Sorbonne Université
M1 Androide

FoSyMa : Fondements des Systèmes Multi-Agents
2024-2025, 2^{eme} semestre

TP n°2

Projet Dedale - Exploration de l'Environnement

Dans sa configuration actuelle, les librairies exploitées par le projet sont *Jade* (plateforme agent), *Junit* (tests-unitaires), *GraphStream* [1] (graphes dynamiques) et *Dedale* (environnement du projet). La librairie *Dedale* sera progressivement mise à jour durant le semestre, à mesure que de nouvelles fonctionnalités vous seront nécessaires.

Comme indiqué dans le sujet du projet, l'environnement dans lequel vos agents vont évoluer est un graphe non-orienté. Les nœuds du graphe représentent les pièces, et les arrêtes les couloirs qui les lient. Deux agents ne peuvent être simultanément sur un même nœud.

1 Démarrage de la plateforme

1.1 Installation

— **Sur les machines de l'université**

1. Lancez eclipse depuis un terminal via la commande `eclipse-2023 &` puis connectez-vous à votre workspace (ou créez-le).
2. Dans eclipse, faites `Windows/Preferences/Maven/User Settings` puis dans le champ `Global settings` entrez : `/usr/share/maven/conf/settings.xml` Validez.
3. Importez le projet Dedale dans Eclipse
   (a) `File/Import/Git/Projects from Git/ Clone URI/`
   (b) Collez l'adresse ci-après dans le champ 'URI' : `https://gitlab.com/dedale/dedale-etu.git`
   (c) `Next/Next/Next/Next/Finish`
   (d) Votre programme va maintenant prendre un certain temps pour télecharger les dépendances.
   (e) Sélectionnez Dedale-etu dans le *project explorer* puis faites `Bouton-droit/Run As/Maven Install`
   (f) **Optionnel** si du rouge reste dans votre projet, faites `Bouton-droit/Run As/Maven Clean` puis `Bouton-droit/Maven/Maven Update` et enfin `Bouton-droit/Run As/Maven Install`.

— **Sur votre machine personnelle**
Suivez les consignes indiquées ici : `https://dedale.gitlab.io/page/tutorial/install/`

1. http ://graphstream-project.org/

1.2 Exécution

1. Exécutez la classe *Principal.java*. La configuration courante démarre un unique agent *DummyMovingAgent* sur un environnement prédéfini. Il est rattaché à l'environnement de manière transparente, puis doté de deux comportements :
   — `RandomWalkBehaviour` : Permet à l'agent d'évoluer dans – et d'interagir avec – l'environnement en se déplaçant de manière aléatoire.
   — `SayHello` : Tente de contacter un autre agent pour lui transmettre sa position courante. La distance entre l'émetteur et le récepteur conditionne la réussite de l'acheminement.

   Cet agent n'avance que lorsque vous appuyez sur `Entrée` dans la console Eclipse afin de vous permettre de suivre ce qui ce passe.
2. Après avoir lu `https://dedale.gitlab.io/page/tutorial/configureenv/` et plus particulièrement sa section 3, modifiez la configuration dans le fichier *ConfigurationFile.java* de façon à ce que l'environnement soit généré aléatoirement, de taille 10, et de type grille. Relancez la plateforme.

2 Principales caractéristiques d'un agent sur Dedale

2.1 Creation d'un agent et interaction avec l'environnement

Comme la semaine passée, la création d'un agent est déclenchée dans la classe *Principal.java*, au sein de la méthode `createAgents()` par le passage en parametre de la classe de l'agent souhaité.

```
//4) Give the class name of your agent to let the system instantiate it
ag=createNewDedaleAgent(c, agentName, DummyMovingAgent.class.getName(), entityParameters2);
```

`entityParameters2` correspond aux éventuels parametres que vous souhaiteriez donner à votre agent lors de sa création. Dans le cas présent aucun n'est nécessaire, la chaine de caractère est transmise a titre d'exemple.

Le code de l'agent *DummyMovingAgent* est présenté sur la page suivante :

```
public class DummyMovingAgent extends AbstractDedaleAgent{

/**
* This method is automatically called when "agent".start() is executed.
* It considers that the agent is launched for the first time and :
* 1) set the agent attributes , 2) add the behaviours
*/
protected void setup(){
super.setup();

//get the parameters given into the object[]
//use them as parameters for your behaviours is you want
final Object[] args = getArguments();

List<Behaviour> lb=new ArrayList<Behaviour>();

/************************************************
* ADD the behaviours of the Dummy Moving Agent
************************************************/
lb.add(new RandomWalkBehaviour(this));
lb.add(new SayHello(this));

/***
* MANDATORY TO ALLOW YOUR AGENT TO BE DEPLOYED CORRECTLY
*/

addBehaviour(new startMyBehaviours(this,lb));
}
```

A l'instar du *DummyMovingAgent*, vos agents devront étendre *AbstractDedaleAgent* et respecter la structure de ce setup() en ajoutant leurs comportements initiaux au sein de la liste `lb`.

`AbstractDedaleAgent` fourni à vos agents les méthodes d'intéraction avec les autres agents et l'environnement.
`StartMyBehaviours` prend en charge les problèmes de distribution que nous verrons plus tard dans le semestre et s'assure que vos comportements ne démarrent qu'une fois l'agent effectivement déployé sur la carte.

Lisez le code du comportement `RandomWalkBehaviour` de l'agent afin de vous familiariser avec l'utilisation de l'API présentée dans la sous-section ci-après.

