Practical assignment: Automated Negotiation

Reyhan Aydoğan & Catholijn M. Jonker

March 14, 2017

The practical assignment aims at familiarizing you with parts of the course material in a more practical way. This assignment is about multilateral negotiation – a form of interaction in which there are more than two agents, with conflicting interests and a desire to cooperate, try to reach a mutually acceptable agreement. The list of tasks is summarized below and you can find the details in the following text:

- Designing and implementing a negotiating agent in GENIUS for multiparty negotiation - which is a multiagent system developed at the University of Delft, The Netherlands.
- Testing and comparing the performance of the agent with the ParsAgent, Atlas3 and RandomDance agents in the given negotiation scenarios.
- Preparing an analysis report with explanation of your solution

Typically, negotiation is viewed as a process divided into several phases. Initially, in a prenegotiation phase the negotiation domain and the issue structure related to the domain of negotiation are fixed. Negotiation may be about many things, ranging from quite personal issues such as deciding on a holiday destination to strictly business deals such as trading orange juice in international trade. In this assignment, the following negotiation scenarios will be examined:

- $partydomain/party1$ + $partydomain/party2$ + $partydomain/party3$
- $university/university7$ + $university/university8$ + $university/university9$
- $energy_grid_domain/consumer$ + $energy_grid_domain/provider$ + $energy_grid_domain/producer$

These preference profiles are represented by means of additive utility functions. For example, if there are four issues to negotiate about, the utility function can be computed by a weighted sum of the values associated with each of these issues. So, let $bid = \langle i_1, i_2, i_3, i_4 \rangle$ be a particular bid. Then the utility $u(bid) = u(i_1, i_2, i_3, i_4)$ (given weights w_1, w_2, w_3, w_4) can be calculated by: $u(i_1, i_2, i_3, i_4) = w_1 \cdot u(i_1) + w_2 \cdot u(i_2) + w_3 \cdot u(i_3) + w_4 \cdot u(i_4)$.

The most important part of this assignment concerns the *negotiation phase* itself i.e. thinking about a strategy used to perform the negotiation such as exchanging offers among your software agent and its opponents. It is worth noting that the negotiation we consider in this assignment is a closed multi-issue negotiation where there are multiple issues (i.e. more than one issue to be agreed on) and the negotiation parties only know their own preferences. It is not allowed to have access to other agents' preferences. The negotiation agent will follow "**Stacked Alternating Offers Protocol for Multi-Lateral Negotiation (SAOPMN)**" [1]. According to this protocol, the first agent will starts the negotiation with an offer that is observed by all others immediately. When an offer is made, the next party in the line can take the following actions:

- Make a counter offer (thus rejecting and overriding the previous offer)
- Accept the offer
- Walk away (e.g. ending the negotiation without any agreement)

This process is repeated in a turn taking fashion until reaching an agreement or reaching the deadline. To reach an agreement, all parties should accept the offer. If at the deadline no agreement has been reached, the negotiation fails. It is worth noticing that you will set the deadline before the negotiation starts (See Section 1.4).

The following block provides some initial guidelines based on human experience with negotiation that may help you during your own negotiations and in building your own negotiating software agent.

- Orient yourself towards a win-win approach.
- Plan and have a concrete strategy.
- Know your reservation value, i.e. determine which bids you will never accept.
- Create options for mutual gain.
- Take the preferences of your opponents into account.
- Generate a variety of possibilities before deciding what to do.
- Pay a lot of attention to the flow of negotiation.
- Examine the previous negotiation sessions to adapt to domain over time.

This assignment concerns building your own negotiating software agent. This will be a team effort: as a team **you will design and implement your negotiating agent in Java** to do negotiations for you. The agent strategies developed for bilateral negotiations can give some insights; for example, you can find time and behavior dependent negotiation strategies in [5]. We recommend to search for strategies used in the ANAC Competition [2, 3, 4, 6, 7, 8, 9, 12]. It is worth noting that the ANAC agent strategies have been designed for bilateral negotiations where the agent has only one opponent while in your assignment, your negotiating agent will negotiate with **two opponent agents**. Although you already know that your agent will negotiate with two agents, you are asked to **design and implement your agent in a generic way** so that it can negotiate with more than two agents too.

You will design and implement your negotiation agent in Genius, which is a negotiation environment that implements an open architecture for heterogeneous negotiating agents [10]. It provides a testbed for negotiating agents that includes a set of negotiation problems for benchmarking agents, a library of negotiation strategies, and analytical tools to evaluate an agent's performance. After implementing your agent, you will analyze the performance of this negotiating agent by comparing with the ParsAgent, Atlas3 and RandomDance agents in terms of average individual utility gained, optimality of the agreement, the duration of agreement (e.g. number of rounds to complete the negotiation), and so on. You will prepare a final analysis report with the explanation of your agent strategy.

The remainder of this document is organized as follows. In Section 1 the objectives, deliverables, requirements and assignment itself are described. Section 3 describes some organizational details and important dates, including deadlines. Finally, Section 4 documents the evaluation criteria and grading for this assignment.

1 Detailed Assignment Description

The assignment must be completed in teams of 3 students. In the following paragraphs the objectives, deliverables, requirements, and the detailed assignment description are documented

1.1 Objectives

- To learn to design a negotiating agent for a realistic domain with discrete issues, including among others a negotiation strategy.
- To learn techniques for implementing (adversarial) search and design heuristics while taking into account time constraints.
- To actively interact with other students and participate in student groups by discussing and coordinating the design and construction of a negotiating agent.

