**CENTRE UNIVERSITAIRE DE MILA**

**INSTITUT DES SCIENCES ET DE LA TECHNOLOGIE**

**2ere Année Master STIC et I2A**

**Module: SMA (Résolution distribuée des problèmes)**

**TP4 Les comportement de l’agent (suite)**

 **Les behaviours composés**

**JADE** offre un ensemble de Behaviours composés qui servent à présenter des tâches complexes. La classe mère de toutes les autres complexes est la classe **jade.core.behaviours.CompositeBehaviour**.

Une instance de cette classe, un Behaviour bien entendu, contient des sous-Behaviours, la méthode **action()** est déjà implémentée et invoque à chaque fois la méthode **action()** de l'un de ses sous-Behaviour. L'ordre d'exécution des Behaviours est à la charge des deux méthodes **scheduleFirst()** et **scheduleNext()** que les classes héritant de la classe CompositeBehaviour doivent implémenter.

Il existe trois classes filles:

**1. SequentielBehaviour**

**2. FSMBehaviour**

**3. ParallelBehaviour**

1. **SequentielBehaviour**

Le Behaviour commence par exécuter le premier sous-Behaviour et lorsque celui-là termine son exécution (sa méthode **done()** retourne true), il passe au prochain Behaviour, et ainsi de suite.

Les sous-Behaviours sont ajoutés au sequentielBehaviour par la méthode *addSubBehaviour()*. L'ordre de l'ajout détermine l'ordre d'exécution.

Voici un exemple :

importjade.core.Agent;

importjade.core.behaviours.OneShotBehaviour;

importjade.core.behaviours.SequentialBehaviour;

importjade.core.behaviours.WakerBehaviour;

public class Seq extends Agent {

protected void setup(){

 SequentialBehaviourcomportementSequentiel = new SequentialBehaviour();

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public void action() {

 System.out.println("le premier sous-comportement");

 }

 });

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public voidaction() {

 System.out.println("le second sous-comportement");

 }

 });

 comportementSequentiel.addSubBehaviour(new

 OneShotBehaviour(){

 @Override

 public voidaction() {

 System.out.println("le derniers sous-comportement");

 myAgent.doDelete();}

 });

 addBehaviour(comportementSequentiel);

 }}

**Exercice1**

4 Behaviors qu'on veut exécuter séquentiellement, chacun affiche un message pendant x secondes (10, 20, 5 et 15). Affiche à la fin le nombre total des secondes.

**2. FSMBehaviour**

**FSMBehaviour** est une sorte de Behaviour qui implémente un automate à états finis dont chaque état correspond à l'exécution d'un sous-Behaviour.

L'introduction de l'automate se fait de la manière suivante :

* L'ajout d'un nouveau état se fait par la méthode **registerState** (Behaviour state, String name) ;
* L'ajout de l'état initial (il n'existe qu'un seule état initial) se fait par la méthode **registerFirstState** (Behaviour state, String name );
* L'ajout d'un état final (il est possible d'en avoir plusieurs) se fait par le méthode **registerLastState** (Behaviour state, String name).
1. State : le Behaviour qui représente l'état;
2. Name: le nom de l'état.
* L'ajout d'une nouvelle transition se fait par la méthode **registerTransition**(String s1, String s2, intevent);
* L'ajout d'une transition par défaut (la seule transition entre deux états ou bien la transition à prendre si aucune autre n'est prise) se fait par la méthode **registerDefaultTransition**(String s1, String s2, String[] toBeReset)
1. s1 : l'état source;
2. s2 : l'état destination;
3. event: l'étiquette de la transition;
4. String[] toBeReset : l'ensemble d'état pour lesquels on doit faire un reset() avant de les ré-exécuter parcequ'on peut pas re-éxecuter un Behaviour une autre fois avant de lui faire un reset().

Exemple : considérons l'automate suivant :

 le FSM Behaviour qui le représente :

importjade.core.Agent;

importjade.core.behaviours.OneShotBehaviour;

importjade.core.behaviours.FSMBehaviour;

public class Agent\_FSM extends Agent{

 protected void setup(){

 FSMBehaviourfsm = new FSMBehaviour(this) {

 public intonEnd() {

 System.out.println("FSM behaviourterminé");

 myAgent.doDelete();

 returnsuper.onEnd();

 }

 };

 //definiton des etats

 fsm.registerFirstState (new UnComportement(),"A");

 fsm.registerState(new UnComportement(),"B");

 fsm.registerState(new UnComportement(),"C");

 fsm.registerLastState(new UnAutreComportement(),"D");

 //definitiondes transaction

 fsm.registerDefaultTransition("A","B");

 fsm.registerTransition("B","B",1);

 fsm.registerTransition("B","C",0);

 fsm.registerTransition("C","C",1);

 fsm.registerTransition("C","D",0);

 addBehaviour(fsm);

 }

 //Inner Class

 private class unComportement extends OneShotBehaviour{

 int aleatoire;

 @Override

 public void action() {

 System.out.println("execution de l'etat "+ getBehaviourName());

 aleatoire = (int)Math.random() \* 1;

 }

 public intonEnd(){

 return aleatoire;

 }

 }

 private class unAutreComprtementextendsOneShotBehaviour{

 @Override

 public voidaction() {

 System.out.println("arrivée à l'etat finale");

 }

 }

}

**3 ParallelBehaviour**

Le parallelBehaviour permet d'exécuter plusieurs Behaviours en parallèle. Par parallèle on comprend qu'après avoir invoqué la méthode **action()** d'un sous-Behaviour, on pointe sur le suivant sans attendre que le premier termine son exécution. L'ajout de sous-Behaviour se fait par la méthode **addSubBehaviour()** . Si on veut que le parallelBehaviour termine dès qu'un de ses sous-Behaviours termine alors on doit passer à son constructeur l'argument **WHEN\_ANY**. Pour attendre la fin de tous les sous-Behaviours on doit lui passer l'argument **WHEN\_ALL**.

Exemple

importjade.core.Agent;

importjade.core.behaviours.OneShotBehaviour;

importjade.core.behaviours.ParallelBehaviour;

importjade.core.behaviours.TickerBehaviour;

importjade.core.behaviours.WakerBehaviour;

 public class agent\_parallel extends Agent{

 protected void setup(){

 ParallelBehaviourcomportementparallele = new ParallelBehaviour(ParallelBehaviour.WHEN\_ANY);

 comportementparallele.addSubBehaviour(new WakerBehaviour(this,1000){

 @Override

 protectedvoidhandleElapsedTimeout() {

 System.out.println("le temps est écoulé..");

 myAgent.doDelete();

 }

 });

 comportementparallele.addSubBehaviour(new TickerBehaviour(this,50){

 intaleatoire;

 @Override

 protected void onTick() {

 aleatoire=(int) (Math.random()\*100);

 System.out.println("aleatoire = "+aleatoire);

 if(aleatoire==5)

 {

 System.out.println("Bingo!");

 myAgent.doDelete();

 }

 }

 });

 addBehaviour(comportementparallele);

 }

 }

**4. Combiner plusieurs Behaviours composés**

Il est possible de combiner les différents comportements composés pour créer des comportements plus complexes.

Par exemple on peut combiner le **FSMBehaviours** et le **parallelBehaviour** précédents de la manière suivante :

Attribuer à l'état C le **parallelBehaviour**. Si le Behaviour réussit à générer le 5 dans une second alors on passe à l'état D .Sinon on passe à l'état E

**Exercice3**

Créer l’agent aspirateur vu en cours et simuler son comportement en utilisant les comportements composés