Intelligent Systems Programming

Academic year 2023-2024

Version 1.0

1 Objectives

The goal of this lab project is to exercise the basics of agent-oriented programming, in order to help understand the concepts taught in the theoretical classes.

The students have to develop a program that runs a tournament among N players, which may have different implementations. Players must follow rules described later in this document, and each student must also provide one or several intelligent players of his/her own, implementing more sophisticated strategies.

The developed players must be able to interoperate with players created by other students. Gathering all the players at the end of the course, a global competition will be played that will determine the best algorithms.

2 The Hawk-Dove Tournament

We shall play a *Hawk-Dove tournament*, where $N \geq 2$ players will play an one-on-one iterative version. The rules for this game will be described on the theory classes, but basically, we have a symmetric matrix with the next payoffs:

A1/A2	H	D
H	-1,-1	10,0
D	0,10	5,5

Table 1: Classical PD game payoffs.

where agent A1 selects an action (H or D) for the rows, agent A2 selects an action (H or D) for the columns, and then, after selecting both actions, they obtain the round payoff (1st value for A1 and 2nd value for A2). For instance: after selecting the actions H by A1 and D by A2, then A1 gets 10 points and A2 obtains 0 points.

Each two agents will play a number of rounds between both, that we denote as a game. Then, a tournament is the set of games between all the possible pairs of N players.

3 The main agent and the multi-agent system

Together with the players (agents), we also consider the existence of a main agent that behaves as a hub, receiving the adequate actions selected by each agent, and providing them the outcomes along the different rounds of the game. Besides, the main agent is in charge of providing a graphical user interface (GUI) to report the present state of the game, including: the number of rounds, the parameters selected, the outcomes per round, the agents participating, their score state, and global information.

Thus, the main agent must keep track of the players' scores and ranking, and it also will be in charge of orchestrating the competition among the player agents. To begin with, it will discover them using the services provided by the *JAVA Agent DEvelopment Framework* (JADE). Then, communication among the agents will be done by using the *Agent Communication Language* (ACL) formats defined by JADE.

In a certain round, when a player has reported its action to the main agent, the latter sends back the corresponding actions selected by the players, and the payoff obtained. Players receive such information, and must update their own statistics accordingly.

Specifically, the main agent will communicate with the agents as follows:

1. Each player will have a unique identifier and will get information about the competition. These are provided by the main agent using a message with performative INFORM and the text `Id`, the symbol `#`, an integer (the player identifier)[1], the symbol `#` and the list of parameters (N, R), where:

   - N: the total number of players.
   - R: the number of rounds in each game between two players.

   A valid example, considering the values by defect, is: `Id#2#25,100`.

2. At the beginning of each one-on-one game, the main agent informs, the two agents that are going to play, by sending a message with performative INFORM with the text `NewGame` and the identifiers. For example: `NewGame#4#7`.

3. After starting a game, the main agent asks the players for their action (H or D) each one will play in that round. The main agent will use a message with performative REQUEST and the text `Action`. Each player will respond with a message with performative INFORM and the text `Action`, the symbol `#` and the action chosen (H or D). A valid response would be, for example: `Action#D`.

4. Next, the main agent reports the contributions to the players, by sending a message with performative INFORM and the text `Results`, the symbol `#` the identifiers, the symbol `#`, the actions, the symbol `#` and finally the payoffs in an increasing order of identifiers[2]. For example: `Results#4,7#D,H#0,10`.

5. In the end of game, when R rounds have been played between the two players, the main agent informs the players with a message with performative INFORM and the

[1] The id of the first player is 0.

[2] This information is relatively redundant, but can help to debug the code.

text `GameOver`, followed by the symbol `#`, then the identifiers, afterwards the symbol `#`, and the *total payoffs* obtained after R rounds by each player. For example: `GameOver#4,7#267,233`.

The main agent must keep track of the agents' score along the different phases, so it knows who wins each game, and also what scores are accumulated during the whole tournament, i.e., determining the winner.

