IN4010 Practical assignment 2: Automated Negotiation

December 9, 2014

1 Introduction

This document describes the practical assignment for the second quarter of the AI course. 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 two or more 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.

- Think of and design three negotiation scenarios for three-party negotiation. This involves:
  - Define a multi-issue negotiation domain with discrete issues,
  - Create three negotiation scenarios on this domain with varying degrees of conflicts among the negotiation parties,
    - For each scenario, create three conflicting preference profiles,
    - Think of the degree of conflict to categorize those scenarios by yourselves (collaborative, moderate and competitive)
- Design and implement a negotiating agent in GENIUS for multiparty negotiation.
- Test the performance of the agents in the negotiation scenarios that you have created.
- Prepare a report explaining your solution
- Compare your agents with other agents developed by other students in the negotiation scenarios provided in the second phase of this assignment
- Improve your agents
- Prepare a final analysis report with explanation of changes that you make in the second phase.

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, you are asked to **define three negotiation scenarios with varying difficulty to find a joint agreement among three negotiation parties** (e.g. collaborative, moderate, and competitive). A negotiation scenario consists of well structured negotiation domain (e.g. a set of negotiation issues and possible values for each issue) and preference profiles of the negotiation parties. The first step is to define a negotiation domain consisting of *discrete issues* by using GENIUS. GENIUS is a negotiation environment that implements an open architecture for heterogeneous negotiating agents [12]. 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.

To see how to create a negotiation domain in GENIUS, you can look at Section 6.2. The second step is to create nine preference profiles where the first three (1-3), the second three (4-6) and last three (7-9) will be used to form the first scenario, the second scenario and the third scenario respectively. Please use the following naming conversion for your preference profiles (*domainName*-ProfileX) where X is the id of the scenario (e.g. between 1 and 9). For example, if the domain name is "Party", you should name the first preference profile as "Party-Profile1". In the first negotiation scenario (Profile1, Profile2, and Profile3), there would be more chance to find a mutually acceptable bids compared to the second scenario (Profile 4, Profile5, and Profile6). When we consider the last scenario, we expect that there are less (possible) mutually acceptable bids for all parties compared to the other scenarios. In other words, the last scenario includes more conflicting preference profiles compared to the other scenarios. To see how to create a preference profile in Genius, you can look at Section 6.3. The second task involves thinking about a strategy used to perform the negotiation itself. But the most important part of this assignment concerns the negotiation phase itself, i.e. the exchange of 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)**". 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.

The following block will give 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.

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. Some techniques relevant for building and designing negotiating agents can be found among others in chapters 16 and 17 in [14]. The assignment will also require you to go beyond the course material in the book. Additional information is provided to you in the form of several papers [1, 9, 12, 5, 15]. There is a lot of other literature available about negotiation that may help you finish this assignment successfully. As for almost any subject, you can find more information about negotiation strategies online. We recommend to search for strategies used in the ANAC Competition [2, 3, 4, 7, 8, 10, 11, 16]. 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.

This assignment consists of two phases. In the first phase, you will create three negotiation scenarios, design and implement a negotiating agent in a multilateral negotiation setting, prepare a full report about your agent and the scenarios that you created. In the second phase, we will provide you a set of negotiation agents and a set of negotiation scenarios. Therefore, you can compare your strategy with others in agent repository in terms of average utility gained, optimality of the agreement, the duration of agreement (e.g. number of rounds to complete the negotiation), and so on. You are asked to evaluate your agent's performance and improve your strategy. You will prepare a final analysis report with explanation of changes that you make in the second phase.

The remainder of this document is organized as follows. In Section 2 the objectives, deliverables, requirements and assignment itself are described. Section 4 describes some organizational details and important dates, including deadlines. Finally, Section 5 documents the evaluation criteria and grading for this assignment.

2 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

2.1 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.

2.2 Deliverables

- A unique number n to identify the team and the negotiating agent (which will be provided to you after registering your group).
- A negotiating agent programmed in Java using the negotiation environment provided to you consisting of the following components.
  - 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.
- A report documenting and explaining the solution.
- A final analysis report including your changes and their explanations

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 first 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 final 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:

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

2.3 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 must take the utility of the opponents into account. Of course, initially you only have information about your own preferences. You will have 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 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. 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.

2.4 Assignment

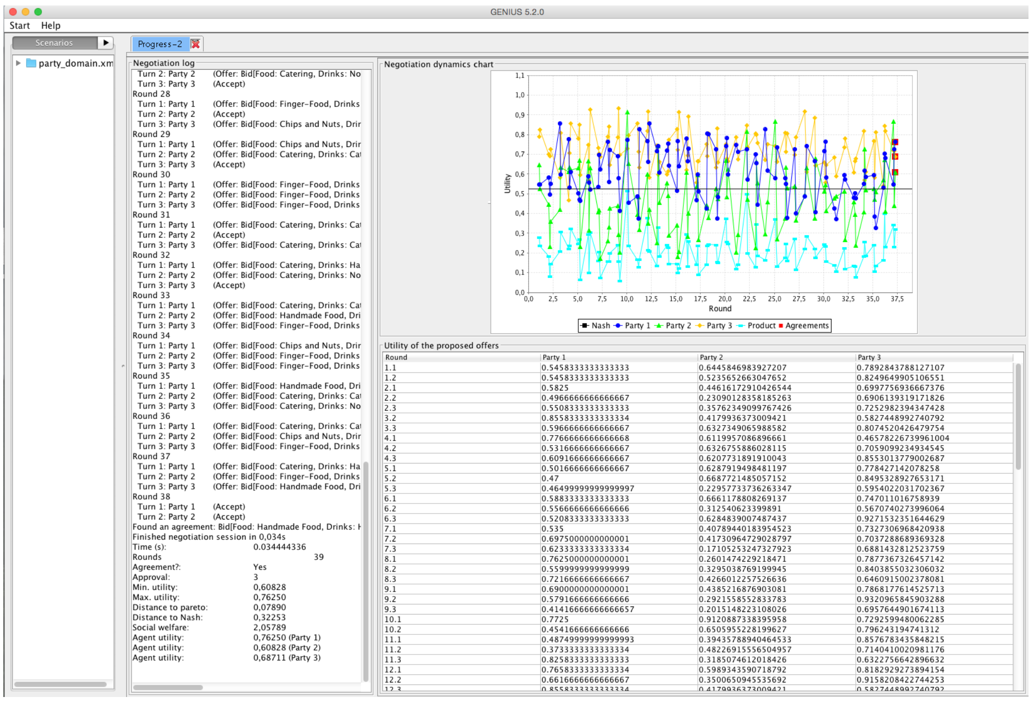
In this section, the main tasks and questions you need complete are presented. But before we do this, we present additional background that may help you to complete this assignment successfully. Note that the negotiation environment is a scientific software tool which is being improved constantly. Please report any major bugs found so that we can resolve them in a next version.

