Master d'informatique | **module PC2R** | **Université Pierre et Marie Curie**
Documents autorisés | *Durée 2 heures* | **Année 2007-2008**

Examen du 17 décembre 2007

Exercice 1 : Robots dans un damier numéroté (en Esterel)

Le but de cet exercice est de programmer des robots se déplaçant sur un damier de n cases numéroté de 0 à $n-1$, voir image suivante pour un damier 5x5 :

Module robot chaque robot est défini de la manière suivante :

- Sa position initiale est fixé à -1, indiquant qu'il est en dehors du damier. Une fois dans le damier, sa position varie entre 0 et la taille du damier élevée au carré moins 1 (pour le damier 5x5 ci-dessous, sa position varie entre 0 et 24).
- Pour éviter les problèmes de collision et pour simplifier, il n'a pas de capteur pour détecter la présence d'un autre robot ou les bords du damier.
- Une fois dans le damier et à chaque `tick`, il peut aléatoirement soit rester sur place, soit se déplacer d'une case vers le nord, le sud, l'est ou l'ouest. Plus exactement, il fait appel à la fonction `calculerNouvellePosition()` en donnant son numéro et sa position courante et la fonction retourne aléatoirement sa prochaine position en évitant toute collision ou toutes places déjà prises.
- Si le paramètre `position_courante` est -1 et si la case 0 est libre, la fonction `calculerNouvellePosition()` retournera la valeur 0, sinon elle retournera la valeur de `position_courante` (c'est à dire -1).
- A chaque `tick`, il signale sa position en émettant un signal P valué avec comme valeur sa nouvelle position.
- Il entre donc dans le damier par la case 0.
- A la réception du signal `RETOUR`, il tente à chaque `tick` et case par case d'aller vers la case 0 en se dirigeant vers le nord, puis vers l'ouest. Pour cela, il fait appel à la fonction `calculerRetour()` en donnant son numéro et sa position courante et la fonction lui retourne une nouvelle case si cette dernière est libre, sinon sa position courante.
- Si il donne 0 comme position courante, la fonction `calculerRetour()` retournera -1 (ce qui indique que le robot sort du damier et c'est aussi la fin de la tâche du robot).
- Aucun signal ne peut le détourner de son chemin de retour.

1. Ecrire le module `robot` ayant les propriétés et les fonctions indiquées ci-dessus. Les fonctions `calculerNouvellePosition()` et `calculerRetour()` sont définies en C.

Module démarrage C'est un dispositif qui gère 3 robots indépendants numérotés de 1 à 3. Il fonctionne de la manière suivante :

- A la réception du signal `DEMARRER`, il démarre les 3 robots.
- Au signal `RETOUR`, tous les robots prennent le chemin de retour et finissent leurs tâches. Après que tous les robots aient fini leurs tâches, on peut redémarrer à nouveau avec le signal `DEMARRER`.
- On devra voir pour le robot 1 (resp. 2 et 3) afficher à la sortie `P1(x)` (resp. `P2(x)` et `P3(x)`) où `x` est la position du robot.

2. Ecrire le module `demarrage` ayant les propriétés et les fonctions indiquées ci-dessus.
3. On souhaite suspendre en entrant (ici manuellement) un ou plusieurs signaux `ARRET` consécutifs et reprendre seulement dans un ou plusieurs `tick` plus tard avec le signal `REPRISE`. Compléter le ou les modules pour cette suspension/reprise.

Un exemple de session		
début	suite	fin

`$ exo-esterel`
`demarrer> DEMARRER;;;;`
`--- Output: P1(0)`
`--- Output: P1(0)`
`--- Output: P1(1) P2(0)`
`--- Output: P1(6) P2(0)`
`demarrer> ARRET;;;REPRISE;;;`
`--- Output:`
`--- Output:`
`--- Output:`
`--- Output: P1(11) P2(0)`
`--- Output: P1(12) P2(5) P3(0)`
`--- Output: P1(17) P2(10) P3(1)`
`demarrer> RETOUR;ARRET;;;;`
`--- Output: P1(12) P2(5) P3(0)`
`--- Output:`
`--- Output:`
`--- Output:`
`--- Output:`
	`demarrer> REPRISE;DEMARRER;;;;;`
	`--- Output: P1(7) P2(5)`
	`--- Output: P1(2) P2(0)`
	`--- Output: P1(1)`
	`--- Output: P1(0)`
	`--- Output:`
	`--- Output:`
	`demarrer> DEMARRER;;;;`
	`--- Output: P1(0)`
	`--- Output: P1(0)`
	`--- Output: P1(1) P2(0)`
	`--- Output: P1(1) P2(0)`
	`demarrer> ;;;;`
	`--- Output: P1(2) P2(5) P3(0)`
	`--- Output: P1(3) P2(10) P3(5)`
	`--- Output: P1(2) P2(15) P3(5)`
	`--- Output: P1(7) P2(16) P3(6)`
		`demarrer> RETOUR;;;;;;;;;;;;;;;`
		`--- Output: P1(2) P2(11) P3(1)`
		`--- Output: P1(2) P2(6) P3(0)`
		`--- Output: P1(1) P2(6)`
		`--- Output: P1(0) P2(1)`
		`--- Output: P2(0)`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`--- Output:`
		`demarrer>`

