TP2
Élection de leader

Le but de ce second TP est d'implémenter deux algorithmes d'élection de leader dans JBotSim, puis de vérifier leurs performances en mesurant le nombre de messages envoyés.

Commencez par créer un nouveau projet, par exemple par copié-collé du projet du TP1.

1 Création d'une topologie en anneau

Dans le TP1, nous avons utilisé l'interface graphique de JBotSim pour générer la topologie du système distribué à chaque exécution. Dans le TP2, nous allons effectuer des tests de performances en exécutant un même algorithme une centaine de fois d'affilée. Il n'est donc pas envisageable de reconstruire la topologie manuellement à chaque fois. Pour éviter ce problème, nous allons à la place définir la topologie une fois pour toute dans le main de notre programme.

```
Topology tp = new Topology();
// Construction de la topologie ici
new JViewer(tp);
tp.start();
```

1.1 Création des noeuds

La topologie que nous allons construire doit être un anneau unidirectionnel composé de 10 noeuds. Afin que le nombre de noeuds soit facile à modifier, vous pouvez définir une constante comme ceci :

```
public static final int TAILLE_ANNEAU = 10;
```

Les noeuds peuvent être créés tout simplement comme suit :

```
Node noeud = new MaClasseDeNoeuds();
```

Et ajoutés au système comme suit :

```
tp.addNode(coord_x, coord_y, noeud);
```

Les deux paramètres `coord_x` et `coord_y` permettent de positionner le noeud dans le canevas de JBotSim. Le point (0,0) se trouve dans le coin supérieur gauche de la fenêtre. Vous pouvez positionner les noeuds comme vous le souhaitez, mais il faut au moins s'assurer que les différents noeuds ne sont pas superposés (aux mêmes coordonnées).

Question 1. Programmez la construction de la topologie et assurez vous que le système contient le bon nombre de noeuds. Testez votre code.

1.2 Création des liens

Pour le moment, JBotSim est en mode wireless. Cela signifie qu'un lien est automatiquement ajouté entre deux noeuds lorsqu'ils sont placés suffisamment près l'un de l'autre. Ce mécanisme risque de créer des liens non voulus, il faut donc le désactiver.

```
tp.disableWireless(); // À faire juste après la création de la topologie.
```

Nous pouvons ensuite créer les liens entre les noeuds comme suit :

```
tp.addLink(new Link(noeud1, noeud2, Orientation.UNDIRECTED)); // Lien bidirectionnel
tp.addLink(new Link(noeudSource, noeudDestinataire, Orientation.DIRECTED)); // Lien unidirectionnel
```

Dans le cadre de ce TP, nous cherchons à construire un anneau unidirectionnel. Il faut donc que chaque noeud dispose d'un lien unidirectionnel vers un seul autre noeud, de sorte que le système forme une boucle.

Notez que les liens unidirectionnels sont invisibles dans l'interface de JBotSim.

Question 2. Connectez les noeuds de façon à former un anneau unidirectionnel. Vous pouvez vérifier que les connexions sont bien faites en exécutant l'algorithme de diffusion naïf du TP1 (tous les noeuds devraient changer de couleur).

1.3 Mélange des identifiants

JBotSim affecte un identifiant à chaque noeud au moment où il est ajouté à la topologie, en les numérotant de façon croissante. Suivant la façon dont vous avez programmé les questions précédentes, il est donc probable que le noeud 0 dispose d'un lien vers le noeud 1, qui dispose d'un lien vers le noeud 2, etc. Le fait que les noeuds soient triés par identifiants croissants est un scénario très spécifique qui peut fausser les performances des algorithmes d'élection de leader. Il faudrait donc changer les identifiants des noeuds de façon à ce que l'anneau ne soit plus trié. Il faut malgré tout que les identifiants soient uniques et compris entre 0 et 9.

Il y a de nombreuses méthodes pour effectuer ce mélange des identifiants. Une méthode simple consiste à ajouter les numéros de 0 à 9 dans une liste, puis mélanger cette liste avant d'affecter les numéros aux noeuds.