A negotiation in this assignment is also called a negotiation session. In a session agents negotiate with each other to settle a conflict and negotiate a deal. Each negotiation session is limited by a fixed amount of time. At the end of a session, a score is determined for both 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. In a sense, there is no winner since 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 both agents. In the tournament that will be played, each agent will negotiate with all other agents and the scores of each session are recorded and averaged to obtain an overall score for the agent (see Section 4). A ranking will be compiled using these overall scores.

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 the first agent and that for the second agent is maximal ([15]). In our case, the Nash product denotes the product of utilities for all agents. The notion of a fair outcome is important in negotiation because it provides a reference point for what your opponent might be willing to accept. Typically, a negotiation is started since reaching an agreement is better than not. But in order to get an agreement, you will need to get your opponents to agree. In a negotiation between self-interested agents, however, each agent is determined to get the best possible outcome for itself.

The Nash solution concept, as it is called, excludes certain outcomes as being unfair. It is, for example, very unlikely that your opponent will accept an offer that is most favorable to you and leaves your opponent with empty hands. In general, it is to be expected that an outcome is the result of a number of concessions that both parties make. Concessions can be made in various ways, quite easily or more slowly. The speed of making concessions, or the concession rate is the derivative of the (size of the) concession steps taken during a negotiation. An agent that makes concessions in very small steps initially uses a so-called Boulware strategy. A strategy that makes faster concessions initially is called a Conceder. Other strategies are conceivable and may be necessary given the deadlines (cf. [6])). To compute whether a bid in a negotiation is Pareto optimal or a Nash solution we use so-called utility functions (cf. [13]). In our case, utility functions assign quantitative values to bids in the negotiation space. In this assignment, you may assume that all utility functions are additive, i.e. they will always be linear in the number of issues. 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 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 1.



Negotiation log

```
  Turn 2: Party 2      (Offer: Bid[Food: Catering, Drinks: No
  Turn 3: Party 3      (Accept)
Round 28
  Turn 1: Party 1      (Offer: Bid[Food: Finger-Food, Drinks
  Turn 2: Party 2      (Accept)
  Turn 3: Party 3      (Offer: Bid[Food: Chips and Nuts, Drir
Round 29
  Turn 1: Party 1      (Offer: Bid[Food: Chips and Nuts, Drir
  Turn 2: Party 2      (Offer: Bid[Food: Catering, Drinks: Cat
  Turn 3: Party 3      (Accept)
Round 30
  Turn 1: Party 1      (Offer: Bid[Food: Finger-Food, Drinks
  Turn 2: Party 2      (Offer: Bid[Food: Finger-Food, Drinks
  Turn 3: Party 3      (Offer: Bid[Food: Finger-Food, Drinks
Round 31
  Turn 1: Party 1      (Offer: Bid[Food: Catering, Drinks: Cat
  Turn 2: Party 2      (Accept)
  Turn 3: Party 3      (Offer: Bid[Food: Catering, Drinks: Cat
Round 32
  Turn 1: Party 1      (Offer: Bid[Food: Catering, Drinks: Ha
  Turn 2: Party 2      (Offer: Bid[Food: Catering, Drinks: No
  Turn 3: Party 3      (Accept)
Round 33
  Turn 1: Party 1      (Offer: Bid[Food: Catering, Drinks: Cat
  Turn 2: Party 2      (Offer: Bid[Food: Handmade Food, Dri
  Turn 3: Party 3      (Offer: Bid[Food: Finger-Food, Drinks
Round 34
  Turn 1: Party 1      (Offer: Bid[Food: Chips and Nuts, Drir
  Turn 2: Party 2      (Offer: Bid[Food: Finger-Food, Drinks
  Turn 3: Party 3      (Offer: Bid[Food: Finger-Food, Drinks
Round 35
  Turn 1: Party 1      (Offer: Bid[Food: Handmade Food, Dri
  Turn 2: Party 2      (Offer: Bid[Food: Catering, Drinks: Cat
  Turn 3: Party 3      (Offer: Bid[Food: Catering, Drinks: No
Round 36
  Turn 1: Party 1      (Offer: Bid[Food: Catering, Drinks: Cat
  Turn 2: Party 2      (Offer: Bid[Food: Chips and Nuts, Drir
  Turn 3: Party 3      (Offer: Bid[Food: Finger-Food, Drinks
Round 37
  Turn 1: Party 1      (Offer: Bid[Food: Catering, Drinks: Ha
  Turn 2: Party 2      (Offer: Bid[Food: Finger-Food, Drinks
  Turn 3: Party 3      (Offer: Bid[Food: Handmade Food, Dri
Round 38
  Turn 1: Party 1      (Accept)
  Turn 2: Party 2      (Accept)
Found an agreement: Bid[Food: Handmade Food, Drinks: H
Finished negotiation session in 0,034s
Time (s):                      0.034444336
Rounds                                     39
Agreement?:                    Yes
Approval:                      3
Min. utility:                  0,60828
Max. utility:                  0,76250
Distance to pareto:            0,07890
Distance to Nash:              0,32253
Social welfare:                2,05789
Agent utility:                 0,76250 (Party 1)
Agent utility:                 0,60828 (Party 2)
Agent utility:                 0,68711 (Party 3)
```

