Intelligent Systems Programming

Academic year 2024-2025

Version 1.5

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 Steal-Split tournament and the stock market

We shall play a tournament in which $N \geq 2$ players will play iteratively an one-on-one iterative version of the Steal-Split game. The rules for this game will be described in the classes but, basically, we have a matrix with the next payoffs:

P1/P2	C	D
C	2,2	0,4
D	4,0	0,0

Table 1: Classical game payoffs.

We consider a tournament round when the games between all pairs of players have happened. The payoff earned by each player is accumulated over successive rounds, and will determine the winner of the tournament. However, we will assume that there is some inflation that, after each round, reduces the value of the accumulated payoffs held by the players. To overcome this, the players can invest part of their earnings in a stock index from which they can eventually recovered the money, paying a given commission fee.

The tournament will consist in a number of rounds (hundreds or thousands) and the winner will be the player who accumulates the highest payoff, taking into account the amounts invested and recovered from the stock index.

3 The main agent and the multi-agent system

In addition to the players, there will be a main agent that behaves as a hub, keeping track of the players' scores and ranking, and orchestrating the competition. To begin with, it will discover the players using the services provided by the *JAVA Agent DEvelopment Framework* (JADE). Then, it will listen to the actions selected by each agent and, in return, inform them of the actions selected by the other players, and the payoffs obtained. Upon receiving such information, the players must update their internal statistics to drive their future decisions.

Communication among the agents will take place via the *Agent Communication Language* (ACL) formats defined by JADE:

1. Each player has a unique identifier and gets information about the competition from the main agent, who sends a message with performative INFORM and the text `Id`, the symbol `#`, an integer (the player's identifier)[1], the symbol `#` and the list of parameters (N, R, F), where:

   - N is the total number of players ($N > 2$),
   - R is the number of rounds in the tournament ($R > 100$), and
   - F is the commission fee applied when selling stock units ($F \in [0, 1]$).

   A valid example, considering the default values, would be `Id#2#25,500,0.01` to inform player 2 that the competition will involve 25 players and 500 rounds, and that the stock market applies a 1% commission fee.

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

3. In each round of a game, the main agent asks the players for their chosen action (D or C), by means of a message with performative REQUEST and the text `Action`. Each player will reply with a message with performative INFORM and the text `Action`, the symbol `#` and the chosen action (D or C). A valid response would be, for example: `Action#D`.

   Next, the main agent reports the outcome of the round, by sending the players a message with performative INFORM and the text `Results`, the symbol `#`, their identifiers, the symbol `#`, their chosen actions, the symbol `#` and the payoffs[2]. For example, the message `Results#4,7#D,C#4,0` means that in a round between players 4 and 7, the former chose D and the latter chose C, so they get the payoffs of 4 and 0, respectively. The message must use ascending order of identifiers (i.e., the bits about player 4 precede those of player 7).

4. When a round is finished (i.e., all agents have played against one another), the main agent sends a message to each player with performative REQUEST and the text `RoundOver`, plus its identifier, the total payoff earned in that round, the payoff accumulated in its own account, the inflation rate currently applied to the accumulated

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

[2] This information is somewhat redundant, but it can help debug the code.

payoff, the number of stock assets owned by the player, and the current value of the stock index (which, multiplied by the number of assets yields the amount of money put by the player in the market). For example:

`RoundOver#4#50#224.55#0.05#33.45#7.35`

would inform player 4 that it has attained a total payoff of 50 units in the round, contributing to an accumulated current payoff of 224.55 (having applied the current inflation rate of 0.05); the agent also has 33.45 assets, valued at 7.35 each.

The player agents should use the information provided in the `RoundOver` message to predict future values of the stock index and the inflation rate, in order to make their own investment decisions. Each agent shall reply to the main agent with a message with performative INFORM and the text `Buy` or `Sell` depending on how much it wants to buy or sell. For example: `Buy#34.55` means the agent decides to invest an amout of 34.55 payoff units, whereas `Sell#30` means the agent wants to sell 30 stock assets. Note that the agent cannot buy or sell more than it has (in that case, the MainAgent will ignore the order).

Finally, the main agent responds with a message with performative INFORM and the text `Accounting`, followed by the updated accumulated payoff and the updated number of assets. For example, if the agent 4 has bought 34.55 payoff units, then the main agent sends `Accounting#4#190#38.15`.

5. When the tournament is over (i.e., all the rounds have been played), the main agent sends a message to each player with performative INFORM and the text `GameOver`, plus the agent's identifier and the total payoff obtained in the tournament. For example: `GameOver#4#2543.73`. In this example, the payoff invested on the stock index is recovered, and added to the agent's accumulated payoff after applying the corresponding fee.

The main agent must keep a ledger with the agents' payoffs along the different rounds, keeping track of the accumulated payoffs and assets.[3] Besides, after each round, the main agent applies the current inflation rate to the money not invested by each player and, when so asked to, sells the assets by applying the commission fee.

Inside the main agent's code, the evolution of the stock index and the inflation rate will be implemented in two methods:

- `public double getIndexValue (int round)`
- `public double getInflationRate (int round)`

Each student may use the mathematical function or series he/she deems better to compute the values returned by these methods. The teachers will provide the ones to be used in the final tournament.

[3] We shall always use two decimal digits for the values.

4 Graphical interface

The main agent will provide a graphical user interface to show the current state of the game, including the number of rounds, the parameters selected, the successive outcomes (with two levels of detail), the participating players and the state of their accounts.

The program must provide, at least, the following functionalities:

- Reset the statistics of all players.
- Remove a selected player.
- Start a new tournament.
- Stop the execution of the tournament.
- Resume the execution of the tournament (if it was stopped).
- Set the number of rounds R to play (default: 500).
- Set the commission fee of the stock market (default: 0.01).
- Enable or disable detailed information about the successive games and rounds.
- Display data about the author.

5 What to develop and deliver

Each student must provide the following:

- An interoperable implementation of the **main agent** in a file named `MainAgent.java`, located in the root folder.
- An interoperable implementation of several **player agents** to play games automatically, located in an `agents` package. This package must include:
  - An agent that chooses the action for each round randomly (`RandomAgent.java`).
  - A reinforcement learning agent (`RL_Agent.java`).
  - A neural network agent (`NN_Agent.java`).
  - The agent that will be submitted to the final tournament (`PSI_x.java`, `x` being the lab account number of the student). This can be any of the two previous ones or a different one.

  If any agent needs any extra file, it must be named `EXTRA_x.dat`, located also in the `agents` folder and smaller than 1 MB.
- A **graphical interface** for a human user to set up and run tournaments.
- A **summary** of less than 3 pages in a file called `readme.txt`, containing at least the following information:

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

Below is 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"
```

6 Evaluation

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

1. The first phase deals with the implementation of all the requirements specified for the game, the GUI, the MainAgent, the random agent and the communication protocol (4 points):
   - Correct implementation: 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 second phase deals with the implementation of the intelligent agents (6 points):
   - `RL_Agent`: up to 2.5 points (mandatory).
   - `NN_Agent`: up to 2.5 points (mandatory).
   - A mix of the above (`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 15th**.
2. The final GUI/MainAgent together with game agents (`RL_Agent.java`, `NN_Agent.java`) should be uploaded no later than **December 20th**.
3. The tournament agent `PSI_x.java` should be uploaded by **January 10th**.

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