Vous pouvez vous aider des fonctions suivantes :

```
Collections.shuffle(maListe); // Mélange les éléments de maListe au hasard
noeud.setID(3); // Affecte l'identifiant 3 au noeud
```

Attention : c'est au moment de l'ajout d'un noeud dans la topologie que JBotSim affecte un identifiant au noeud. Il faut donc que réaffectation des identifiants se fasse après les appels à `addNode()`.

Question 3. Mélangez les identifiants des noeuds pour éviter que l'anneau ne soit trié.

1.4 Positionnement en cercle (question bonus, à faire si vous êtes à l'aise)

Petit rappel de géométrie. Pour placer un point sur un cercle à un certain angle (en radians), les coordonnées du point doivent être les suivantes :

Abscisse $= cos(angle) \times rayon + abscisse\_centre$
Ordonnée $= sin(angle) \times rayon + ordonnee\_centre$

Vous pouvez vous aider de la fonction suivante :

```
double angleEnRadians = Math.toRadians(angleEnDegres);
```

Question 4. Bonus. Positionnez les noeuds en cercle.

2 Algorithme de Chang-Roberts

Dans un algorithme d'élection de leader, on cherche à sélectionner un unique noeud parmi tous ceux du système distribué. L'algorithme de Chang-Roberts (slides 10-13 du CM3) est un algorithme d'élection de leader déterministe pour un anneau unidirectionnel.

Dans cet algorithme, certains noeuds se portent candidats pour devenir leader. Afin de simplifier les choses, nous allons dans un premier temps présumer que tous les noeuds se portent candidats dès le début de l'algorithme.

Chaque processus garde en mémoire l'identité de son leader actuel. Initialement, les candidats se choisissent eux même comme leader.

Chaque noeud candidat envoie un message de candidature contenant son identifiant à son successeur dans l'anneau. Lorsqu'un noeud reçoit l'identifiant d'un autre candidat dans un message, si cet identifiant est supérieur à son leader actuel, il change de leader et retransmet le message. Sinon, le message n'est pas retransmis.

Lorsqu'un candidat reçoit un message contenant son propre identifiant, c'est que ce message a fait le tour complet de l'anneau sans être contesté. Ce candidat est alors élu ! Il envoie alors un message d'élection qui sera retransmis dans tout l'anneau pour informer les autres noeuds de sa victoire.

L'algorithme utilise deux types de messages : les messages de candidature et les messages d'élection. Dans les deux cas, le message contient un identifiant de processus.

Question 1. Quelles sont les hypothèses de l'algorithme de Chang-Roberts ?

2.1 Implémentation avec tous les noeuds candidats

Dans le TP1, les messages ne contenaient qu'une seule information (le type de message). Ici, les messages devront contenir deux informations : le type de message (candidature ou élection) et l'identifiant du candidat ou leader qui a émis le message. La classe `Message` de JBotSim ne contient qu'un seul objet, il peut donc être utile de définir une classe pour stocker deux variables dans un message.

```
public class ContenuMessage {
    int var1; // Ces deux attributs sont un exemple de contenu d'un message
    double var2;
    public ContenuMessage(int var1, double var2) {this.var1 = var1; this.var2 = var2;}
}

// ... plus loin ...

Message msg = new Message(new ContenuMessage(5, 4.3));
sendAll(msg); // Dans un anneau unidirectionnel, chaque noeud n'a qu'un voisin.
// sendAll() permet donc d'envoyer un message à cet unique voisin.

// ... encore plus loin ...

public void onMessage(Message msg) {
    int entierRecu = ((ContenuMessage) msg.getContent()).var1;
    double doubleRecu = ((ContenuMessage) msg.getContent()).var2;
    // On peut ensuite faire ce qu'on veut avec ces informations.
}
```

Afin de visualiser la progression de l'algorithme dans l'interface graphique de JBotSim, il peut être utile de colorer les noeuds. Il faut que la couleur d'un noeud représente son leader.

```
setColor(Color.getColorAt(identifiantLeader)); // Donne au noeud une couleur suivant son leader
```

À terme, les noeuds devront tous avoir la même couleur (et donc le même leader).