1.2 Deliverables

This project consist of four stages:

- **Initial stage** - You should register your group via e-mail (umut.ulutas@ozu.edu.tr) until 20.03.2017. After the registration, a unique number **n** to identify the team and the negotiating agent will be provided to you.
- **Inner class competition stage** - You are supposed to submit your agent's executable class file to the LMS by 15.04.2017. The teaching assistant will run a tournament and your agent will be graded according to their performance against other agents.
- **Submission stage** - After informing the performance of your agent, you are expected to revise your agent strategy and run a number of experiments against ParsAgent, Atlas3 and RandomDance. You will prepare a analysis report explaining the solution and experimental results and you are going to submit your final project to the LMS by 10.05.2017. You will submit the following items:
  1. A negotiating agent programmed in Java using the negotiation environment provided to you consisting of the following components.
  2. A package containing your agent code: both the class and src files. It is obligatory to use the package `negotiator.group`*n* where *n* is your group number.
  3. A analysis report documenting, and explaining the solution and experimental results.
- **Demonstration stage** - You are going to make a demo to the teaching assistant between 11.05.2017 and 16.05.2017.

The reports need not be lengthy (10 A4 pages may be enough, 15 A4 pages maximum), but should include an explanation and motivation of *all* of the choices made in the design of negotiating agent. The report should also help the reader to understand the organization of the source code (important details should be commented on in the source code itself). This means that the main Java methods used by your agent should be explained in the report itself. The analysis report should involve an elaborate analysis of your agent's performance from different perspectives (e.g. individual utility gained, social welfare - the sum of utilities of all agents, optimality of the outcome, fairness etc.).

Please make sure all your files are in the directory structure as explained above. Assuming that the group has number three, a valid directory structure is:

```
partyrepository.xml
Group3 report.pdf
negotiator/
    group3/
        Group3.class
        Group3.java
        SomeHelperClass.class
        SomeHelperClass.java
```

1.3 Requirements

- The agent should make valid responses to the `chooseAction()` method.
- The agent could aim at the best negotiation outcome possible (i.e. the highest score for itself given the agent's preferences, while taking into account that it may need to concede to its opponents).
- The agent should take the utility of the opponents into account. Of course, initially you only have information about your own preferences. Your agent needs to learn the preferences of the opponents during the negotiation process. The default way is by constructing an opponent model to have an idea of the utility of your proposed bids for the opponents.
- The agent should learn and adapt in Multiateral Negotiations by accesing data from their past negotiations in the tournament setting (See Section 6)
- The report should include:
  - the group number
  - an introduction to the assignment
  - a high-level description of the agent and its structure, including the main Java methods (mention these explicitly!) used in the negotiating agent that have been implemented in the source code
  - an explanation of the negotiation strategy, decision function for accepting offers, any important preparatory steps, and heuristics that the agent uses to decide what to do next, including the factors that have been selected and their combination into these functions
  - a section documenting the tests you performed to improve the negotiation strength of your agent. You must include scores of various tests over multiple sessions that you performed while testing your agent against the ParsAgent, Atlas3 and RandomDance agents. Describe how you set up the testing situation and how you used the results to modify your agent
  - a conclusion in which you summarize your experience as a team with regards to building the negotiating agent and discuss what extensions are required to use your agent in real-life negotiations to support (or even take over) negotiations performed by humans.

1.4 Assignment

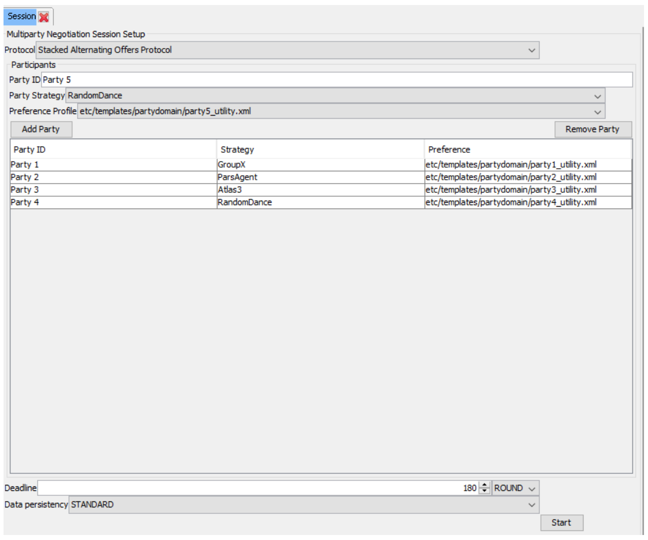
In this section, the main tasks and questions you need complete are presented. Note that the negotiation environment is a scientific software tool which is being improved constantly. Please report any major bugs found so that we can resolve them in a next version.

Negotiations are multilateral and based on a multi-player version of the alternating-offers protocol. Offers are exchanged in real time with a deadline after 3 minutes. The challenge for an agent is to negotiate with two opponents without any prior knowledge of the preferences and strategies of the opponent. As agents negotiate repeatedly with the same agents they may learn from their previous negotiations with these opponents.

GENIUS allows agents to access data from their past negotiation sessions in **Multi-Party Tournament**. The agent can access the following information:

1. The name and order of the agents involved in all their previous negotiations.
2. The utilities of the exchanged offers in any previous negotiation session (according to its own utility space). It is worth mentioning that the agents can access their own utility of a given offer. They cannot see the utilities of the other agents in any previous negotiation sessions .
3. The agreement that is reached by the agents it negotiated with.

Agents may use this information learn about and adapt to domain over time, and to use this information to negotiate better with their opponents. After implementing your negotiation agent in Genius, you need to run a "multiparty negotiation" session in order to test whether your agent is working properly (Start -> Multiparty Negotiation). Figure 1 shows how to set up a negotiation session. "Stacked Alternating Offers Protocol for Multi-Lateral Negotiation" is automatically chosen so you do not need to change this part. Then, you need to set a unique id for each agent and specify which strategy they employ and their preference profile. Each negotiation session is limited by a fixed amount of time. You can set the deadline in terms of rounds or seconds. It is worth noting that all agents have exactly the same deadline for achieving a deal and all agents "know" this. Please test your agent with different deadlines to see the effect of deadline (e.g. 10, 50, 100). Its worth noting that you cannot test learning from other sessions in a single Multi-Party Negotiation session. You need to use the Multi-Party Tournament to test learning from past sessions.



