Escola Tècnica Superior d'Enginyeria

Universitat Rovira i Virgili

[DEIM]

Multi-Agent Systems

Jordi Pascual – jordi.pascual@urv.cat

Introduction to Dedale

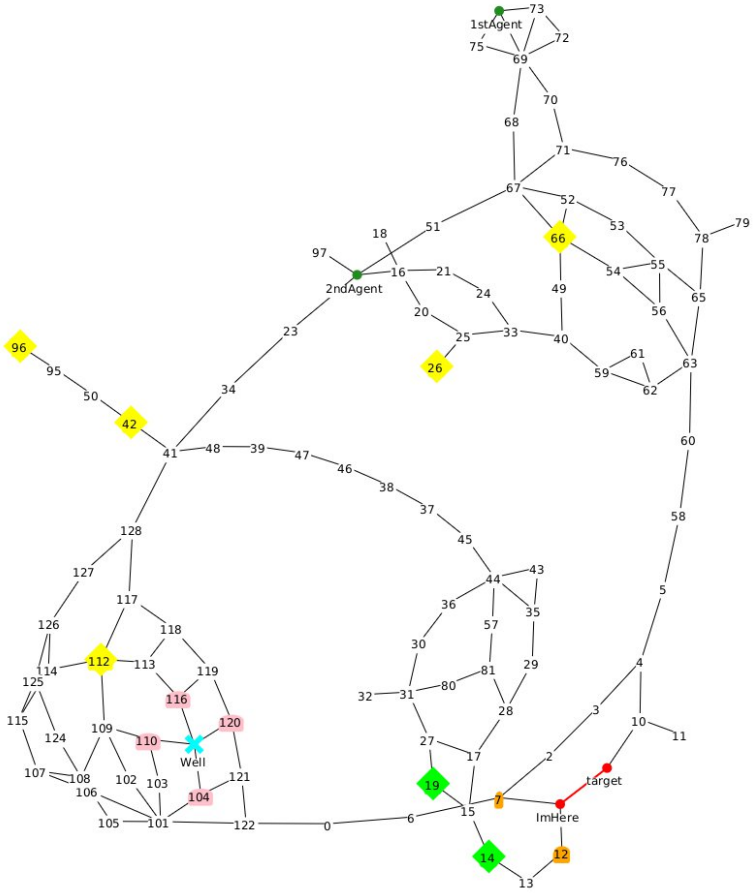
MESIIA – Master's Degree in Computer Security Engineering and Artificial Intelligence
MAI - Master's Degree in Artificial Intelligence

Outline

1. Introduction to Dedale
2. Install and execute Dedale
3. Environment
4. Agents
5. Exercise
6. Practical work settings
7. More resources

1. Introduction to Dedale

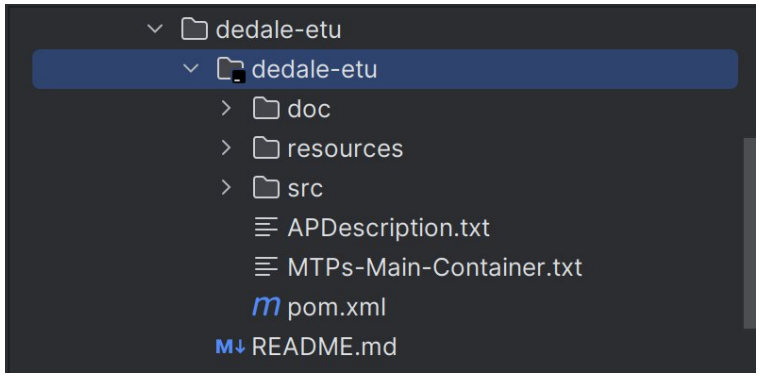
- JADE-based environment for studying multi-agents systems
- Initially unknown environment has to be explored by the agents
- Treasure hunt problem
- Agents coordination is needed to find all the treasures in the least time possible

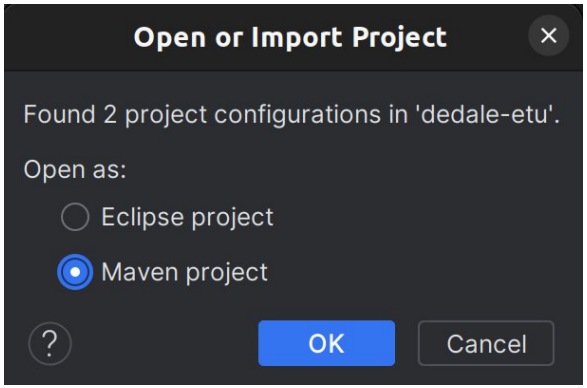


2. Install and execute Dedale

Follow the next steps to install and execute Dedale

1. Clone the Git repository https://gitlab.com/dedale/dedale-etu
2. Open the **dedale-etu** project using IntelliJ. **Note**: open the inner dedale-etu folder
3. Import the project as a **Maven project**
4. If prompted, **trust the project**