Question 2. Programmez et testez l'algorithme de Chang-Roberts, en présumant que tous les noeuds se portent candidats.

2.2 Mesures expérimentales

Nous voudrions maintenant mesurer les performances de l'algorithme, en comptant le nombre de messages envoyés. Comme dans le TP1, on trichera en utilisant une variable statique pour comptabiliser les messages au fur et à mesure qu'ils sont envoyés.

L'affichage du résultat se fera lors de la terminaison de l'algorithme, une fois que le leader a fini d'informer tout le monde de son élection (donc quand il recevra son propre message d'élection).

Question 3. Affichez le nombre total de messages envoyés à la fin de l'élection.

Le nombre de messages envoyés n'est pas déterministe, puisqu'il dépend de l'agencement des identifiants des noeuds qui sont tirés aléatoirement (Exercice 1.3).

Afin d'obtenir des statistiques fiables, on souhaite exécuter l'algorithme 100 fois d'affilée. Vous pouvez utiliser une seconde variable statique pour compter le nombre d'exécutions écoulées.

Lorsqu'on détecte la terminaison de l'algorithme, il faut donc :

1. Afficher le nombre de messages envoyés lors de cette exécution ;
2. Incrémenter le compteur d'exécutions ;

3. Vérifier si il reste encore des exécutions, ou si les 100 exécutions sont terminées.

Si il reste encore une exécution, il faut alors mélanger à nouveau les identifiants des noeuds (réutilisez le code de l'exercice 1.3) et relancer une nouvelle exécution.

```
getTopology().restart(); // Relance une nouvelle exécution depuis le début
```

Une fois la dernière exécution terminée, il faut afficher le nombre moyen de messages envoyés par exécution, le nombre minimum de messages envoyés sur une exécution, et le nombre maximum de messages envoyés. Pour cela vous aurez besoin de créer des variables statiques supplémentaires.

Afin d'accélérer l'exécution, pensez à commenter ou supprimer les lignes suivantes dans votre main :

```
// On ne veut pas d'affichage graphique
// new JViewer(tp);
// On ne veut pas non plus limiter la vitesse d'exécution
// tp.setTimeUnit(1500);
```

Il est aussi conseillé de supprimer tous les affichages en cours d'exécution.

Question 4. Testez votre programme, et notez le nombre moyen de messages envoyés sur 100 exécutions, ainsi que le nombre minimum et maximum.

2.3 Implémentation avec candidats aléatoires

Jusqu'ici, nous avons présumé que tous les noeuds se portaient candidats dès le début de l'exécution. Dans l'algorithme de Chang-Roberts, ce n'est pas nécessairement le cas : seul certains noeuds sont candidats au début de l'exécution.

Chaque noeud devra maintenant mémoriser son état : candidat ou non.

Lorsqu'un noeud non-candidat reçoit un message de candidature portant un identifiant plus petit que le sien, il décide de se porter candidat et envoie son propre message de candidature.

Au début de l'exécution, chaque noeud décide aléatoirement de se porter candidat ou non. Arbitrairement, on décide que chaque noeud a 20% de chances d'être candidat dès le début. Pour décider si un noeud est candidat, vous pouvez utiliser la fonction suivante :

```
Math.random(); // Retourne un double aléatoire entre 0.0 et 1.0
```

Dans certains cas, il se peut qu'aucun noeud ne se porte candidat. Afin d'éviter ce scénario gênant qui bloquerait l'exécution, on décide que le noeud d'identifiant 0 sera toujours candidat (il y a donc entre 1 et 10 candidats).

Question 5. Modifiez votre programme pour que seuls certains noeuds se portent candidats.

Question 6. À la fin de chaque exécution de l'algorithme, en plus du nombre de messages envoyés, affichez le nombre de candidats initial. Après 100 exécutions, constatez vous une relation entre le nombre de candidats et le nombre de messages envoyés ?