Conseil sous Eclipse : Il vous suffit de selectionner le nom d'une classe, d'une méthode ou d'une variable puis d'appuyer sur `F3` pour vous rendre automatiquement à l'endroit où est déclarée la classe/méthode/variable considérée.

2.2 API de l'environnement

Voici les principales méthodes de l'API :

- ***String getCurrentPosition()*** Retourne la position courante de l'agent
- ***List<Couple<String,List<Attribute>>> observe()***
  Retourne l'ensemble des observables depuis la position courante de l'agent sous la forme d'une liste de couple (position, liste (attribut/valeur))
- ***boolean moveTo(String myDestination)***
  Permet à votre agent de se déplacer dans l'environnement jusqu'à la position fournie en paramètre (si atteignable). Afin de garantir la cohérence de votre agent et de vous faciliter le travail d'implémentation, cette fonction, lorsqu' elle est appelée, **doit etre la dernière méthode de votre behaviour**.
- ***void sendMessage(ACLMessage msg)***
  Fonction d'envoi de message qui gère le rayon de communication de agents. Vous devez utiliser **exclusivement** celle-ci.
- ***int getBackPackFreeSpace()***
  Retourne l'espace libre dans le sac à dos de l'agent.
- ***int pick()***
  Permet de récupérer tout ou partie du trésor présent sur la position courante (en fonction de la capacité d'emport de l'agent et du respect des conditions d'ouverture du coffre)
- ***String getMyTreasureType()***
  Permet à l'agent de connaitre son type
- ***boolean EmptyMyBackPack(String agentSiloName)***
  Permet à l'agent de vider son sac dans le silo, sous réserve qu'il soit à porté.

Les fonctions relatives aux trésors seront utilisées plus tard dans le semestre. La Javadoc – accessible sur le site de Dedale ainsi que depuis le code source – vous donnera le détail complet de l'ensemble des fonctions de l'API, et le code du *RandomWalkBehaviour* illustre l'utilisation de chacune d'elle.

3 Exploration et représentation de l'environnement

3.1 Exploration mono-agent

Dans la classe *Principal.java*, méthode `createAgents`, remplacez la création de l'agent `DummyMovingAgent` par `ExploreCoopAgent` puis exécutez le code.

```
//4) Give the class name of your agent to let the system instantiate it
348 //ag=createNewDedaleAgent(c,agentName,DummyMovingAgent.class.getName(),entityParameters2
349 ag=createNewDedaleAgent(c,agentName,ExploreCoopAgent.class.getName(),entityParameters2);
```

`ExploreCoopAgent` explore son environnement et construit sa propre représentation de celui-ci en vous donnant un aperçu temps réel de ses connaissances [2]. Lisez le code de l'agent, son behaviour et sa structure de représentation mémoire.

1. Quel type de parcours Elsa effectue-t'elle ? Pourquoi est-ce préférable à un autre type de parcours ?

2. Deux exemples simplistes de création et visualisation de graphe sont disponibles dans le package `test`

2. Quand celui-ci s'arrête t'il ?
3. Elsa ne s'intéresse qu'à la topologie. D'autres élements seront nécessaires lorsque votre agent devra collecter des ressources et détecter les éventuels adversaires. Réfléchissez à une représentation mémoire adaptée.

3.2 Exploration multi-agent

En vous inspirant (ou pas) de ExploreCoopAgent, proposez un mécanisme permettant à plusieurs agents d'explorer collectivement **et intelligemment** l'environnement. En particulier, réfléchissez à la façon la plus pertinente de représenter et partager l'information.

La solution que vous proposerez devra être en mesure de fonctionner quel que soit le nombre d'agents de votre équipe, leur(s) rayon(s) de communication (potentiellement différents), et la topologie sur laquelle ils évoluent.

Réfléchissez consciencieusement aux différentes situations que vos agents sont susceptibles de rencontrer, dessinez le protocole de communication envisagé et identifiez les rôles associés. Analysez les différentes façon de décomposer ceux-ci en behaviours en tenant compte de l'impact des différentes solutions sur l'information à représenter et partager entre les behaviours d'un agent, ainsi qu'entre les agents eux-mêmes (complexité en temps, mémoire, nombre de messages, taille des messages,..).
On rappelle que le contenu d'un message entre agent n'est pas limité à une chaine de caractère. L'utilisation de la méthode `msg.setContentObject(Object o)` vous permet de transmettre tout objet **sérializable**.

Bonne chance à vos agents !

Déploiement de plusieurs agents sur la carte

1. Lisez `https://dedale.gitlab.io/page/tutorial/deployagents/`.
2. Dans la classe *Principal.java*, `createAgents()`, décommentez l'agent Tim.
3. Ouvrez le fichier *ressources/agentExplo* et créez en une copie. Modifier celle-ci de façon à autoriser la création de Tim en plus de Elza puis pointez dessus dans *ConfigurationFile.java*
4. Lancez la plateforme. Si vous avez fait ce qu'il faut, vos deux agents se promènent sur la carte.

Pensez a tester vos solutions (exploration, interblocage, collecte,..) sur différentes topologies et avec différents nombres d'agents pour vous assurer un minimum de robustesse. Cela vous évitera de potentielles mauvaises surprises lors de l'évaluation finale..

Aide, remarques et discussions sur le projet

— Installation, configuration : `dedale.gitlab.io`
— Pour toute question, Discord : `https://discord.gg/JZVz6sR`
— Pour tout bug ou demande de nouvelle fonctionnalité, Discord ou : `https://gitlab.com/dedale/dedale/-/issues/new`