Intelligent Systems Programming

Academic year 2022-2023

Version 1.1

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. Two types of players will follow the rules described later in this document, while 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 CRD game

We shall play the public good game known as the *Collective-Risk Dilemma (CRD)*. In this game, $N \geq 2$ players are each given an endowment ($E = 40$ units), and they must decide whether to contribute to the common good over a certain number of rounds ($R = 10$). Therefore, in each round, each player P_i chooses an action/contribution ($a_i \in \{0, 1, 2, 3, 4\}$); without knowing what action the other players have chosen. Then, the contributions are added to the common good, and the players are informed about what the others have contributed, until the last round is reached.

At that point, there are two possible outcomes for the game:

- On the one hand, if the joint contributions of all the participants over those rounds are equal or above a certain threshold ($T = \frac{E*N}{2}$), then a disaster is averted, and each one receives as a reward the remainder of its individual endowment (hence the dilemma).
- On the other hand, if the threshold T is not reached, there is a probability that a disaster may occur ($P_d = 0.8$), resulting in economical loss for all the participants (they lose the remainder of their endowment).

In our case, we will play a global tournament in two phases: first one to warm up, then a battle royale, and then the final among the last survivors. These phases are described next:

1. **Battle Royal**: This phase will use sequences of (Gen=250) generations, each generation playing (NumGames=100) composed by (R=10) rounds each. Each generation is composed by 5 players chosen randomly among the whole population of players (choosing first among those ones that have played less times).

   At this stage, points will be added to each personal account, and the average will be performed per each generation, to consider cases where there is a difference in the number of times played.

   After those 250 generations, the last player(s) will be erased from the list of participants; and this process is repeated until only 5 players remain.

2. **Final**: The last 5 players will reset their scores, and then play (NumGames=1000, again with R = 10) to get their final score.

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 personal accounts state, and the common global account state. The GUI should also provide means to change the parameters provided by defect.

Thus, the main agent must keep track of the players' scores and ranking, and, besides, it will be in charge of orchestrating 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, once all players have reported their selections to the main agent, the latter sends back the corresponding actions selected by the players. All players receive such information, and must update their personal and global accounts correspondingly.

Specifically, the main agent communicates on each game with any player agent as follows:

1. Each player must have a unique identifier and information about the competition. These are provided by the main agent to each player 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, E, R, P_d, $NumGames$), where:

   - N: the total number of players.
   - E: the initial endowment provided to each player.

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

- R: the average number of rounds in each game.
- P_d: the probability of a disaster when the game ends, and the threshold has not been achieved.
- $NumGames$: the number of runs the collective game is played.

A valid example, considering the values by defect, is: `Id#2#5,40,10,0.8,100`.

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

3. After starting a game, the main agent asks the players for the contribution they choose 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 (a value from 0 to 4). A valid response would be, for example: `Action#2`.

4. Next, the main agent reports the contributions to the players, by sending a message with performative INFORM and the text `Results`, the symbol `#` and the payoffs in an increasing order of identifiers. For example: `Results#1,0,4,3,1`.

5. In the end of the game, the main agent informs the players with a message with performative INFORM and the text `GameOver`.

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 play-off, i.e., determining the play-off finalist and eventually the top 3 players.

4 Visual interface

The program must provide the next minimum functionalities, also 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 series of games.
  - Stop: stops the execution of the current game.
  - Continue: continues the execution if it was stopped.
  - Number of games: sets the number of games to play.
  - Change $Param$: allows to modify the parameter $Param$.
- **Window**:

  - Verbose on/off: enable or disable 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 where game information appears when it is being played.

5 What to develop and deliver

Each student must provide the following software:

- An interoperable implementation of the **main agent**.
- The interoperable implementation of several **player agents** to play games automatically and must be located in an **agents package**. This package includes:
  - A *random agent*, that chooses the action randomly.
  - A reinforcement learning (RL) agent implementing a RL method.
  - A neural network (NN) agent implementing a neural network approach.
  - 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. A *verbose mode* can be activated to show detailed information about each round of the current game, or deactivated to show only the accumulated contributions.

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

```
java -classpath "./jade.jar:./:./agents" jade.Boot -agents
"Referee:MainAgent;player02:PSI02;player20:PSI20;player40:PSI40"
```

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 `PSIx.java` (the one to be used for the tournament[2]), and be located in a folder/package named `agents`. If any agent needs any extra file, it must be named `EXTRAx.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 2000 words in a file called `readme.txt`, containing at least the following information:

- Name and account number of the student.
- A precise description of the steps needed to compile and run its practice.
- A motivated description of the algorithms implemented in each of the intelligent players (RL, NN, etc.).
- A clear description of the algorithm to be used in the final tournament (`PSIx.java`).
- Descriptions of any extra functionalities or final comments.

The aforementioned material must be zipped and uploaded to the Moovi platform in the next two deadlines: **October 28th, 2022** (main agent + random agent) and **December 16th, 2022** (the whole code, including the intelligent agents).

6 Evaluation

The correct implementation of all the requirements specified in this document implies that the student gets 4 points in this practice. The remaining 6 points will be distributed as follows:

- Quality of the source code (including comments/documentation): up to 1 point.
- Presentation and extra functionalities in the MainAgent: up to 1 point.
- Strategies used by the intelligent players: up to 4 points.

A MainAgent, among the whole set delivered by the students, will be selected to play the final tournament; depending on its graphical quality, robustness and correctness. From such tournament some extra points will be delivered:

- The author of the selected MainAgent will get **1 extra point**.
- The three intelligent agents that achieve the best results in the tournament will get **1, 0.75 and 0.5 points, respectively**.

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

Useful resources

- Documentation, tutorials and examples of the JAVA Agent DEvelopment Framework, from `http://jade.tilab.com/`
- Chapter on *Graphical User Interfaces* from *Building Java Programs: A Back to Basics Approach*, 2nd edition, by S. Reges and M. Stepp. Available at `https://www.buildingjavaprograms.com/samples4.shtml`
- Slides and code on *Java Graphics & GUIs* and *GUI Event-Driven Programming* from the course *CSE 331: Software Design and Implementation* taught by Michael Ernst. Available at `https://github.com/ldfaiztt/CSE331/tree/master/Week%2009`