Multiparty Negotiation Session Setup

Protocol: Stacked Alternating Offers Protocol

Participants

Party ID: Party 5

Party Strategy: RandomDance

Preference Profile: etc/templates/partydomain/party5_utility.xml

Party ID	Strategy	Preference
Party 1	GroupX	etc/templates/partydomain/party1_utility.xml
Party 2	ParsAgent	etc/templates/partydomain/party2_utility.xml
Party 3	Atlas3	etc/templates/partydomain/party3_utility.xml
Party 4	RandomDance	etc/templates/partydomain/party4_utility.xml

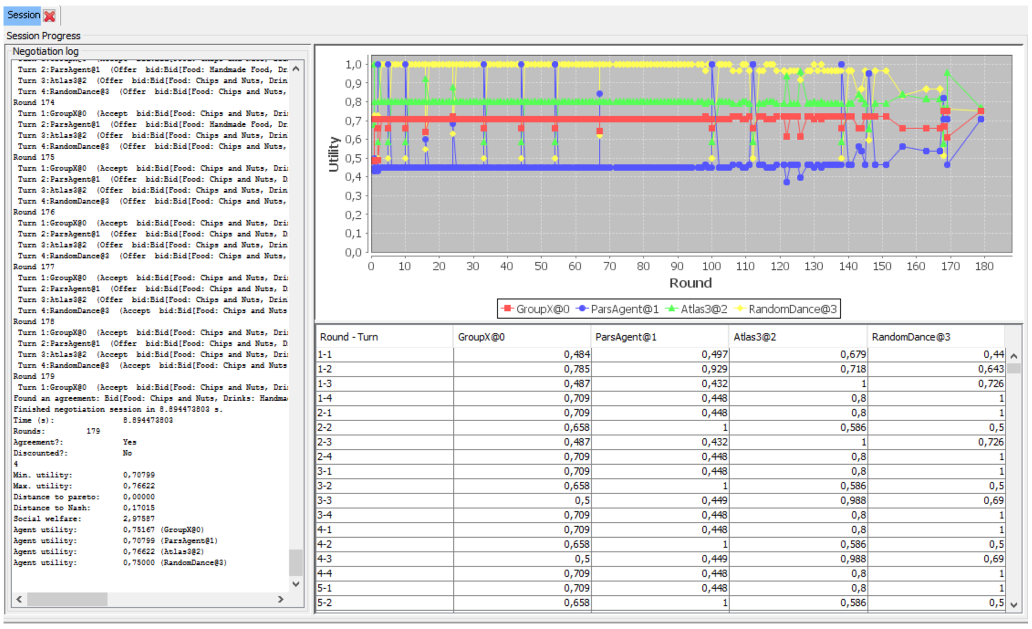
Deadline: 180 ROUND

Data persistency: STANDARD

Figure 1: Multiparty Negotiation in Genius

At the end of a session, a score is determined for the agents based on the utility of the deal for each agent if there is a deal; otherwise, the score equals the reservation value; in this assignment 0.0. Each agent will obtain a score based on the outcome and its own utility function. A failed negotiation, in the sense that no deal is reached, thus is a missed opportunity for all agents.

The outcome space of a negotiation, i.e. all possible bids, can be plotted on a graph which indicates the utility of all agents on the y axes and the round numbers on the x axes. An example is provided in Figure 2. As seen on the left side, a detailed log of negotiation is seen with a summary of negotiation analysis where the table shows the individual utilities at each round.



