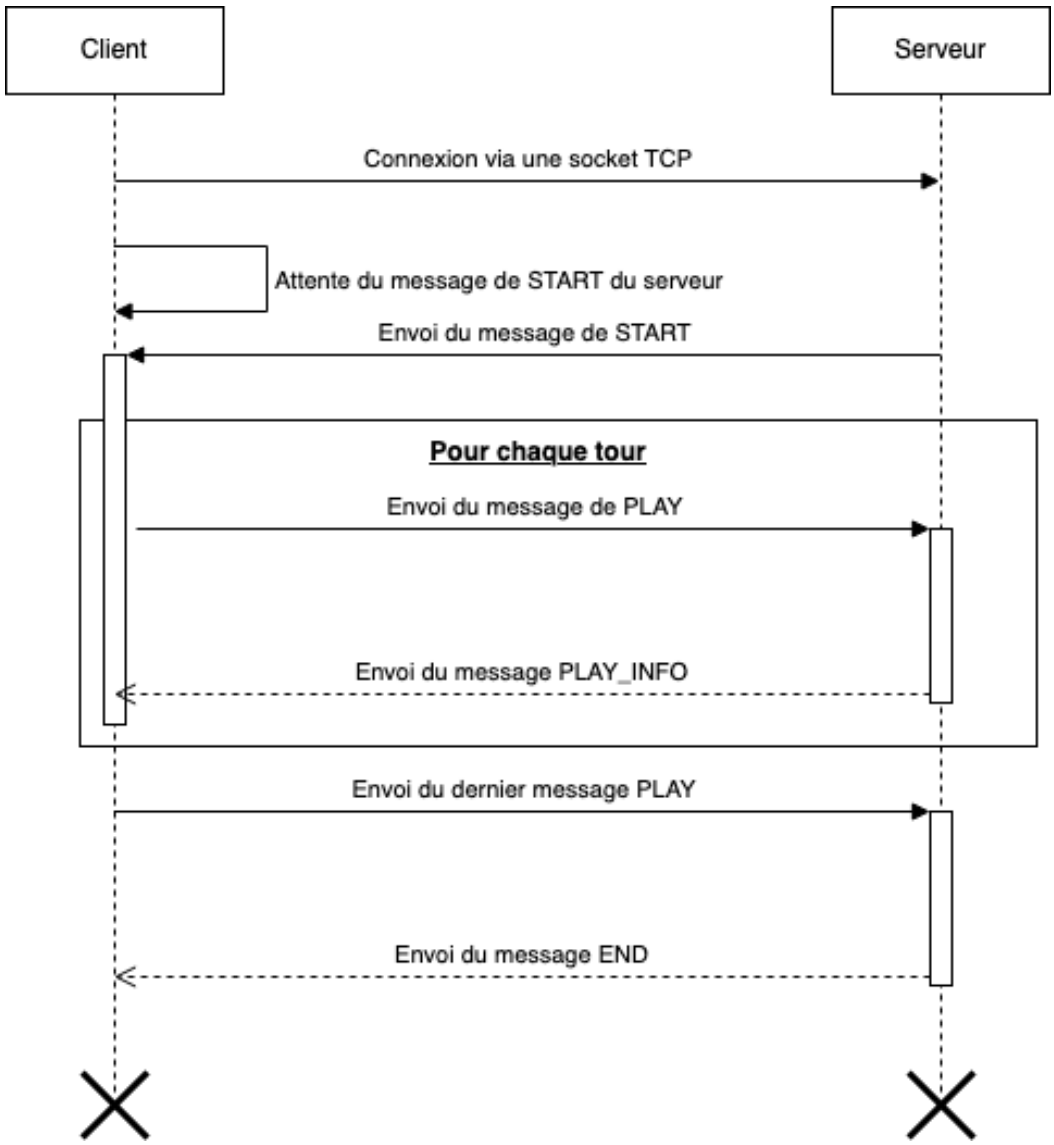
Comment les messages sont-ils structurés ?

Tous les messages échangés doivent être formattés en JSON. C'est un format qui permet de structurer des données dans un message. De la même manière que pour les sockets, nous vous laissons vous documenter sur la manière de formatter un message en JSON dans votre langage de programmation.

