Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2009-2010**

Examen du 15 décembre 2009

Exercice 1 : Matrices de calcul en RMI, application au jeu de la vie

Le but de cet exercice est d'exposer en RMI un service de calcul permettant de lancer une fonction sur les éléments d'une matrice d'entiers. L'interface `CalculRMI` suivante décrit le service du point des types et de l'arité de la méthode.

```
public interface CalculRMI extends Remote {
  int[][] calcule (int [][] m) throws RemoteException;
}
```

On utilisera un tel service pour calculer les générations des automates cellulaires du jeu de la vie.

Rappel sur le jeu de la vie
Le jeu de la Vie est un automate cellulaire visant à simuler l'évolution d'une population de cellules dans leur environnement à partir d'un ensemble de règles.

- L'environnement est un rectangle de cases dont les bords opposés peuvent être adjacents (le monde est un tore : chambre à air)
- apparition : si une case vide a exactement 3 voisins
- disparition : si une case pleine a moins de 2 ou plus de 3 voisins.

Seule l'ancienne génération doit être prise en compte pour le calcul de la nouvelle génération.

```
-------------------------------------------------
| I  |    |    |    |    |    |    |    |  Y |
-------------------------------------------------
| I  |    |    |    |    |    |    |  I |    |  I |
-------------------------------------------------
|    |    |    |  I |    |  I |    |    |    |    |
-------------------------------------------------
|    |    |    |    |  X |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |  I |    |    |  I |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |    |
-------------------------------------------------
|    |    |    |    |    |    |    |    |    |  I |
-------------------------------------------------
```

La cellule `X` possède 3 cellules dans son voisinage (3x3)
La cellule `Y` possède 4 cellules dans son voisinage (3x3)

1. Ecrire une classe `NbNombreVoisins` qui implante l'interface `CalculRMI` et dont la méthode `calcule` compte pour chaque case de la matrice `m` son nombre de voisins. Chaque case de la matrice `m` contient un 0 ou un 1 indiquant l'absence ou la présence d'un élément. Compter le nombre de voisins consiste à compter le nombre de 1 des 8 cases entourant une case. Toutes les case possèdent 8 voisins, le voisin d'un bord est la case à l'opposé sur la même ligne ou la même colonne, idem pour les coins. La matrice résultat est une matrice de même taille que l'argument contenant le nombre de voisins de chaque case de la matrice argument.
2. Ecrire une classe `NouvelleGeneration` qui implante l'interface `CalculRMI` et dont la méthode `calcule` effectue les apparitions et les disparitions de la nouvelle génération selon les règles du jeu de la vie à partir d'une matrice de nombre de voisins passée comme argument.
3. On suppose une instance de `NbVoisins` et une instance de `NouvelleGeneration` qui sont exposées sur le serveur `exam.pc2r.fr` sous les noms `onv1` et `ong1`. Ecrire un programme client qui calcule les 20 premières générations à partir d'une configuration connue et les affiche toutes en utilisant les ressources de calcul proposées par ces deux objets distants.
4. Pour le calcul des nouvelles générations, on cherche à lancer un thread par ligne de calcul de la nouvelle génération. Pour cela on hérite de la classe `NouvelleGeneration` et on redéfinit la méthode calcule héritée. Le calcul de la nouvelle génération de chaque ligne est alors effectué par un thread spécifique. Quand tous les threads sont terminés, alors le résultat de la nouvelle génération peut être envoyé. Indiquer avant de l'implanter comment effectuer la synchronisation des calculs.
5. On cherche maintenant à utiliser un mécanisme de rappel qui effectue un effet de bord sur le client pour chaque case de matrice calculée dont le calcul est terminé. Pour cela définir une nouvelle interface, appelée MRCalculRMI, pour ces services de calcul avec mécanisme de rappel. Implanter ensuite une nouvelle classe de calcul de voisins implantant cette nouvelle interface.
6. Modifier en conséquence le programme principal pour le calcul des voisins, en particulier indiquer comment le client se synchronise avant la fin du calcul distant.
7. On cherche alors à modifier le calcul threadé en ligne de nouvelles générations en utilisant le mécanime de rappel. L'idée est d'alors d'affecter une ligne complète sur le client dès que le serveur a fini de la calculer. Indiquer les modifications à apporter au service de calcul de nouvelle génération pour intégrer un mécanisme de rappel et au programme principal pour tenir comtpe de ces modifications.
8. Plutôt que de repasser par le client à la fin du calcul des voisins, et pour profiter des calcul threadés par ligne, proposer une architecture qui permette d'utiliser un objet exposé de calcul de voisins threadé par ligne par un service de calcul de nouvelle génération.