Utility of the proposed offers

Round	Party 1	Party 2	Party 3
1.1	0.5458333333333333	0.6445846983927207	0.7892843788127107
1.2	0.5458333333333333	0.5235652663047652	0.8249649905106551
2.1	0.5825	0.44616172910426544	0.6997756936667376
2.2	0.4966666666666667	0.23090128358185263	0.6906139319171826
2.3	0.5508333333333333	0.35762349099767426	0.7252982394347428
3.2	0.8558333333333334	0.4179936373009421	0.5827448992740792
3.3	0.5966666666666667	0.6327349065988582	0.8074520426479754
4.1	0.7766666666666668	0.6119957086896661	0.46578226739961004
4.2	0.5316666666666667	0.6326755886028115	0.7059099234934545
4.3	0.6091666666666667	0.6207731891910043	0.8553013779002687
5.1	0.5016666666666667	0.6287919498481197	0.778427142078258
5.2	0.47	0.6687721485057152	0.8495328927653171
5.3	0.46499999999999997	0.22957733736263347	0.5954022031702367
6.1	0.5883333333333333	0.6661178808269137	0.747011016758939
6.2	0.5566666666666666	0.312540623399891	0.5670740273996064
6.3	0.5208333333333333	0.6284839007487437	0.9271532351644629
7.1	0.535	0.40789440183954523	0.7327306968420938
7.2	0.6975000000000001	0.41730964729028797	0.7037288689369328
7.3	0.6233333333333334	0.17105253247327923	0.6881432812523759
8.1	0.7625000000000001	0.2601474229218471	0.7877367326457142
8.2	0.5599999999999999	0.3295038769199945	0.8403855032306032
8.3	0.7216666666666667	0.4266012257526636	0.6460915002378081
9.1	0.6900000000000001	0.4385216876903081	0.7868177614525713
9.2	0.5791666666666666	0.2921558552833783	0.9320965845903288
9.3	0.41416666666666657	0.2015148223108026	0.6957644901674113
10.1	0.7725	0.9120887338395958	0.7292599480062285
10.2	0.4541666666666666	0.6505955228199627	0.796243194741312
11.1	0.48749999999999993	0.39435788940464533	0.8576783435848215
11.2	0.3733333333333334	0.48226915556504957	0.7140410020981176
11.3	0.8258333333333333	0.3185074612018426	0.6322756642896632
12.1	0.7658333333333334	0.5989343590718792	0.8182929273894154
12.2	0.6616666666666667	0.3500650945535692	0.9158208422744253
12.3	0.8558333333333334	0.4179936373009421	0.5827448992740792

Figure 1: Trace of a three party negotiation

What sets the negotiations considered in this assignment apart from other negotiations is the fact that all agents have exactly the same deadline for achieving a deal and all agents "know" this.

3 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 or fork the skeleton project provided to you on github via the following link:

https://github.com/festen/genius-skeleton-2014

.

For a step-by-step guide on how to set up the project, please refer to the appendix at Section 6.1.

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 the practical assignment, 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 assignment you handed in is runnable on our machines, which will run Eclipse and Java 7.

If you downloaded a fresh copy of the GitHub repository or followed the step-by-step guide in Chapter 6.1, 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 `multipartyprotocolrepository.xml` in the root folder and edit the *classpath* value to your group name.

4 Organization

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

1. 10 December: Deadline for registering your group. Please send an e-mail to ai@ii.tudelft.nl listing your group members
2. 10 December: We will assign a group number to you.
3. 15 January, 09:00: Deadline for handing in full report about the agent & scenarios you created along with the agent and scenarios
4. 2 February, 09.00: Submitting the final analysis report with the final negotiating software agent

Please submit your report in PDF format and the package for your negotiating agent by mail to ai@ii.tudelft.nl. 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 deliverables will be accepted. The deadline for submitting the assignment is strict. If you have problem with your agents, please contact us in advance.

5 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 deliverables 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. Quality of the deliverables: 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,
2. Performance: Agents will be ranked according to negotiating strength that is measured in a tournament (see also the next section below),
3. 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.

5.1 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.

5.2 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.	50%
Report	Agent design and the final report about your negotiating agent.	50%

Table 1: Grading criteria and weight

During the semester, you had the opportunity to participate in a PN negotiation session. If you did, the utility you gained during the experiment will be added to your final score as a bonus. The students whose opponent walked away before they reach agreement, will get 0.2.

5.3 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 ask the course assistants, have a look at the last page of this assignment, or contact us for more information: R.Aydogan@tudelft.nl.

Acknowledgements

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

6 Appendix

6.1 Get Genius up and running in Eclipse

1. To set up Genius, clone the git repository in Eclipse at `https://github.com/festen/genius-skeleton-2014.`
2. Run the `NegoGuiApp` class from the root folder

NB: A step by step guide in pictures is provided at the end of the appendix.

6.2 Create a negotiation domain in Genius

To create a negotiation domain, you can do the followings:

1. Run Genius
2. Right click on the "Scenarios" tab and choose "New domain" (See Figure 14)
3. Name your domain
4. Click "Add issue" button
5. Name the issue and enter the possible values for this issue (See Figure 15)
6. Repeat 4-5 as much as you need.

7. Click "Save changes" button

6.3 Create a negotiation profile in Genius

1. Run Genius
2. Right click on the "Scenarios" tab and choose "New preference profile" (See Figure 17)
3. Choose weights
4. Click "Save changes" button

References

[1] Tim Baarslag, Koen Hindriks, Mark Hendrikx, Alex Dirkzwager, and Catholijn Jonker. Decoupling negotiating agents to explore the space of negotiation strategies. In *Proceedings of The Fifth International Workshop on Agent-based Complex Automated Negotiations (ACAN 2012)*, 2012. URL = http://mmi.tudelft.nl/sites/default/files/boa.pdf.

[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] S. Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings. Optimal negotiation strategies for agents with incomplete information. In John-Jules Ch. Meyer and Milind Tambe, editors, *Proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 53–68, 2001. URL = citeseer.ist.psu.edu/fatima01optimal.html, November 2006.