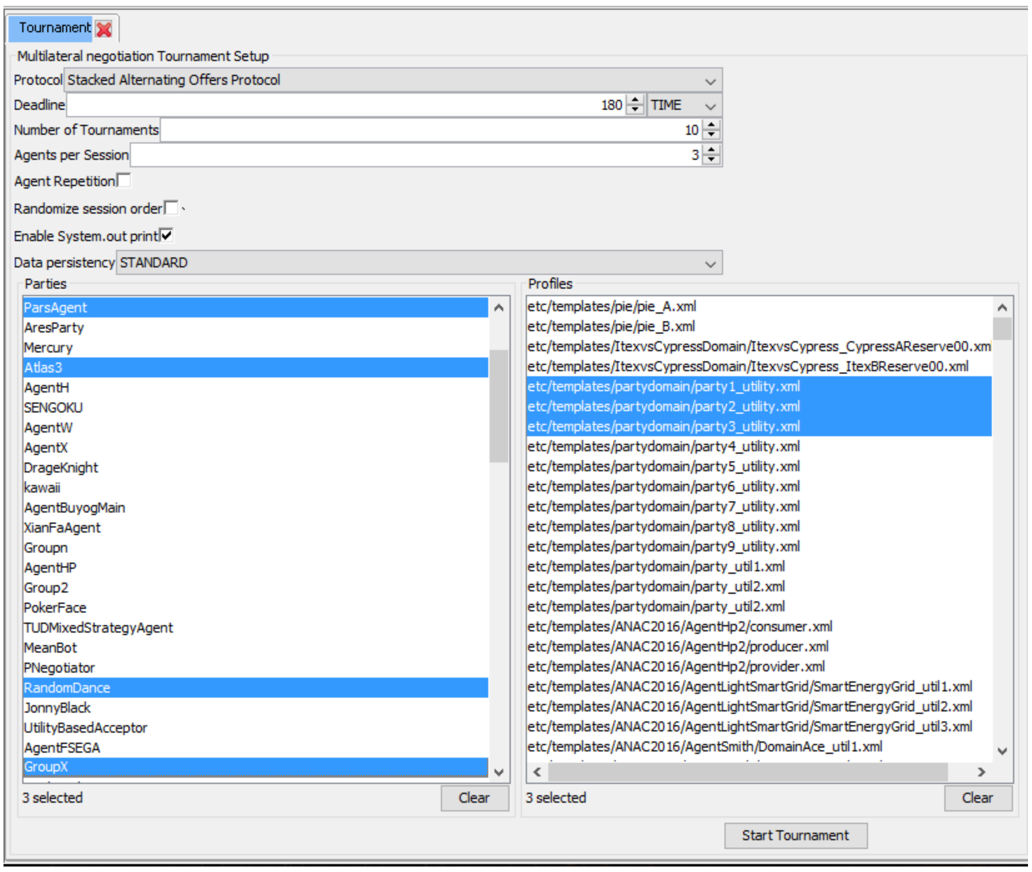
```
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Handmade Food, Dr
 Turn 3:Atlas3@2  (Offer  bid:Bid[Food: Chips and Nuts, Drin
 Turn 4:RandomDance@3  (Offer  bid:Bid[Food: Chips and Nuts,
Round 174
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Handmade Food, Dr
 Turn 3:Atlas3@2  (Offer  bid:Bid[Food: Chips and Nuts, Drin
 Turn 4:RandomDance@3  (Offer  bid:Bid[Food: Chips and Nuts,
Round 175
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Chips and Nuts, D
 Turn 3:Atlas3@2  (Offer  bid:Bid[Food: Chips and Nuts, Drin
 Turn 4:RandomDance@3  (Offer  bid:Bid[Food: Chips and Nuts,
Round 176
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Chips and Nuts, D
 Turn 3:Atlas3@2  (Offer  bid:Bid[Food: Chips and Nuts, Drin
 Turn 4:RandomDance@3  (Offer  bid:Bid[Food: Chips and Nuts,
Round 177
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Chips and Nuts, D
 Turn 3:Atlas3@2  (Offer  bid:Bid[Food: Chips and Nuts, Drin
 Turn 4:RandomDance@3  (Accept  bid:Bid[Food: Chips and Nuts
Round 178
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 2:ParsAgent@1  (Offer  bid:Bid[Food: Chips and Nuts, D
 Turn 3:Atlas3@2  (Accept  bid:Bid[Food: Chips and Nuts, Dri
 Turn 4:RandomDance@3  (Accept  bid:Bid[Food: Chips and Nuts
Round 179
 Turn 1:GroupX@0  (Accept  bid:Bid[Food: Chips and Nuts, Dri
Found an agreement: Bid[Food: Chips and Nuts, Drinks: Handma
Finished negotiation session in 8.894473803 s.
Time (s):               8.894473803
Rounds:         179
Agreement?:             Yes
Discounted?:            No
4
Min. utility:           0,70799
Max. utility:           0,76622
Distance to pareto:     0,00000
Distance to Nash:       0,17015
Social welfare:         2,97587
Agent utility:          0,75167 (GroupX@0)
Agent utility:          0,70799 (ParsAgent@1)
Agent utility:          0,76622 (Atlas3@2)
Agent utility:          0,75000 (RandomDance@3)
```

Round - Turn	GroupX@0	ParsAgent@1	Atlas3@2	RandomDance@3
1-1	0,484	0,497	0,679	0,44
1-2	0,785	0,929	0,718	0,643
1-3	0,487	0,432	1	0,726
1-4	0,709	0,448	0,8	1
2-1	0,709	0,448	0,8	1
2-2	0,658	1	0,586	0,5
2-3	0,487	0,432	1	0,726
2-4	0,709	0,448	0,8	1
3-1	0,709	0,448	0,8	1
3-2	0,658	1	0,586	0,5
3-3	0,5	0,449	0,988	0,69
3-4	0,709	0,448	0,8	1
4-1	0,709	0,448	0,8	1
4-2	0,658	1	0,586	0,5
4-3	0,5	0,449	0,988	0,69
4-4	0,709	0,448	0,8	1
5-1	0,709	0,448	0,8	1
5-2	0,658	1	0,586	0,5

Figure 2: Trace of a three party negotiation

Key to this assignment is the fact that you have incomplete information about your opponent. You may assume that there is a conflict of interests, but at the start you do not know on which issues agents agree and on which they disagree. For example, in a negotiation about buying a laptop the buyer may prefer to have a middle-sized screen but the seller may prefer to sell laptops with small screens because (s)he has more of those in stock. They could, however, agree on the brand of laptop that they want to buy/sell. An outcome of a negotiation reconciles such differences and results in an agreed solution to resolve the conflict. Ideally, such an outcome has certain properties. One of these properties is that the outcome should be Pareto optimal. An outcome is Pareto optimal when there does not exist a deal where one agent can do better and one can do better or the same (i.e. they both score higher or equal, and therefore both would prefer this new deal over the old one). Another property is related to the Nash solution concept. A Nash solution is an outcome that satisfies certain bargaining axioms and may be viewed as a "fair outcome" which is reasonable to accept for both parties. An outcome is said to be a Nash solution whenever the product of the utility (in the range [0; 1]) of the outcome for both agents is maximal ([11]). In our case, the Nash product denotes the product of utilities for all agents. Another performance criterion is related to social welfare. We measure the social welfare by summing the individual utilities that the agents gained.