2. Install and execute Dedale

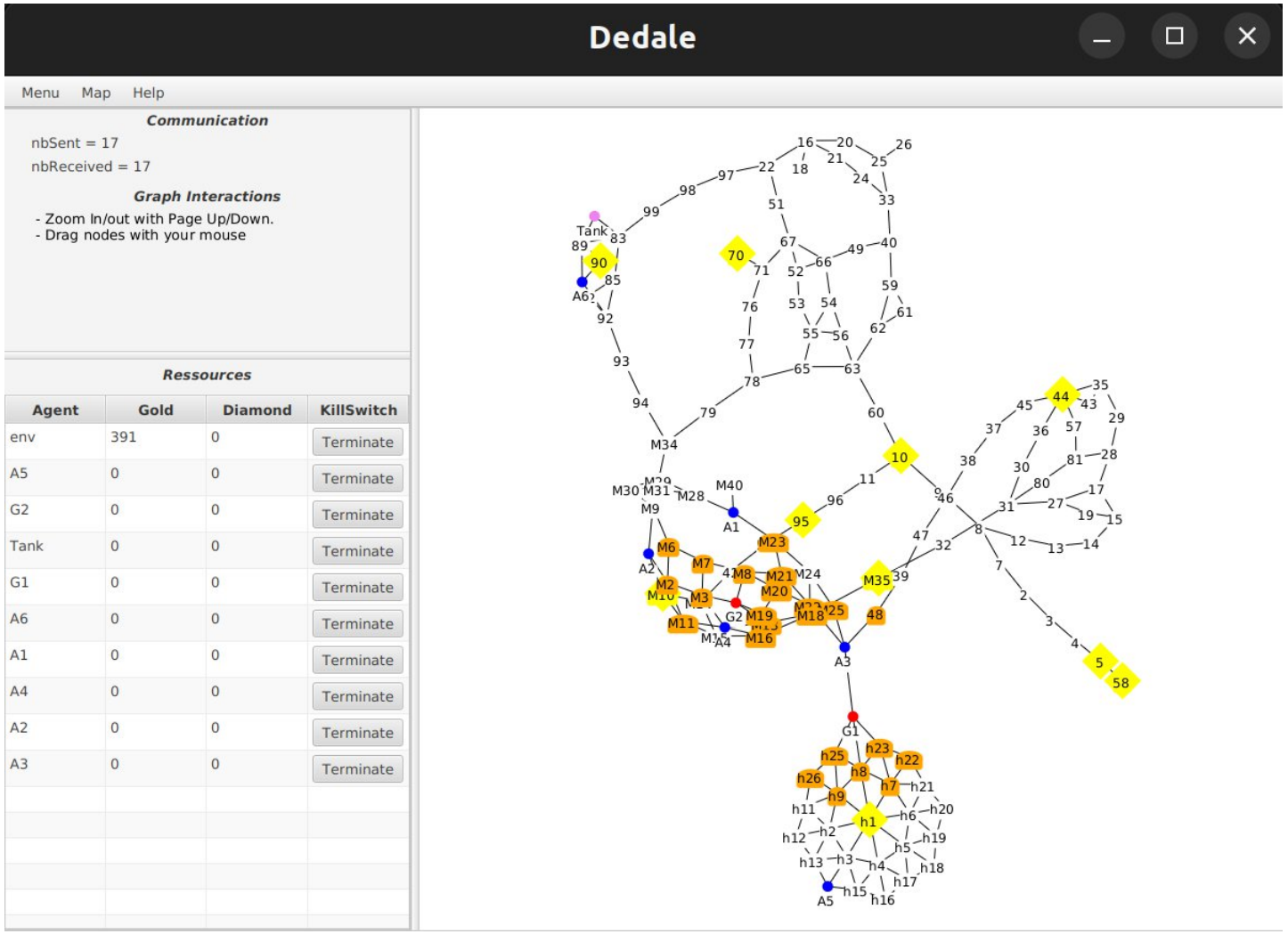
5. Add a new plugin at the **<build>/<plugins>** section of the **POM** file to execute by default the **Principal** class. No Maven profiles are needed with Dedale

```
<plugins>
        ...
        <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                        <mainClass>eu.su.mas.dedaleEtu.princ.Principal</mainClass>
                </configuration>
        </plugin>
</plugins>
```