[7] 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.

[8] 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.

[9] C.M. Jonker and J. Treur. An agent architecture for multi-attribute negotiation. In *International joint conference on artificial intelligence*, volume 17, pages 1195–1201. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[10] 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.

[11] 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.

[12] 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.

[13] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010.

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

[16] 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.

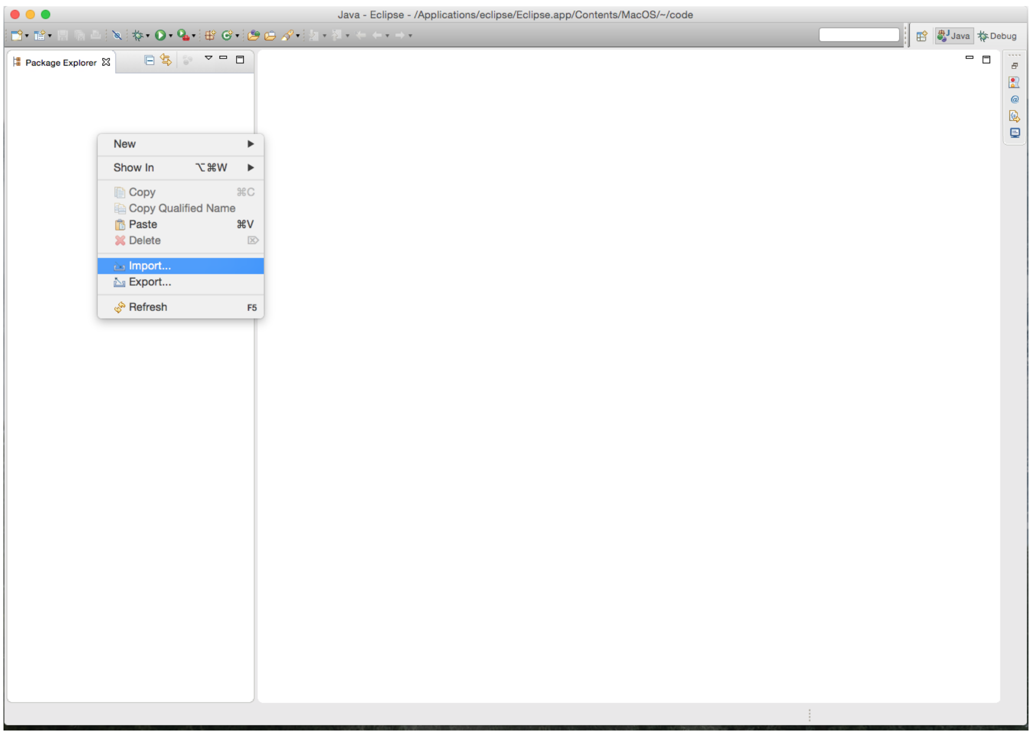
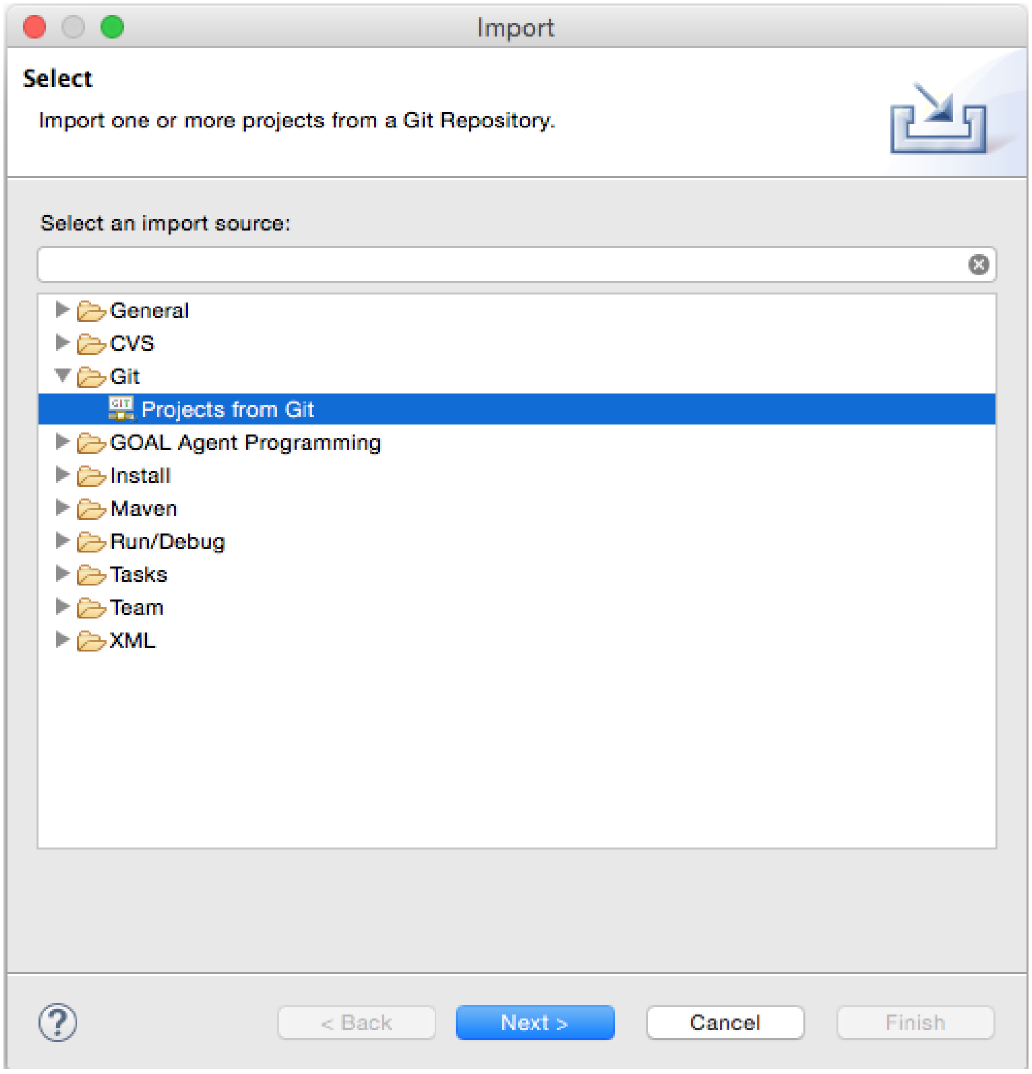


Figure 2: How to run genius - Part 1



Import

Select

Import one or more projects from a Git Repository.