After testing your agent in multiparty negotiation sessions, you are expected to run a multi-party tournament session. Figure 3 shows the interface for running a tournament on desired settings. Here, you need to set the deadline in terms of second. To do this, you need to click the button next to the deadline and set the parameters accordingly. In this assignment, the deadline for the tournament setting will be taken as 180 **seconds**. Data persistency should be set to "STANDART" so that agents can access to past session history. Agent Repetition and Randomizing Session Order should be disabled to make sessions don't have more than one of the same agent and agents participate in sessions in turn. Also in order to able to print on console to test your agents further during tournament, "Enable System.out.print" should be selected. Since each individual negotiation session may end up with a different outcome, it is desired to run the negotiation session 10 times and report the average with standard deviation. To do this, you need to set the number of tournament as 10 as shown in the figure. When you click the "start tournament", the framework will run a number of negotiation session in the background. When it completes, a log file will be generated in the folder. You need to analyze those results and report them.


Tournament

Multilateral negotiation Tournament Setup

Protocol: Stacked Alternating Offers Protocol
Deadline: 180 TIME
Number of Tournaments: 10
Agents per Session: 3
Agent Repetition: ☐
Randomize session order: ☐
Enable System.out print: ✓
Data persistency: STANDARD

Parties:
ParsAgent
AresParty
Mercury
Atlas3
AgentH
SENGOKU
AgentW
AgentX
DrageKnight
kawaii
AgentBuyogMain
XianFaAgent
Groupn
AgentHP
Group2
PokerFace
TUDMixedStrategyAgent
MeanBot
PNegotiator
RandomDance
JonnyBlack
UtilityBasedAcceptor
AgentFSEGA
GroupX

3 selected

Profiles:
etc/templates/pie/pie_A.xml
etc/templates/pie/pie_B.xml
etc/templates/ItexvsCypressDomain/ItexvsCypress_CypressAReserve00.xm
etc/templates/ItexvsCypressDomain/ItexvsCypress_ItexBReserve00.xml
etc/templates/partydomain/party1_utility.xml
etc/templates/partydomain/party2_utility.xml
etc/templates/partydomain/party3_utility.xml
etc/templates/partydomain/party4_utility.xml
etc/templates/partydomain/party5_utility.xml
etc/templates/partydomain/party6_utility.xml
etc/templates/partydomain/party7_utility.xml
etc/templates/partydomain/party8_utility.xml
etc/templates/partydomain/party9_utility.xml
etc/templates/partydomain/party_util1.xml
etc/templates/partydomain/party_util2.xml
etc/templates/partydomain/party_util2.xml
etc/templates/ANAC2016/AgentHp2/consumer.xml
etc/templates/ANAC2016/AgentHp2/producer.xml
etc/templates/ANAC2016/AgentHp2/provider.xml
etc/templates/ANAC2016/AgentLightSmartGrid/SmartEnergyGrid_util1.xml
etc/templates/ANAC2016/AgentLightSmartGrid/SmartEnergyGrid_util2.xml
etc/templates/ANAC2016/AgentLightSmartGrid/SmartEnergyGrid_util3.xml
etc/templates/ANAC2016/AgentSmith/DomainAce_util1.xml

3 selected

Figure 3: Multiparty Tournament

2 Setting up the environment

Download the latest version of eclipse from https://eclipse.org/downloads/. (If you are unfamiliar with the different flavors of Eclipse, use "Eclipse IDE for Java Developers", which will contain what we need for GENIUS development).

To run GENIUS, you need to download the latest version, 7.1 from the following link

http://ii.tudelft.nl/genius/?q=article/releases

.

For a step-by-step guide on how to set up the project, please refer to the appendix at Section 5.

Try to run the existing agent against itself to get an idea of what the program can do for you. For this practical assignment we will focus on the options `Multi-Party Negotiation` and `Multi-Party Tournament`. The former sets up a single multilateral negotiation and displays it's trace, while the later will run a tournament of multiple negotiations to get a statistically meaningful result.

The next step is to get GENIUS working in Eclipse so that you can start writing you own agent. Please note that although GENIUS will work in most editors, Eclipse is the only editor officially supported for this competition, this means that for other editors, we might not be able to help you if anything does not work. You should also make sure that the agent you provide in is runnable on our machines, which will run Eclipse and Java 7 under Windows.

If your followed the step-by-step guide in Chapter 5, your Eclipse should be set up correctly. If you want to set it up manually, make sure that the genius jar is added to the build path and run the program using the `NegoGUIApp` main method from the root of the project.

Now you can change the agent's class name from groupn to your group name (for example group3), also change the package name to your group name. Lastly you need to open `partyrepository.xml` in the root folder and edit the *classpath* value to your group name.

3 Organization

You may only use the e-mail address below to submit deliverables to the assignment or ask questions Important Dates:

1. 20 March: Deadline for registering your group. Please send an e-mail to umut.ulutas@ozu.edu.tr listing your group members
2. 21 March: We will assign a group number to you.
3. 15 April: Deadline for submiting your agents executable class file to LMS.
4. 10 May: Deadline for handing in full report about the agent along with the negotiating software agent.
5. 11-16 May: Time period for making a demonstration for your agent.

Please submit your report in PDF format and the package for your negotiating agent by mail to XXXXX. Use the naming conventions described elsewhere in this document, including your group number. Do not submit incomplete assignment solutions; only a complete assignment solution containing all deliverable will be accepted. The deadline for submitting the assignment is strict. If you have problem with your agents, please contact us in advance.

4 Evaluation

Assignments are evaluated based on several criteria. Assignments need to be complete and satisfy the requirements stated in the detailed assignment description section above. Incomplete assignments are not evaluated. That is, if any of the deliverable is incomplete or missing (or fails to run), then the assignment is not evaluated.