2. Install and execute Dedale

6. Build and execute Dedale using Maven **mvn install exec:java**. If no agents show, cleaning the project (**mvn clean**) might solve the issues
7. By default, the **JADE** and **Dedale GUI** will open. A **sniffer** and several types of agents will also be created

2. Install and execute Dedale



3. Environment

- **Undirected graph**
- Nodes are the **locations** of the map
- Each node has a **unique identifier**
- Nodes can have certain objects
- Edges are the **paths** on the map
- Agents can move through nodes using the available edges
- Maps can be automatically generated using one of the available generators or importing them from OpenStreetMap

3. Environment: Map topology

- All maps from the *resources* folder can be used
- They follow the DGS File Format Specification
- OpenStreetMap topologies can also be transformed to **DGS** using gs-geography
- Maps can also be auto-generated using one of the available map generators

3. Environment: Load DGS map

- Configuration in the **ConfigurationFile** class
- Environment type -> Graph (line 51)

```
public static EnvironmentType ENVIRONMENT_TYPE=EnvironmentType.GS;
```

- Generator type -> Manual (line 56)

```
public static GeneratorType GENERATOR_TYPE=GeneratorType.MANUAL;
```

- Instance topology -> Path to DGS map (line 83)

```
public static String INSTANCE_TOPOLOGY="resources/topology/map.dgs";
```

3. Environment: Auto-generate map

- Configuration in the **ConfigurationFile** class
- Environment type -> Graph (line 51)

```
public static EnvironmentType ENVIRONMENT_TYPE=EnvironmentType.GS;
```

- Generator type -> GRID/BARABASI/DOROGOVTSEV (line 56)

```
public static GeneratorType GENERATOR_TYPE=GeneratorType.MANUAL;
```

- Set required generator parameters (lines 108-143)

3. Environment: Elements

- Elements can be manually set on a **DGS** map
- Three elements available: **gold** / **diamonds** / well
- Required parameters
  - type:position(nodeID):amount:strength:lock-picking
- Wells only require the position
- Elements saved in a file and configured in the **ConfigurationFile** (line 105)

```
public static String INSTANCE_CONFIGURATION_ELEMENTS="resources/map-elements";
```

```
mapname:name
gold:82:95:0:0
gold:96:24:1:2
gold:42:60:1:1
gold:66:46:2:0
gold:112:10:2:3
diamonds:14:24:0:0
diamonds:19:41:1:0
well:111
```

4. Agents

- Three steps to add agents to the environment
  1. Create the Java agents
  2. Configure the agents characteristics
  3. Deploy the agents

4. Agents: Create the agents

- They must extend **AbstractDedaleAgent**
- Use startMyBehaviours to add the Behaviours

```
public class DummyAgent extends AbstractDedaleAgent {

	@Override
	protected void setup(){
		super.setup();

		List<Behaviour> lb = new ArrayList<Behaviour>();
		lb.add(new FirstBehaviour(this));
		lb.add(new SecondBehaviour(this));

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

4. Agents: Interaction

Agents can interact with the environment using the Dedale API

- observe(): returns the set of observables that can be perceived from the agent current position as a list of couple (*position*, *List(ObservationType*, *Value)*)
- moveTo(Location node): makes your agent move to nodeID (if reachable). Must be the last function called within your behaviour
- sendMessage(ACLMessage msg): send a message and manage the communication radius. You must use only this method when communicating
- getMyTreasureType(): type of treasure the agent can grab (only one type per agent)
- openLock(Observation o): open the safe (Gold or Diamond) if the required expertise is provided. This method aggregates all the expertises of the agents surrounding the agent triggering this method
- pick(): grab all possible treasure available on the current position
- emptyMyBackPack(String tankerAgentName): transfer its backpack within the Tanker agent if it is in the vicinity

4. Agents: Configuration

- **JSON file** in the resources folder
- Parameters:
  - **agentType**: agentExplo / agentCollect / agentTanker
  - **agentName**: name of the agent. Will be used later on
  - **initialLocation**: *free* to randomly place the agent, or *nodeID*
  - **backPackCapacityGold/Diamond**: maximum gold/diamond the agent can carry. -1 if it cannot grab them
- Set it in the **ConfigurationFile** (line 150)