Select an import source:

- General
- CVS
- Git
  - Projects from Git
- GOAL Agent Programming
- Install
- Maven
- Run/Debug
- Tasks
- Team
- XML

Figure 3: How to run genius - Part 2

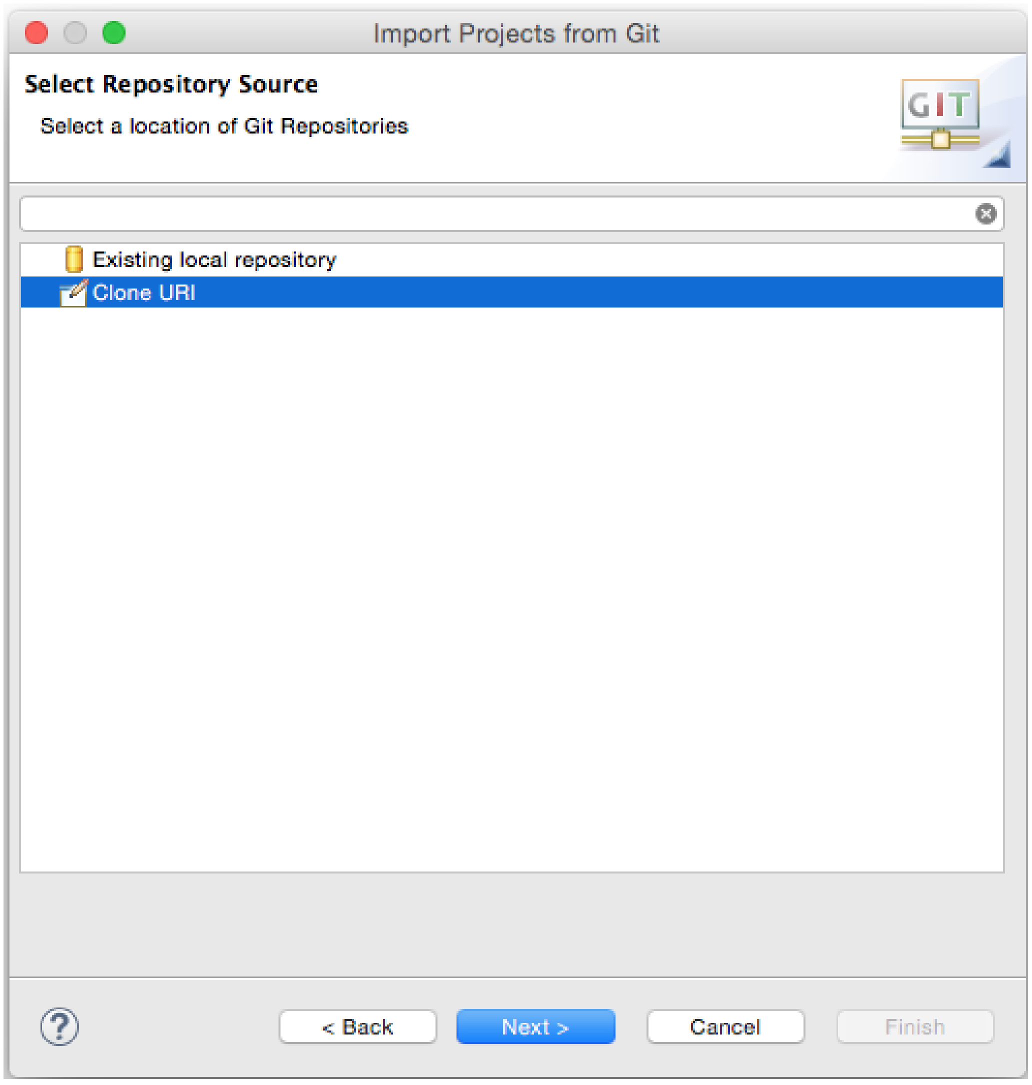
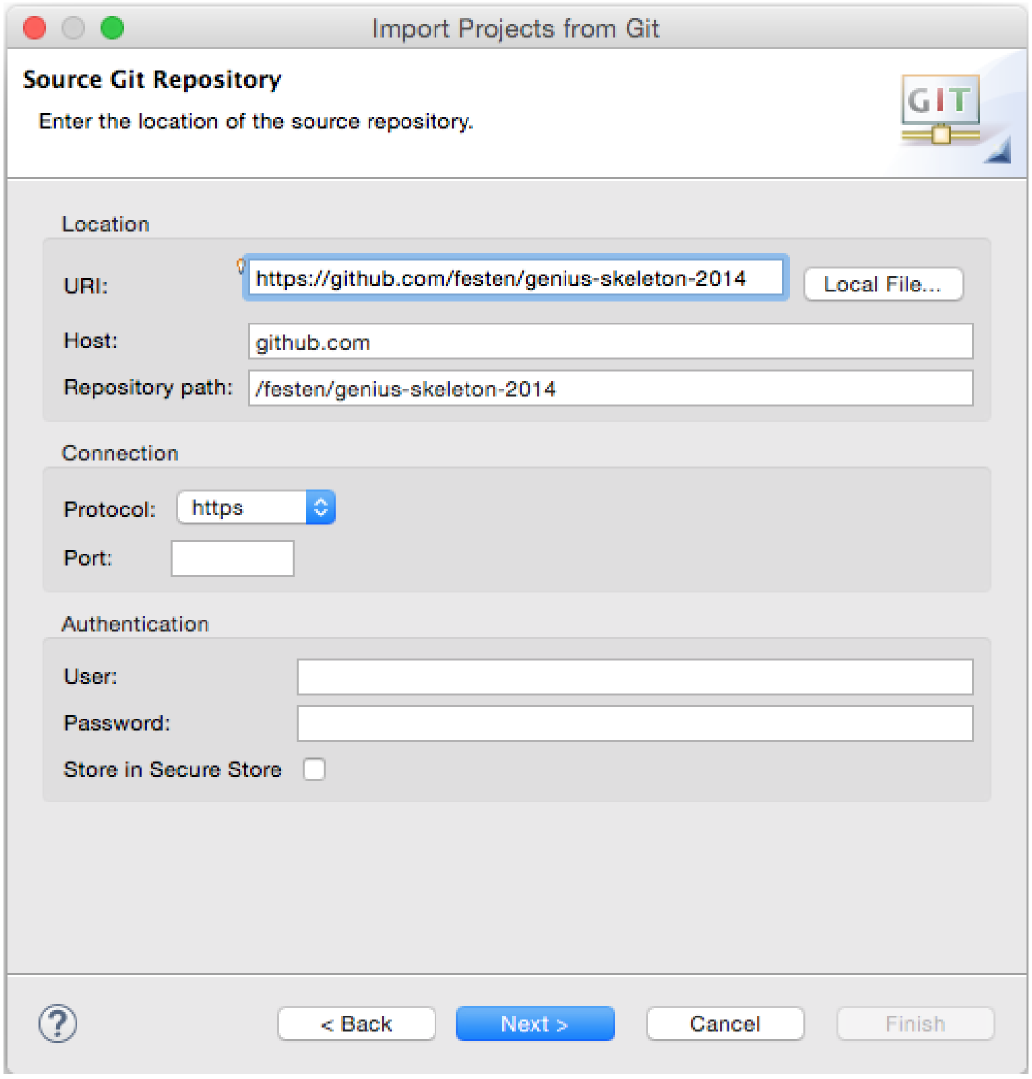


