Cours 3

La partie réseau

Sommaire

Section 3

Les sockets

Client et serveur

Nous allons créer un programme qui établi une communication entre deux machines.

Les deux machines pourront s'échanger des messages à tour de rôle.

Le programme installé sur l'une de ces machines jouera le rôle d'un serveur.

Le programme installé sur l'autre jouera le rôle d'un client.

Serveur

Le serveur fonctionne en continu sur une machine dont l'identité est bien définie sur le réseau grâce à une adresse IP spécifique.

Il guette en permanence l'arrivée de requêtes expédiées par les clients potentiels par l'intermédiaire d'un port de communication bien déterminé.

Client

Le client tente d'établir la connexion en émettant une requête appropriée.

Cette requête est un message qui est confié au réseau.

Pour que la destination visée puisse être atteinte, la requête contient l'adresse IP et le port de communication destinataires.

Lorsque la connexion est établie avec le serveur, on peut considérer qu'un canal privilégié relie les deux machines. L'échange d'informations proprement dit peut commencer.

Sockets

Pour utiliser les ports de communication réseau, les programmes font appel à un ensemble de procédures et de fonctions du système d'exploitation.

Ceci est fait par l'intermédiaire d'objets interfaces que l'on appelle donc des sockets.

Ceux-ci peuvent mettre en oeuvre deux techniques de communication différentes et complémentaires :

- celle des paquets (que l'on appelle aussi des datagrammes) ; et
- celle de la connexion continue, ou stream socket, qui est un peu plus simple.

Le client

Voire fichiers :

- `Client.java`
- `TestClient.java`

Les serveurs

Voire fichiers :

- `RegServer.java`
- `ComServer.java`
- `TestServer.java`

Section 4

La partie réseau du projet

Protocole du client

```
send 'register <nom>'
msg = receive
if msg != 'ack <num>':
    termine communication
    return
msg = receive
if msg == 'start <start_str>':
    while (msg != 'end'):
        msg = receive
        if msg == 'error <err_str>':
            traiter...
        elif msg == 'move <num_client> <move_str>':
            traiter...
        elif msg == 'play':
            demander coup au joueur
            send 'move <move_str>' or send 'exit'
        elif msg == 'ask <ask_str>':
            demander reponse au joueur
            send 'respond <resp_str>'
        elif msg == 'info <info_str>':
            traiter...
        else:
            break
send 'exit'
termine communication
```

Protocole du client

`'register <nom>'`
Demande de se connecter à une partie.

`'exit'`
Informe le serveur que le joueur a quitté la partie.
Le serveur doit informer les autres clients que quelqu'un a quitté la partie et fini la partie.

`'move <move_str>'`
Demande au serveur d'exécuter un coup.

`'respond <resp_str>'`
Repond à la suggestion faite par un autre joueur.

Protocole du serveur d'inscriptions

```
1 while (pas trop de clients) and (inscr. encore ouverte):
2     msg = receive
3     if msg == 'register <nom>':
4         create player
5         send player 'ack <num>'
6 for p in players:
7     send player 'start <start_str>'
```

Protocole du serveur d'inscriptions

`'ack <num>'`

Confirme l'enregistrement avec le numéro d'inscription.

`'start <start_str>'`

Informe le début de partie avec les cartes pour chaque joueur.

Syntaxe des chaînes

`start_str`

- `<card_1> <card_2>... <card_n>`

Chaque `<card_i>` est le nom d'une carte du jeu. Exemples :

- Plum
- Scarlett
- Pipe
- Dining
- ...

Protocole du serveur de communication

```
1  while (partie n'est pas finie):
2      player = prochain joueur
3      send player 'play'
4      msg = receive player
5      if msg == 'exit':
6          send all 'error exit <num_client>'
7          break
8      elif msg = 'move <move_str>':
9          if (coup valide):
10             send all 'move <num_client> <move_str>'
11             answers = les reponses a la suggestion
12             for each a in answers:
13                if (a = cannot help):
14                    send all 'info <info_str>'
15                else:
16                   send player 'info <resp_str>'
17                   send all 'info <info_str>'
18         else:
19             send 'error <err_str>'
20     else:
21         send 'error <err_str>'
22 send all 'end'
23 termine communications
```

Protocole du serveur de communication

`'play'`
Informe le client qui est sont tour de jouer.

`'error <err_str>'`
Informe une erreur.

`'move <num_client> <move_str>'`
Informe un coup joué par un joueur.

`'ask <ask_str>'`
Demande quelle carte le joueur va montrer.

`'info <info_str>'`
Donne une information sur je match.

`'end'`
Informe que la partie a terminée.

Syntaxe des chaînes

error_str

- `exit <num_client>`
  Informe que joueur <player> a quitté la partie.
- `forbidden <msg>`
  Informe que le coup n'est pas permis.
  `<msg>` (optionnel) contient les détailles.
- `invalid <msg>`
  Informe que le coup est invalide.
  `<msg>` (optionnel) contient les détailles.
- `other <msg>`
  Informe toute autre erreur.
  `<msg>` (optionnel) contient les détailles.

Syntaxe des chaînes

`move_str`

- `suggest <card_1> <card_2> <card_3>`
- `accuse <card_1> <card_2> <card_3>`

`ask_str`

- `<card_1> <card_2> <card_3>`

où :

- `<card_i>` est le nom d'une carte.

Syntaxe des chaînes

`info_str`

- `noshow <player_1> <player_2>`
  Le joueur 1 ne peut pas aider le joueur 2, parce qu'il n'a pas les cartes de la suggestion.
- `show <player_1> <player_2>`
  Le joueur 1 a montré une carte au joueur 2.

où :

- `<player_i>` est le numéro d'un joueur.