Voici le fonctionnement du protocole auquel le serveur s'attend :

1ère étape : le client se connecte au serveur. Vous voyez donc sur le serveur (projeté au tableau) apparaitre une ligne donnant votre adresse IP et le port sur lequel vous êtes.
2ème étape : le client attend un message de START de la part du serveur qui indique aux clients que tout le monde est connecté et que la partie commence.
3ème étape : les clients jouent, tour après tour en envoyant un message PLAY. Le serveur répond alors à chaque fois pour donner l'état actuel de la partie avec un message PLAY_INFO. Si une erreur s'est produite, le serveur répond avec un message ERR.
4ème étape : dès que la partie est terminée, le serveur envoie un message END aux clients en donnant l'information du gagnant.

Vous trouverez ci-dessous un schéma représentant les échanges entre le client et le serveur.



Tous les messages sont structurés en JSON et le premier attribut est toujours le type du message. Voici tous les types de messages possibles :

Type du message	Description	Exemple
START	C'est le premier message que le serveur envoie. Il est envoyé avec un champ booléen tiré au sort « play » qui détermine si c'est à votre tour de jouer ou non. Il contient également un champ you et other correspondant respectivement à votre identifiant de joueur et celui de votre adversaire	`{"type": "START", "play": True, "you": "172.16.4.2315476", "other": "172.16.4.2415477"}`
PLAY	Ce message est envoyé par le client pour le serveur. Il permet de jouer. Le champ « payload » indique la colonne voulue (la première étant la colonne 0).	`{"type": "PLAY", "payload": "2"}`
PLAY_INFO	Ce type de message veut dire que quelqu'un vient de jouer. Il contient le champ « play » (le même que dans le message START) ainsi qu'un champ « payload » donnant l'état actuel du jeu (tableau à deux dimensions). Pour chaque case, le tableau contient 0 si la case est vide ou l'identifiant du joueur qui a posé le jeton	`{"type": "PLAY_INFO", "play": False, "payload": [[0, 0, 0, ...], [...], ...]}`
END	Ce type de message informe aux joueurs que la partie est terminée. Il contient un champ « win » qui vous informe si vous avez gagné ou non. En cas de match nul, les deux joueurs reçoivent le champ win à False.	`{"type": "END", "win": True}`
ERR	Cela signifie qu'une erreur s'est produite et que votre tour n'a pas été pris en compte. Ce message contient un champ « payload » qui vous donne une indication sur la nature de l'erreur.	`{"type": "ERR", "payload": "BAD_INPUT"}`

Ci-dessous, le tableau avec la liste des erreurs possibles dans le champ « payload » des messages ERR.

Nom de l'erreur (champ payload)	Description
WAIT_FOR_PLAYER	Attente de l'autre joueur : vous ne pouvez pas jouer car l'autre joueur n'a pas encore joué.
BAD_PAYLOAD	Le message envoyé n'est pas formaté correctement, et est donc incompréhensible par le serveur. Par exemple, le message n'est peut-être pas en JSON.
BAD_INPUT	Le message envoyé est formaté correctement mais son contenu n'a pas de sens. Par exemple, vous essayez de jouer sur une mauvaise colonne.
GAME_NOT_STARTED	Il n'y a pas encore assez de joueurs sur la partie, elle n'a donc pas encore été lancée.

Le sujet : 1ère partie

Dans un premier temps, il vous est demandé de réaliser un programme dans le langage de votre choix permettant à un utilisateur de jouer à une partie de puissance 4, en utilisant le serveur mis à votre disposition. Quel que soit le langage, votre code devra donc se connecter au serveur grâce à une socket TCP, puis formatter et envoyer correctement vos messages afin de jouer avec un adversaire. Vous n'avez pas besoin de créer une interface graphique, une interface en lignes de commandes suffit.

Le sujet : 2ème partie

Une fois la 1ère partie réalisée et testée, modifiez votre code pour que le programme joue à votre place. Vous devez alors anticiper les coups de votre adversaire, tout en essayant d'aligner 4 jetons de la même couleur.

Important : l'évaluation se concrétisera par des matchs entre les programmes des différentes équipes. Si vos algorithmes sont tous performants, vous allez vous retrouver en situation de blocage (il y aura match nul et on ne pourra pas déterminer le gagnant). Dans ce cas précis, le serveur se basera sur vos temps de réponse respectifs. Celui qui aura le plus faible remportera la partie. Nous attirons donc votre attention sur le fait de produire un code le plus simple et le plus optimisé possible, qui minimise donc les temps de réponse.