Figure 4: How to run genius - Part 3


Import Projects from Git

Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/festen/genius-skeleton-2014

Host: github.com

Repository path: /festen/genius-skeleton-2014

Connection

Protocol: https

Port:

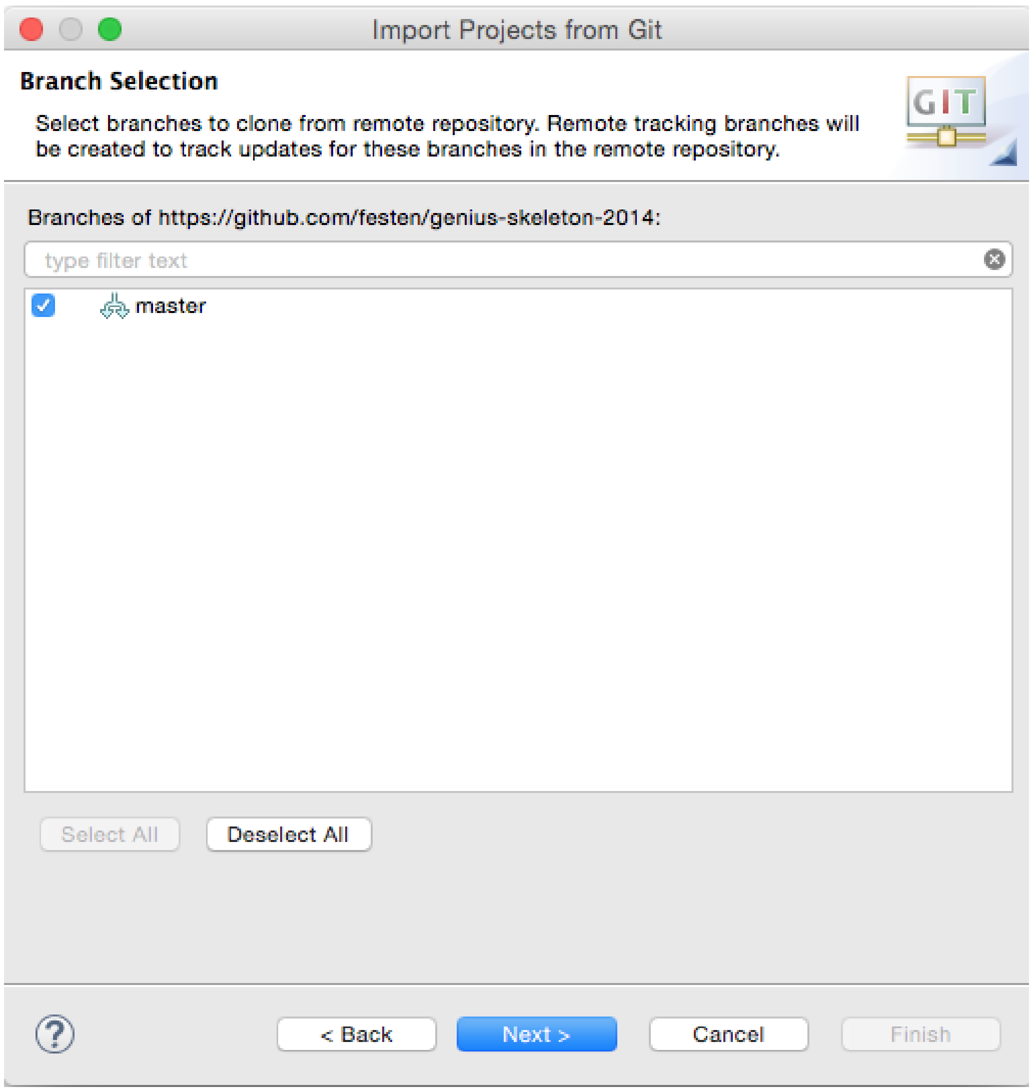
Authentication

User:

Password:

Store in Secure Store ☐

Figure 5: How to run genius - Part 4



Import Projects from Git

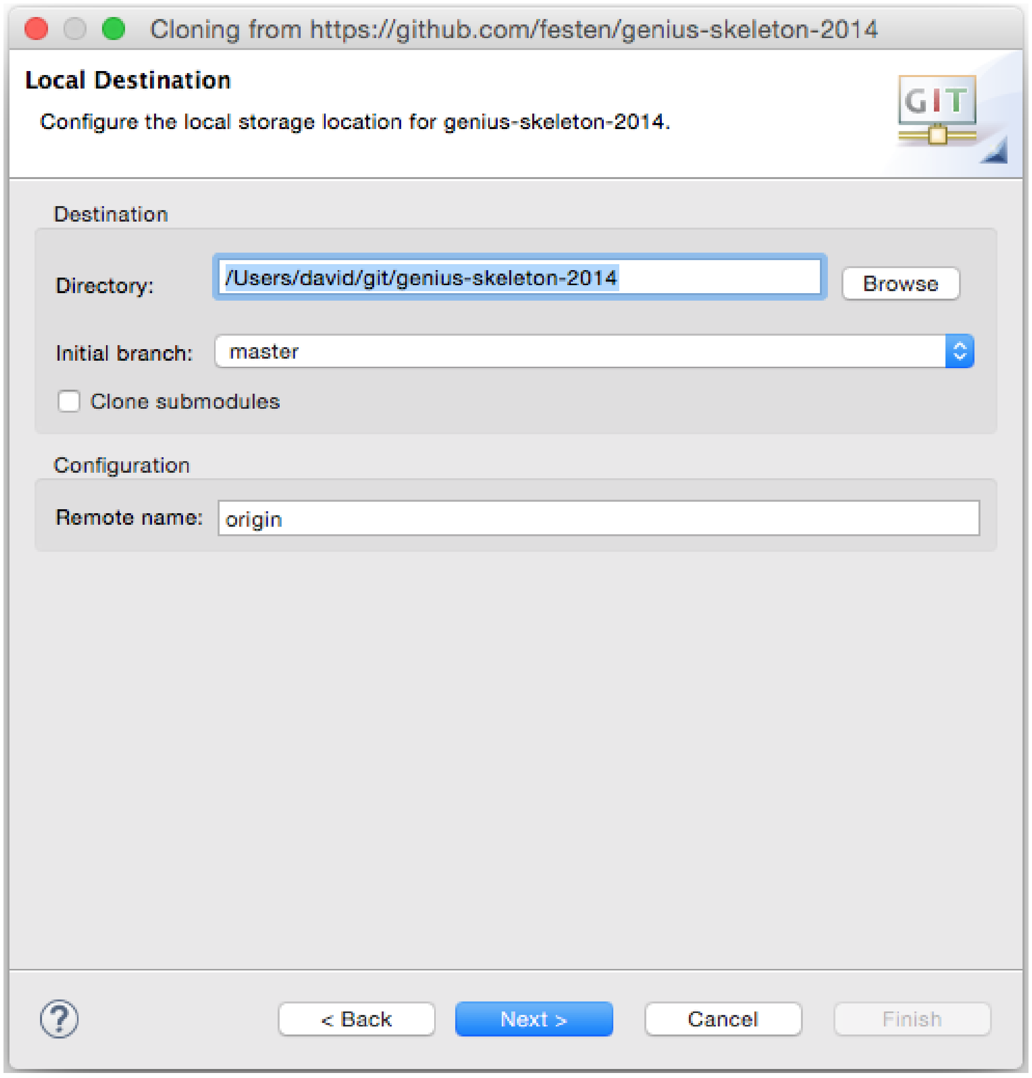
Branch Selection

Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.

Branches of https://github.com/festen/genius-skeleton-2014:

[✓] master

Figure 6: How to run genius - Part 5


Cloning from https://github.com/festen/genius-skeleton-2014

Local Destination

Configure the local storage location for genius-skeleton-2014.