Exercice 2 : Jeu iBataille en client/serveur (plusieurs langages)

On cherche à implanter un jeu réseau à n joueurs, appelé iBataille, en utilisant un premier langage pour le serveur et un deuxième langage différent pour un client.

Chaque joueur possède n cartes numérotées de 1 à n. A chaque tour chaque joueur dépose une carte. La carte la plus forte gagne sur les autres les n points du tour. En cas d'égalité les n points du tour sont à partager sur l'ensemble des joueurs ayant déposé la carte la plus forte. Le nombre de tours correspond au nombre de jouers. La partie se termine à la fin des n tours.

Le jeu démarre quand il y a n joueurs connectés. Un jeu comprend alors n tours de jeu. Si un joueur se déconnecte en cours de partie, cela ne doit pas influencer le déroulement du reste de la partie. Pendant une partie aucun nouveau joueur ne peut s'y connecter.

Partie A : le serveur

Le serveur doit gérer les tâches suivantes :

- attendre que tous les joueurs (le nombre est fixé ici à n) soient connectés, leur donner un identifiant (numéro unique) puis leur indiquer le début de partie ;

- faire pour n tours :
  - écouter la proposition de chaque joueur en vérifiant que le coup proposé n'a pas encore été joué par ce joueur,
  - dès que tous les joueurs ont joué, envoyer l'ensemble des coups joués à ce tour à tous les autres joueurs et passer au tour suivant.

1. Définir un protocole (celui-ci pourra s'étendre en cas de besoin au fur et à mesure des question) indépendant du langage permettant d'indiquer le début de partie, le début d'un tour, le coup d'un joueur, la validité ou non du coup d'un joueur, les coups de tous les joueurs à un tour donné, la fin du tour et la fin de la partie.
2. Implanter dans le langage de votre choix (O'Caml, Java ou C) un serveur du jeu iBataille. Vous pouvez, en les citant, utiliser des morceaux de code issus du polycopié. Au début de cette question indiquer l'architecture logicielle que vous allez employer, et indiquer le type des méthodes/fonctions principales de votre développement.
3. On cherche à limiter la durée d'un coup. Indiquer l'ajout d'un temps limite pour la proposition d'un coup d'un joueur. Si un joueur dépasse ce temps son coup vaut 0.

Partie B : le client

Le client qui se connecte, reçoit un identifiant, puis attend le début de partie. Ensuite pour n tours, il joue une carte dans son jeu de carte allant de 1 à n. Attention une carte ne peut être jouée qu'une seule fois.
Quand le tour est terminé, les coups de tous les joueurs sont envoyés à tous les joueurs et on passe au tour suivant. Quand les tours sont finis, la partie est déclarée terminée.

4. Ecrire un client simple, dans un autre langage que celui du serveur, qui joue de manière systématique de la plus faible à la plus forte carte pendant les n tours.
5. Ajouter une gestion des scores. A la fin du tour quand le serveur envoie l'ensemble des coups joués, chaque joueur calcule le score du tour et le score des tours cumulés de la manière suivante : le joueur ayant soumis la plus forte carte gagne les n points, en cas d'égalité les n points sont divisés (division entière) par le nombre de joueurs ayant proposé la plus forte carte.
6. On cherche à améliorer le client pour qu'il joue en fonction des cartes déjà posées des autres joueurs. Pour cela on conservera pour chaque joueur l'ensemble des coups joués. Proposer une nouvelle stratégie du jeu pour tenir compte des coups joués et modifier le client en conséquence.