4 Visual interface

The program must provide the next minimum functionalities, considering that every student can include extra ones:

- **Edit**:
  - Reset players: resets the statistics of all players.
  - Remove player: from the game.
- **Run**:
  - New: starts a new tournament, as the full series of games.
  - Stop: stops the execution of the current game.
  - Continue: continues the execution if it was stopped.
  - Number of rounds: sets maximum number of rounds R to play in a game.
- **Window**:
  - Verbose on/off: enable or disable console comments along the matches.
- **Help**:
  - About: data about the program author.

In addition, the main window should display and update:

- The number of players participating.
- Information about the parameter values.
- The number of games already played.
- Names and statistics of players involved.
- An area (console) where game information appears when it is being played.

5 What to develop and deliver

Each student must provide the following:

- An interoperable implementation of the **main agent**.
- An interoperable implementation of several **player agents** to play games automatically and must be located in an **agents package**. This package must include:
  - A *random agent*, that chooses the action randomly.
  - A reinforcement learning agent (`RL_Agent.java`).
  - A neural network agent (`NN_Agent.java`).
  - The agent that will be submitted to the final tournament (it can be any of the two previous ones or a different one).
- A **graphical interface** for a human user to set up and launch a league. The main window should display useful information to monitor the current game and the overall competition.

See next an example about the syntax to execute JADE with the MainAgent and three agents from the prompt line (case sensitive):

```
java -classpath "./jade.jar:./" jade.Boot -agents
"MainAgent:MainAgent;intel02:PSI_02;intel20:PSI_20;intel25:PSI_25"
```

The code of the **MainAgent** will be named `MainAgent.java`, and located in the root folder, while the code of the players will be named `RandomAgent.java`, `RL_Agent.java`, `NN_Agent.java` and `PSI_x.java` (the one to be used for the tournament[3]), and be located in a folder/package named `agents`. If any agent needs any extra file, it must be named `EXTRA_x.dat`, located also in the agents folder and must be smaller than 1 MB.

Each student must deliver the implementation files plus a summary of less than 3 pages in a file called `readme.txt`, containing at least the following information:

1. Name and account number of the student.
2. A precise description of the steps and the command lines needed to compile and run its practice.
3. A motivated description of the algorithms implemented in each of the intelligent players (RL, NN, etc.).
4. A clear description of the agent algorithm to be used in the final tournament.
5. Descriptions of any extra functionalities and/or any final comments (optional).

[3] being `x` the lab account number of the student.

6 Evaluation

The evaluation of the practice will be done in two phases:

1. The 1st phase deals with the implementation of all the requirements specified for the game, the GUI, the MainAgent, the random agent and the communication (4 points):
   - Correct implementation of those requirements: up to 2.5 points.
   - Source code quality (including comments/documentation): up to 0.5 points.
   - GUI quality and extra functionalities: up to 1 point.
2. The 2nd phase deals with the implementation of the intelligent agents (6 points) and requires submitting:
   - A RL_Agent: up to 2.5 points (required).
   - A NN_Agent: up to 2.5 points (required).
   - A mix of the two previous ones (RL_NN_Agent): up to 1 point (optional).

Besides, there are some extra points that can be obtained by any student:

- A GUI+MainAgent, from the whole set delivered by the students, will be selected to play the final tournament; depending on its graphical quality, robustness and correctness. The author of the selected code will get **1 extra point**.
- The three intelligent agents that achieve the best results on the final tournament will obtain **1, 0.75 and 0.5 points, respectively**.

7 Delivery dates

There are three deadlines for the material requested:

1. The code containing the GUI and the MainAgent, implementing the game, together with the random agent should be uploaded no later than **November 17th, Friday**.
2. The final GUI/MainAgent together with game agents (`RL_Agent.java`, `NN_Agent.java`) should be uploaded no later than **December 22th, Friday**.
3. The tournament agent `PSI_x.java` should be uploaded up to **January 12th, Friday**.

The tournament will be performed in January together with the Group C presentations.