```
[
{
   "agentType":"agentExplo",
   "agentName": "Explo1",
   "communicationRange": 3,
   "initialLocation": "free",
   "backPackCapacityGold": -1,
   "backPackCapacityDiamond":-1,
   "detectionRadius": 0,
   "strengthExpertise": 1,
   "lockPickingExpertise":1
}
]
```

```
public static String INSTANCE_CONFIGURATION_ENTITIES="resources/agents.json";
```

4. Agents: Deploy

- Deployed in the **Principal** class
- Configured in method **createAgents()** (between lines 281-554)

1. Select the container where the agent will be deployed

```
c = containerList.get(ConfigurationFile.LOCAL_CONTAINER2_NAME);
Assert.assertNotNull("This container does not exist", c);
```

2. Set the agent name. Must exist on the configuration file

```
agentName = "Explo1";
```

3. Set the agent parameters if required

```
Object [] entityParametersExplo1 = { "My parameters" };
```

4. Create the agent

```
ag = createNewDedaleAgent(c, agentName, AgentClass.class.getName(), entityParametersExplo1);
```

5. Add it to the agents list

```
agentList.add(ag);
```

5. Exercise

In this exercise, you will have to add some agents to Dedale. Follow the next steps:

1. Configure Dedale to use the **URV.dgs** map available on the URV Virtual Campus
2. Create a **new elements file** with a couple treasures. Define different attributes for each of them
3. Create a **new configuration JSON** file for 2 explorer agents, and 1 collector agent. Define different attributes for each of them. Spawn them in random (*free*) nodes
4. Modify the **Principal** class to create the configured agents. The agent class for explorers has to be **ExploreCoopAgent**, and for collectors **DummyCollectorAgent**

Reminder: run Dedale with **mvn install exec:java**

5. Exercise: Solution

1. Download **URV.dgs** from the URV Virtual Campus into the resources folder
2. Set the **ConfigurationFile** to use the URV map

```
public static String INSTANCE_TOPOLOGY="resources/URV.dgs";
```

3. Create a new file (e.g. URVelements) in the resources folder and add some elements

```
mapname:URV
gold:-139586:100:0:0
gold:-139601:100:0:0
diamonds:-139316:50:0:0
diamonds:-138849:50:0:0
```

4. Set the **ConfigurationFile** to use those elements

```
public static String INSTANCE_CONFIGURATION_ELEMENTS="resources/URVelements";
```

5. Exercise: Solution

5. Create **agents.json** file in the resources folder with the agents configuration and set **ConfigurationFile** to use it

```
[
{
"agentType":"agentExplo",
"agentName": "Explo1",
"communicationRange": 3,
 "initialLocation": "free",
"backPackCapacityGold": -1,
"backPackCapacityDiamond":-1,
"detectionRadius": 0,
"strengthExpertise": 1,
"lockPickingExpertise":1
},
{
"agentType":"agentExplo",
"agentName": "Explo2",
"communicationRange": 3,
"initialLocation": "free",
"backPackCapacityGold": -1,
"backPackCapacityDiamond":-1,
"detectionRadius": 0,
"strengthExpertise": 1,
"lockPickingExpertise":1
},
{
"agentType":"agentCollect",
"agentName": "Collect1",
"communicationRange": 3,
"initialLocation": "free",
"backPackCapacityGold": 500,
"backPackCapacityDiamond":500,
"detectionRadius": 0,
"strengthExpertise": 3,
"lockPickingExpertise":3
}
]
```

