Glaziou Guilhem – PC 3

Projet Java

- UNO en réseau -

Version : toutes les cartes chiffres et +2 seulement.

Class Client :

Pour le client, j'ai essayé de faire quelque chose de sécurisé en termes de consignes rentrées par l'utilisateur afin de s'assurer de respecter le protocole de communication avec le serveur. Ainsi, le client sauvegarde et édite en local la classe joueur permettant de garder en local les valeurs du talon, sa main et son score. Il s'agit bien d'une copie, le serveur réalisé garde en local toutes les informations nécessaires pour la partie afin d'éviter des clients « malveillants ».

Utilisation de ClientMain :

La Class ClientMain permet de se connecter en local puis de sélectionner manuellement le port de connexion au serveur via la console. La valeur par défaut est le port 6789. Une fois le port sélectionné, il faut rentrer le nom du joueur qui sera ajouté à « je-suis » pour suivre le protocole de communication avec le serveur. Si la connexion est établie, vous recevrez « bienvenue » de la part du serveur ainsi que « Vous êtes entré dans la partie » de la part du client affiché dans la console. Une fois les cartes reçues, vous pouvez suivre les indications affichées par le client, celui-ci affiche à chaque début de tour sa main et le talon de la partie.

Utilisation en créant son propre ClientMain :

Pour créer sa propre class ClientMain il faut initialiser un client avec soit un constructeur ne prenant que l'ip (String) (le port est alors demandé à l'exécution de la classe) soit un constructeur spécifiant dans l'ordre l'ip (String) puis le port (int). Une fois le client initialisé, il faut lancer la lecture de message pour jouer avec la fonction .jouer() . Une fois l'exécution de .jouer() pour le client initialisé précédemment il faut utiliser sur celui-ci la fonction .close() pour fermer toutes les connexions.

Propriété du client :

Les actions que peut envoyer le client sont restreintes à celles permise par le protocole. Les actions possibles sont rappelées par le client dans la console à chaque tour.

Si vous posez une carte que vous ne possédez pas, le client bloque et demande une nouvelle carte (vous pouvez ne taper que la nouvelle carte mais également passer ou piocher à ce moment selon vos actions précédente).

Si vous souhaitez poser une carte non jouable, le client vous demande une confirmation en répondant « oui » ou « non » et vous permet de changer d'action (nouvelle carte, piocher ou passer, selon vos actions précédentes) si vous répondez non.

Class Serveur :

La classe serveur garde en local toutes les informations sur la partie (Clients, joueurs, nombre de manches et nombre de joueurs) en local et lance l'une après l'autre les manches contenant un duplicata de la liste des joueurs, leurs mains, la pioche et le talon. Les clients se connecte en séquence sur des ports successifs (pas d'utilisation de threads). Avant d'envoyer « joue » au joueur suivant, le serveur vérifie que la manche n'est pas terminée.

Utilisation avec ServeurMain

Pour utiliser ServeurMain après exécution en tant qu'application java, il faut choisir dans la console le premier port (6789 par défaut) de connexion qui définit les ports suivant (autant de ports que de joueur pour ne pas utiliser de threads). Il faut entrer le nombre de joueur à attendre par le serveur (2 par défaut) et enfin entrer le nombre de manche à jouer (1 par défaut). Le serveur attend alors la connexion d'un client sur le premier port, le deuxième, etc… Les clients doivent donc être lancé dans l'ordre avec des ports croissants. Une fois que tout le monde est connecté, le serveur est autonome et n'affiche seulement qui joue dans la console ainsi que qui fait piocher combien à qui si un joueur pose une carte +2.

Utilisation en créant son propre ServeurMain

Pour créer son propre ServeurMain avec cette class Serveur, il faut initialiser un serveur avec le constructeur sans argument de cette class. Ce constructeur se charge de tout jusqu'au lancement de la partie. Il faut alors ensuite lancer sur le serveur la fonction .joueurPartie(int i) ou i définit le type de deck voulu, cad 1 -> une pioche avec un seul type de carte (2-rouge), 2 -> le deck du scénario du serveur de test, et 0 ou 3+ -> une nouvelle pioche aléatoire. Cette fonction communique avec tous les joueurs et est autonome. Un fois la partie terminée, il faut fermer les sockets avec la fonction .close() à appliquer sur le serveur généré.

Propriétés du serveur :

Si le client se connectant ne respecte pas le protocole avec « je-suis nom » alors le nom du joueur est alors tout le message.

Si une manche commence par un +2, le premier joueur pioche 2 cartes et le second commence.

Si un joueur termine avec un +2, le joueur suivant récupère 2 cartes avant que le comptage des points ne soit fait pas le serveur.

L'utilisation de connexions séquencées sur le serveur ne permet pas d'envoyer « serveur plein » à un client de trop voulant se connecter mais celui-ci ne recevra pas « bienvenue » ce qui permet de savoir si la connexion a bien été établit côté client.

Remarques :

Je n'ai pas réussi à lancer le client de test et j'ai donc réalisé les tests avec mon propre client. Celui-ci ne pouvant envoyer que des messages suivant le protocole de communication, j'ai considéré que le serveur respectait également le protocole.