The assignment will be evaluated using the following evaluation criteria:

1. Presentation: Each group will present an existing negotiation strategy in the class on 15.03.2017.
2. Competition in class: The performance of the agents in class competitions has an influence on your final grade.
3. Quality of the deliverable: Overall explanation and motivation of the design of your negotiating agent; Quality and completeness of answers and explanations provided to the questions posed in the assignment; Explanatory comments in source code, quality of documentation,
4. Performance: Agents will be ranked according to negotiating strength that is measured in a tournament (see also the next section below),
5. Originality: Any original features of the agent or solutions to questions posed are evaluated positively. Note that you are required to submit original source code designed and written by your own team. Source code that is very similar to that of known agents or other students will not be evaluated and be judged as insufficient. Detection of fraud will be reported to the administration.

4.1 Competition

Agents of teams will be ranked according to negotiation strength. The ranking will be decided by playing a tournament in which every agent plays negotiation sessions against a set of agents – including the agents programmed by your peers – on a set of domains. Therefore, your agent should be generic enough to play on any domain.

Agents may be disqualified if they violate the spirit of fair play. In particular, the following behaviors are strictly prohibited: designing an agent in such a way that it benefits some specific other agent, starting new Threads, or hacking the API in any way.

With your agent, you can participate in the Fifth International Automated Negotiating Agents Competition. Please find the details in the following Web site:

http://web.tuat.ac.jp/ katfuji/ANAC2017/

.

4.2 Grading

Grades will be determined as a weighted average of several components. The grade for the practical assignment will be determined by your team solution (i.e. your assignment, the performance of your negotiating agent and report). The components that are graded and their relative weights are described in Table 1 below.

Assignment	Grading Method	Weight
Negotiating Agent	Performance in a tournament with other agents	35%
Report	Agent design and the final report about your negotiating agent.	50%
Presentation and Demonstration	Presenting skills and answering the questions.	15%

Table 1: Grading criteria and weight

4.3 Master Thesis about negotiation

Did you like thinking about efficient negotiating strategies, or implementing a successful negotiating agent? Negotiation provides a lot of subjects to do your Master Thesis on! You can always contact us for more information: reyhan.aydogan@ozyegin.edu.tr or R.Aydogan@tudelft.nl.

Acknowledgements

This assignment could not have been written without the help of many people, including R. Aydoğan, T. Baarslag, D. Festen, B. Grundeken, M. Hendrikx, K. Hindriks, C. Jonker, W. Pasman, D. Tykhonov, and W. Visser.

5 Get Genius up and Running in Eclipse

1. If you do not have Eclipse IDE, please download the latest version of eclipse from https://eclipse.org/downloads/. You need to use "Eclipse IDE for Java Developers", which will contain what we need for Genius development and make use that you have Java 7.

2. After downloading the Genius 7.1, extract its contents.

3. Create a new java project on eclipse (File -> New -> Java Project). Name it "Genius" and set JRE to "1.6" (See Figure 4).

4. Click Next, go to "Libraries" tab. Add "negosimulator.jar" (located in Genius main folder) as external JAR by clicking "Add External JARs..." (See Figure 5). Click Finish.

5. Copy ALL the contents of genius folder to your created project folder in workspace. (See Figure 6)

6. Refresh the project on Eclipse (Click the project on project explorer and hit F5.)
7. To run the GENIUS in Eclipse, you need to choose "Run as Java application" option and specify "NegoGuiApp" for the Java application (See Figure7 and Figure 8).
8. You can work on the template negotiating party class "GroupX.java" located in "storageexample" folder. If you want to change the name of this class, please do not forget to update the `partyrepository.xml` in the root folder accordingly.

6 Learning and Adapting from Previous Sessions

The agent can access the following information:

1. The name and order of the agents involved in all their previous negotiations.
2. The utilities of the exchanged offers in any previous negotiation session (according to its own utility space). It is worth mentioning that the agents can access their own utility of a given offer. They cannot see the utilities of the other agents in any previous negotiation sessions .
3. The agreement that is reached by the agents it negotiated with.

Agents may use this information learn about and adapt to domain over time, and to use this information to negotiate better with their opponents. For details, please refer to the Genius manual and the frequently asked questions.

http://tinyurl.com/ANAC2017GeniusFAQ

.

For an example agent that can access to session history, run the GroupX.java located in "storageexample" folder in Genius.

References

[1] Reyhan Aydoğan, David Festen, Koen Hindriks, and C. M. Jonker. Alternating offers protocol for multilateral negotiation. In K. Fujita, Q. Bai, T. Ito, M. Zhang, F. Ren, R. Aydoğan, and R. Hadfi, editors, *Modern Approaches to Agent-based Complex Automated Negotiation*. Springer International Publishing, 2017.

[2] Tim Baarslag, Koen Hindriks, and Catholijn Jonker. A tit for tat negotiation strategy for real-time bilateral negotiations. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 229–233. Springer Berlin Heidelberg, 2013.

[3] Mai Ben Adar, Nadav Sofy, and Avshalom Elimelech. Gahboninho: Strategy for balancing pressure and compromise in automated negotiation. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 205–208. Springer Berlin Heidelberg, 2013.

[4] A.S.Y. Dirkzwager, M.J.C. Hendrikx, and J.R. Ruiter. The negotiator: A dynamic strategy for bilateral negotiations with time-based discounts. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 217–221. Springer Berlin Heidelberg, 2013.

[5] P. Faratin, C. Sierra, and N.R. Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3):159–182, 1998.

[6] Radmila Fishel, Maya Bercovitch, and Ya'akov(Kobi) Gal. Bram agent. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 213–216. Springer Berlin Heidelberg, 2013.

[7] Asaf Frieder and Gal Miller. Value model agent: A novel preference profiler for negotiation with agents. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 199–203. Springer Berlin Heidelberg, 2013.