```
public static String INSTANCE_CONFIGURATION_ENTITIES="resources/agents.json";
```

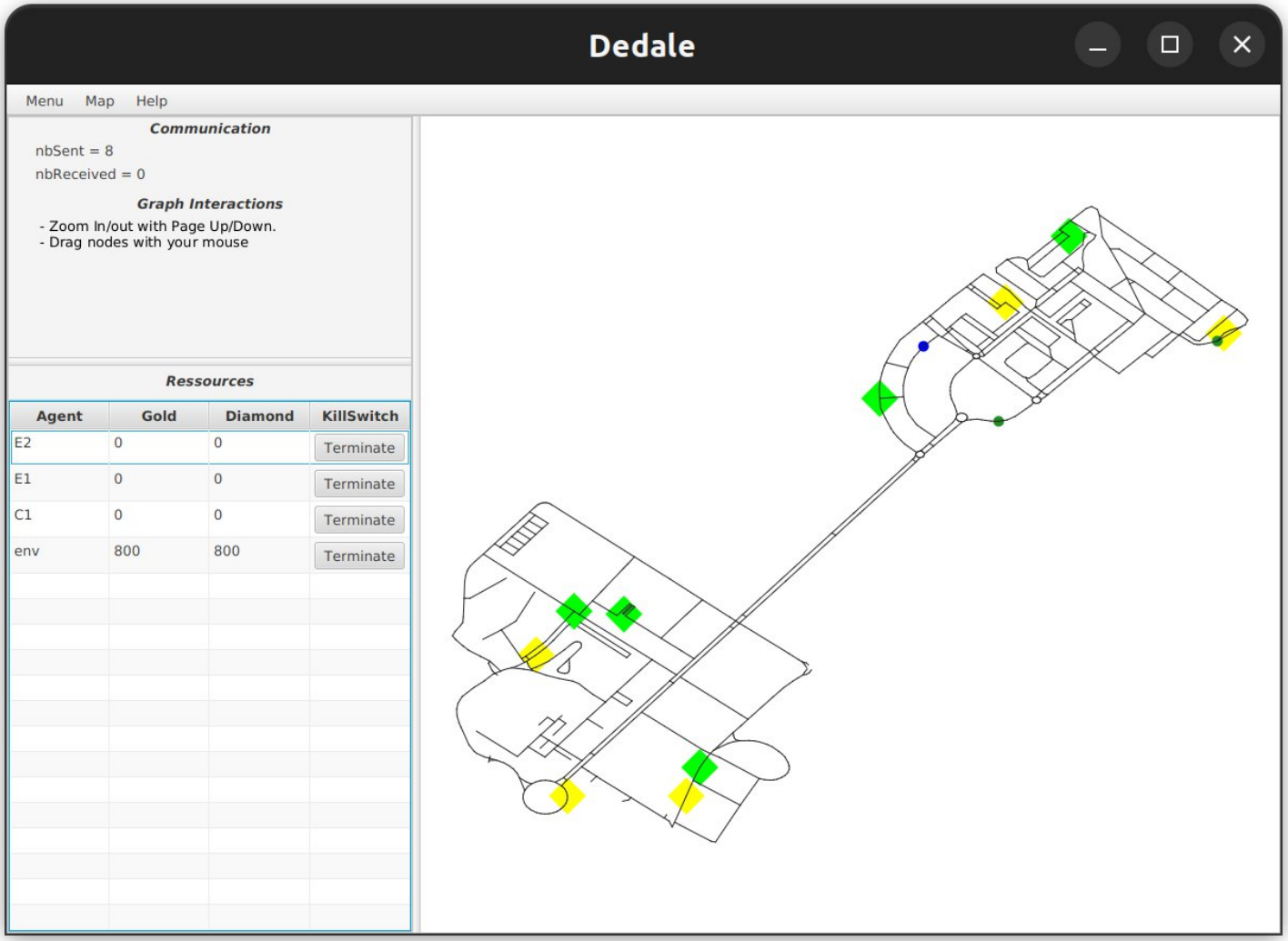
5. Exercise: Solution

6. Deploy the agents on the **Principal** class

```
c = containerList.get(ConfigurationFile.LOCAL_CONTAINER2_NAME);
Assert.assertNotNull("This container does not exist", c);
Object [] entityParameters = { "My parameters" };

agentName = "Explo1";
ag = createNewDedaleAgent(c, agentName, ExploreCoopAgent.class.getName(), entityParameters);
agentList.add(ag);

agentName = "Explo2";
ag = createNewDedaleAgent(c, agentName, ExploreCoopAgent.class.getName(), entityParameters);
agentList.add(ag);

agentName = "Collect1";
ag = createNewDedaleAgent(c, agentName, DummyCollectorAgent.class.getName(), entityParameters);
agentList.add(ag);
```

7. Build and execute Dedale using **mvn install exec:java**

5. Exercise: Solution



6. Practical work settings

By default, part of the remaining treasure is lost when grabbing part of it. We are not going to use this function for the practical work, hence, it must be disabled in the **main** function of the **Principal** class

```
InGameConfigurationFile.PERCENTAGE_TREASURE_LOSS = 0d;
```

Having the MapRepresentation GUIs open all the time, might be computationally expensive and might freeze Dedale. You can comment lines **72-74** on the **MapRepresentation** class to not have it open by default, and use the **OpenGui** and **CloseGui** methods provided by the **MapRepresentation** class to open and close them at your will

7. More resources

- FIPA Protocols
- JADE Javadoc
- JADE Guides
- JADE Maven Setup for Beginners
- Dedale Webpage
- Dedale API
- Dedale GitLab