3 Algorithme d'Itai-Rodeh

L'algorithme de Chang-Roberts vu précédemment nécessite que chaque noeud ait un identifiant unique. À l'inverse, Itai-Rodeh (slides 14-17 du CM3) est un algorithme d'élection de leader pour un anneau unidirectionnel anonyme (sans identifiants uniques). Il s'agit d'un algorithme probabiliste, dont la terminaison n'est donc pas garantie. Les noeuds se choisissent un identifiant aléatoirement, et la difficulté est de garantir que deux noeuds n'auront pas pioché le même identifiant.

Comme dans l'algorithme de Chang-Roberts, certains noeuds se portent candidats dès le début. Dans cet exercice, ces noeuds sont à nouveau tirés aléatoirement (20% de chance d'être candidat). Un noeud choisi arbitrairement doit toujours se porter candidat, pour éviter une exécution sans candidat.

L'algorithme fonctionne par phases (il s'agit de rondes asynchrones). À chaque phase, seuls les candidats qui ont tiré l'identifiant le plus grand continuent à la phase suivante. Les autres candidats abandonnent. Si deux candidats tirent perpétuellement le même identifiant, alors l'algorithme ne se termine jamais.

Chaque noeud doit stocker au moins les informations suivantes :

— Le numéro de la phase actuelle (initialement 1) ;
— L'état du noeud (candidat ou non).

Lorsqu'un noeud se porte candidat, il se choisir un identifiant aléatoirement entre 0 et 20. Il envoie ensuite un message à son successeur.

Dans l'algorithme du cours, les messages comportent 4 variables :
- L'identifiant (tiré aléatoirement) du candidat ;
- Le numéro de phase du candidat ;
- Le nombre de fois que ce message a été transmis (1 lors du premier envoi, puis +1 à chaque retransmission) ;
- Un booléen indiquant si l'identifiant du candidat est unique ou non (initialement oui).

Pour des raisons pratiques, on peut ajouter une cinquième variable qui indique si l'élection est terminée ou non.

Dans l'algorithme d'Itai-Rodeh, on présume que chaque noeud connaît le nombre de noeuds dans le système. Vous pouvez récupérer cette information comme suit :

```
getTopology().getNodes().size(); // Retourne le nombre de noeuds dans le système
```

Lorsqu'un noeud qui n'est pas candidat reçoit un message, il se contente de le faire suivre en incrémentant le nombre de fois que ce message a été envoyé.

Lorsqu'un candidat reçoit un message qui a été transmis autant de fois qu'il y a de noeuds dans le système, il sait donc qu'il a reçu son propre message. Si le contenu du message indique que son identifiant est unique, ce noeud est alors élu et peut envoyer un second message pour informer les autres noeuds de sa victoire.

Si le candidat reçoit son propre message mais que son identifiant n'est pas unique, il faut alors incrémenter le numéro de phase, tirer un nouvel identifiant aléatoire et envoyer un nouveau message pour retenter sa chance.

Si un candidat reçoit un message portant son identifiant et son numéro de phase, mais que ce message n'a pas fait le tour complet de l'anneau, cela signifie qu'un autre noeud a tiré le même identifiant aléatoire lors de la même phase. Il faut alors faire suivre le message en passant le booléen à faux : cet identifiant n'est plus unique.

Si un candidat reçoit un message appartenant à une phase supérieure à la leur, ou si le message appartient à la même phase que la leur mais que l'identifiant dans le message est supérieur au leur, alors le noeud abandonne sa candidature et retransmet le message.

Question 1. Programmez l'algorithme d'Itai-Rodeh, et testez le.

On veut maintenant effectuer des mesures de performances, en utilisant les mêmes techniques qu'à l'exercice précédent. Comme précédemment, on peut détecter la terminaison lorsque le leader élu a fini d'informer le reste du système de sa victoire.

Question 2. Testez l'algorithme sur 100 exécutions. Quel est le nombre de messages envoyés en moyenne ? Quel est le nombre de messages minimum, et maximum ? Comment ces valeurs se comparent elles à celles trouvées avec l'algorithme de Chang-Roberts ?