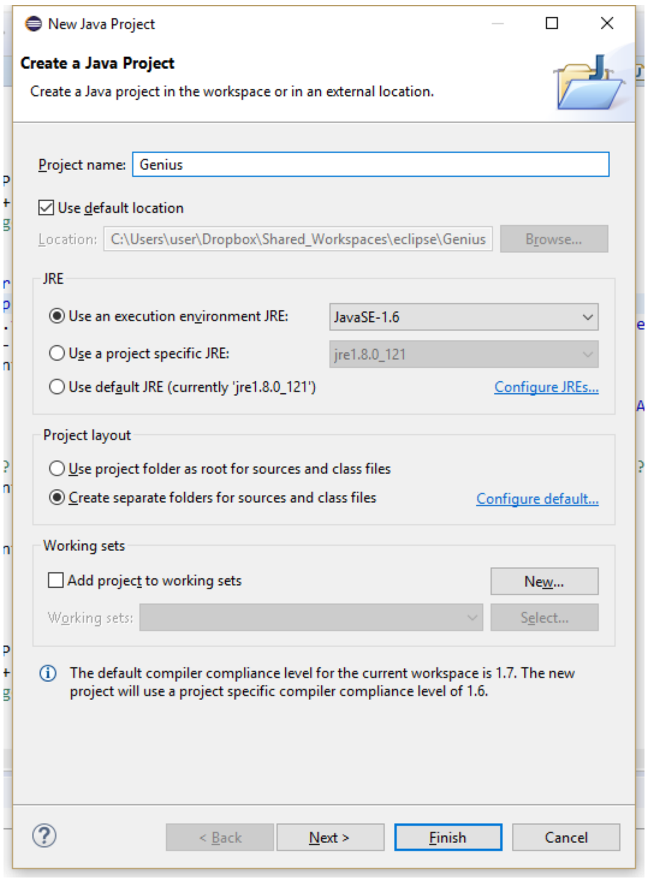
[8] Shogo Kawaguchi, Katsuhide Fujita, and Takayuki Ito. Agentk2: Compromising strategy based on estimated maximum utility for automated negotiating agents. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 235–241. Springer Berlin Heidelberg, 2013.

[9] Thijs Krimpen, Daphne Looije, and Siamak Hajizadeh. Hardheaded. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 223–227. Springer Berlin Heidelberg, 2013.

[10] Raz Lin, Sarit Kraus, Tim Baarslag, Dmytro Tykhonov, Koen Hindriks, and Catholijn M. Jonker. Genius: An integrated environment for supporting the design of generic automated negotiators. *Computational Intelligence*, 30(1):48–70, 2014.

[11] Roberto Serrano. Bargaining, 2005. URL = http://levine.sscnet.ucla.edu/econ504/bargaining.pdf, November, 2005.

[12] Colin R. Williams, Valentin Robu, Enrico H. Gerding, and Nicholas R. Jennings. Iamhaggler2011: A gaussian process regression based negotiation agent. In Takayuki Ito, Minjie Zhang, Valentin Robu, and Tokuro Matsuo, editors, *Complex Automated Negotiations: Theories, Models, and Software Competitions*, volume 435 of *Studies in Computational Intelligence*, pages 209–212. Springer Berlin Heidelberg, 2013.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Genius

☑ Use default location

Location: C:\Users\user\Dropbox\Shared_Workspaces\eclipse\Genius

JRE

● Use an execution environment JRE: JavaSE-1.6

○ Use a project specific JRE: jre1.8.0_121

○ Use default JRE (currently 'jre1.8.0_121')

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

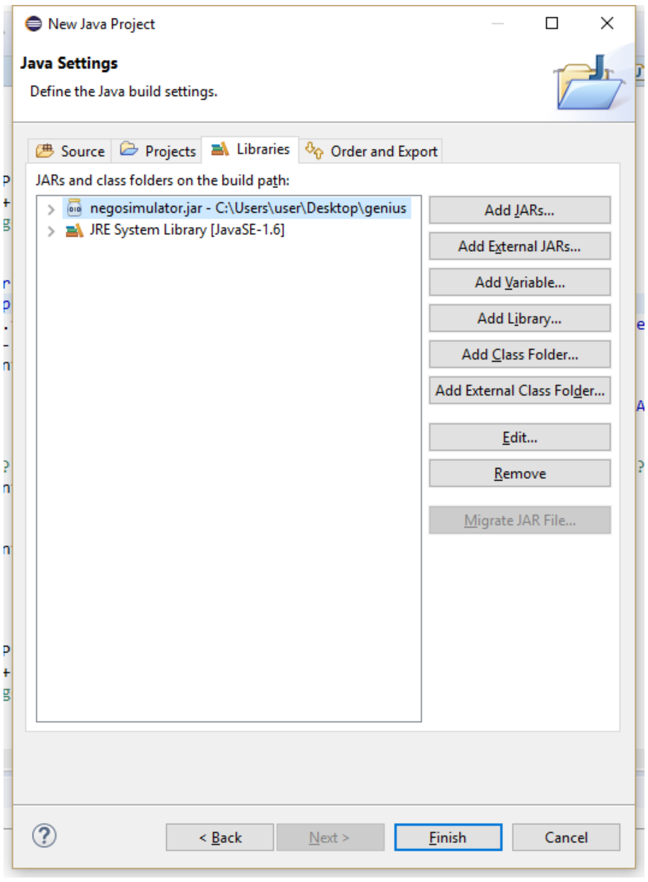
Working sets

☐ Add project to working sets

Working sets:

The default compiler compliance level for the current workspace is 1.7. The new project will use a project specific compiler compliance level of 1.6.