Exercice 2 : Jeu iNumber en Client/serveur (plusieurs langages)

On cherche à implanter un jeu réseau à plusieur joueurs, appelé iNumbers. A chaque tour les joueurs doivent deviner un nombre caché par l'un d'entre eux. Pendant un jeu complet chaque joueur pourra cacher une fois un nombre. Pour déterminer le nombre caché chaque joueur peut : proposer le nombre ou demander de vérifier une propriété mathématique du nombre caché. Les demandes de chacun, sauf la proposition du nombre juste, sont visibles des autres joueurs. On ne cherchera à implanter que 2 propriétés : *est diviseur de* et *est multiple de*.

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.

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 et leur indiquer le début de partie ;
- faire pour n tours :
  - indiquer le joueur qui choisit le nombre
  - attendre son choix
  - écouter en parallèle les différents choix ou questions des joueurs
  - dès qu'un joueur a trouvé le nombre, le tour est fini

1. Définir un protocole (celui-ci pourra s'étendre en cas de besoin au fur et à mesure des question) permettant d'indiquer le début de partie, le début d'un tour, le joueur qui cache le nombre, le début du tour, le choix d'un nombre, la vérification d'une propriété (multiple de x ou diviseur de x), 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 iNumber en autorisant uniquement la proposition d'un nombre. Vous pouvez, en les citant, utiliser des morceaux de code issus du polycopié.
3. Ajouter la demande de vérification d'une propriété et sa réponse à tous les joueurs.

client se connecter, reçoit un identifiant, puis attend le début de partie. Ensuite pour n tours, il attend le début de tour, sauf s'il est sélectionné comme joueur cachant le nombre (il doit alors répondre). Le joueur cachant le nombre n'interagit plus pendant ce tour. Quand le tour est terminé (un joueur a trouvé le nombre) on passe au tour suivant. Quand les tours sont finis, la partie est déclarée terminée.

4. Ecrire dans un autre langage que celui du serveur un client simple qui ne permet que de proposer un nombre.
5. Enrichisser le en proposant la vérification de propriétés (multiple, diviseur) du nombre caché.

Exercice 3 : Mécanisme de réplication en RMI (en Java)

On cherche à définir un service de réplication d'états locaux (ici des points) à l'exécution d'un programme sur une machine donnée vers un serveur RMI permettant ainsi l'accès à ces états sans perturber l'exécution en cours. Dans la suite on utilisera l'interface suivante :

```
public interface ExposeRMI extends Remote {
  void expose(String name, int x, int y) throws  RemoteException;
}
```

1. Ecrire une interface `PointRMI` pour l'appel RMI de points distants qui possède 3 déclarations de méthodes :
   - mise à jour des coordonnées d'un point `moveto`
   - récupération les coordonnées en x et en y d'un point (`getx`, `gety`).
2. Implanter la classe `PointD` respectant l'interface que vous venez de décrire.
3. On cherche maintenant à définir une classe `ExposeD` qui implante l'interface `ExposeRMI` contenant la méthode `expose` qui prend les coordonnées d'un point et un nom d'exposition, et expose un point distant de ces coordonnées sous le nom indiqué.
4. Ecrire un serveur qui crée un objet de la classe `ExposeD` sous le nom `exp1`. Ecrire les commandes pour lancer le serveur sur la machine `exam.jussieu.fr` (sur laquelle vous êtes connecté) sur le port 2007.
5. On cherche maintenant à modifier la classe `Point` classique (voir polycopié) pour qu'à chaque création d'un point il y ait la création d'un point distant d'un certain nom sur un serveur donné ; ces informations étant passées à la création du point. La référence sur le point distant est conservée dans le point. On utilise l'objet distant `exp1` de la question précédente.
6. Pour tenir compte des modifications locales d'un point, modifier les méthodes `moveto` et `rmoveto` pour signaler au serveur d'exposition de l'état du point que celui-ci vient de changer.
7. Le point exposé possède une méthode de modification d'état, proposer une solution pour que seul le point créateur puisse l'utiliser. Implanter si possible votre solution.