Figure 4: How to run genius - Part 1


New Java Project

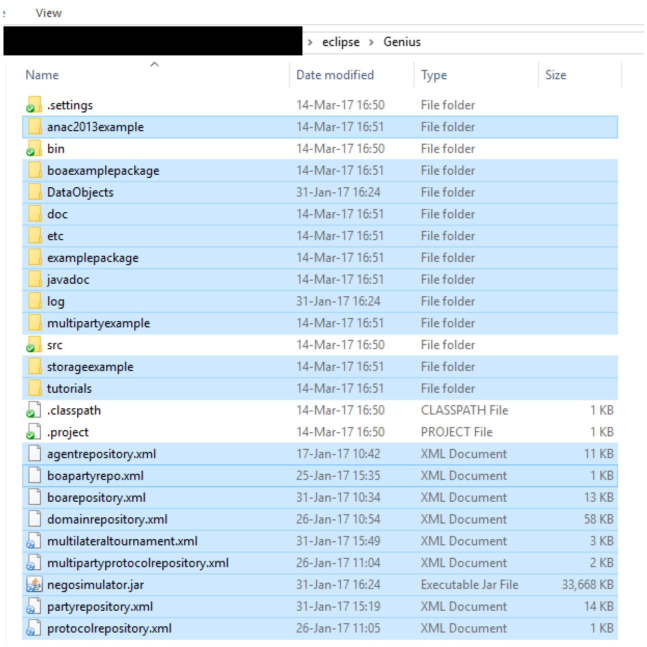
Java Settings

Define the Java build settings.

JARs and class folders on the build path:

- negosimulator.jar - C:\Users\user\Desktop\genius
- JRE System Library [JavaSE-1.6]

Figure 5: How to run genius - Part 2



View

[redacted] › eclipse › Genius

Name	Date modified	Type	Size
.settings	14-Mar-17 16:50	File folder	
anac2013example	14-Mar-17 16:51	File folder	
bin	14-Mar-17 16:50	File folder	
boaexamplepackage	14-Mar-17 16:51	File folder	
DataObjects	31-Jan-17 16:24	File folder	
doc	14-Mar-17 16:51	File folder	
etc	14-Mar-17 16:51	File folder	
examplepackage	14-Mar-17 16:51	File folder	
javadoc	14-Mar-17 16:51	File folder	
log	31-Jan-17 16:24	File folder	
multipartyexample	14-Mar-17 16:51	File folder	
src	14-Mar-17 16:50	File folder	
storageexample	14-Mar-17 16:51	File folder	
tutorials	14-Mar-17 16:51	File folder	
.classpath	14-Mar-17 16:50	CLASSPATH File	1 KB
.project	14-Mar-17 16:50	PROJECT File	1 KB
agentrepository.xml	17-Jan-17 10:42	XML Document	11 KB
boapartyrepo.xml	25-Jan-17 15:35	XML Document	1 KB
boarepository.xml	31-Jan-17 10:34	XML Document	13 KB
domainrepository.xml	26-Jan-17 10:54	XML Document	58 KB
multilateraltournament.xml	31-Jan-17 15:49	XML Document	3 KB
multipartyprotocolrepository.xml	26-Jan-17 11:04	XML Document	2 KB
negosimulator.jar	31-Jan-17 16:24	Executable Jar File	33,668 KB
partyrepository.xml	31-Jan-17 15:19	XML Document	14 KB
protocolrepository.xml	26-Jan-17 11:05	XML Document	1 KB

Figure 6: How to run genius - Part 3

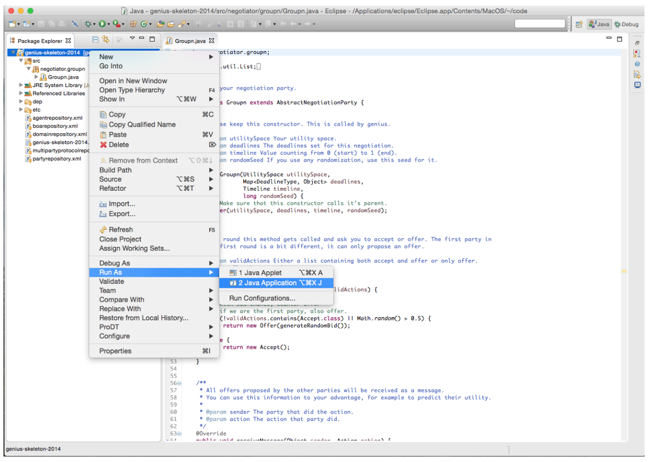
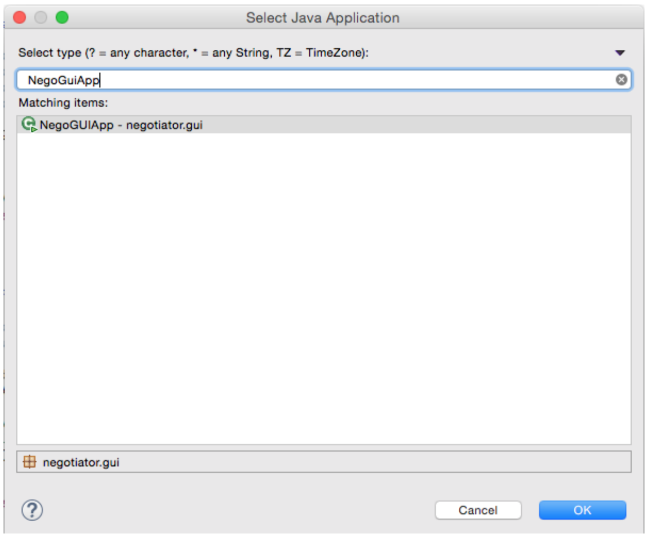


Figure 7: How to run genius - Part 4



Select Java Application

Select type (? = any character, * = any String, TZ = TimeZone):

NegoGuiApp

Matching items:

NegoGUIApp - negotiator.gui

negotiator.gui

Figure 8: How to run genius - Part 5