Destination

Directory: /Users/david/git/genius-skeleton-2014

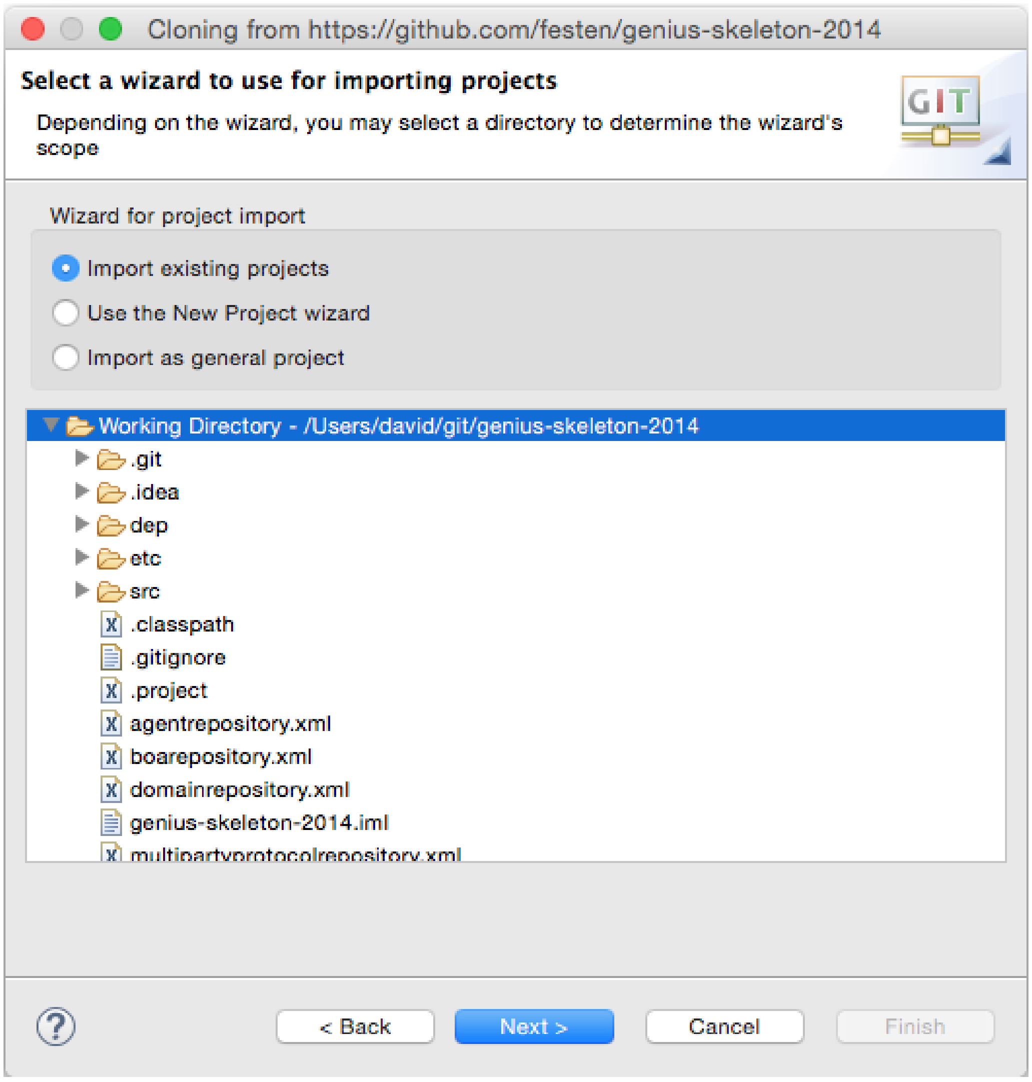
Initial branch: master

Clone submodules [ ]

Configuration

Remote name: origin

Figure 7: How to run genius - Part 6
NB: You are free to choose the location of the directory,
mine is located at `/Users/david/git`*, you are probably not David ;).*


Cloning from https://github.com/festen/genius-skeleton-2014

Select a wizard to use for importing projects

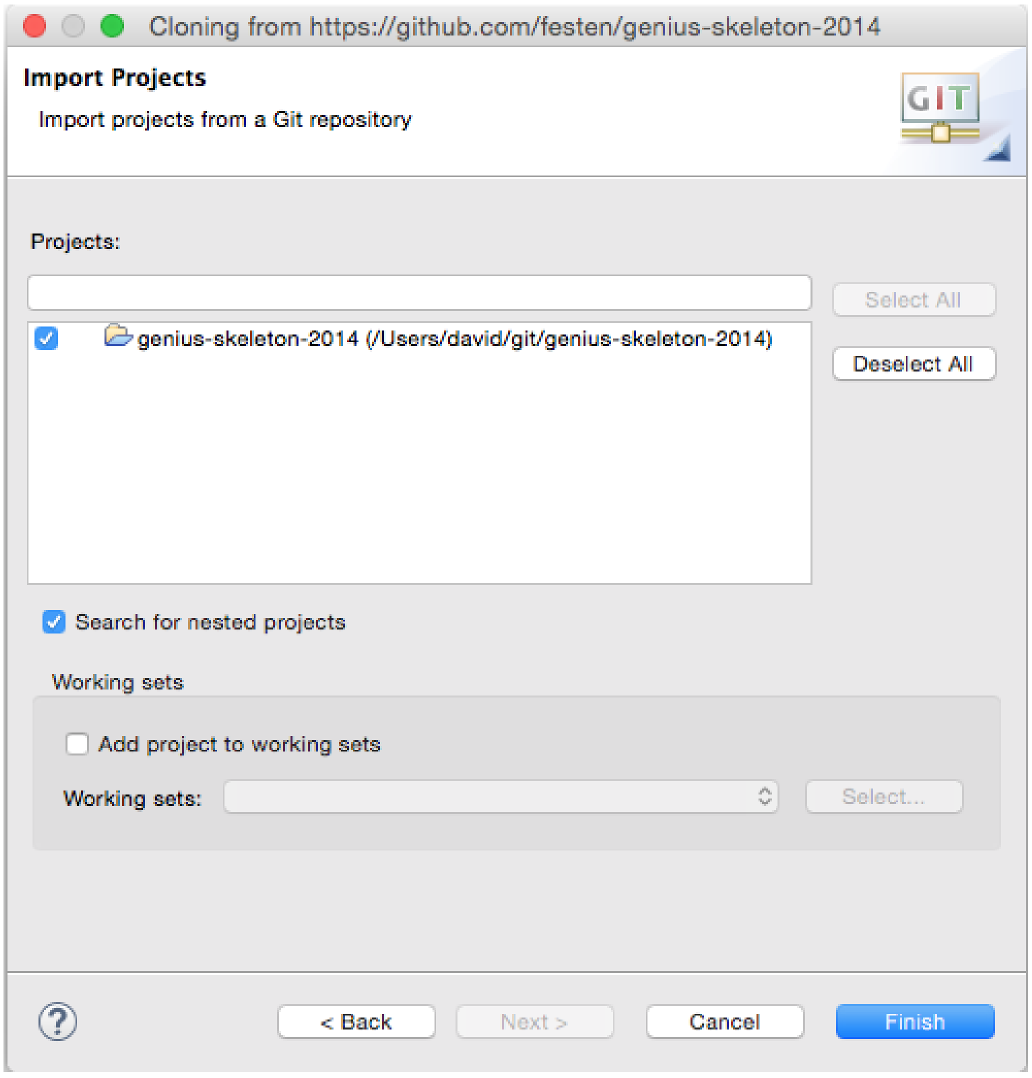
Depending on the wizard, you may select a directory to determine the wizard's scope

Wizard for project import

[✓ Import existing projects]
Use the New Project wizard
Import as general project

```
Working Directory - /Users/david/git/genius-skeleton-2014
  .git
  .idea
  dep
  etc
  src
  .classpath
  .gitignore
  .project
  agentrepository.xml
  boarepository.xml
  domainrepository.xml
  genius-skeleton-2014.iml
  multipartyprotocolrepository.xml
```

Figure 8: How to run genius - Part 7


Cloning from https://github.com/festen/genius-skeleton-2014

Import Projects

Import projects from a Git repository

Projects:

[✓] genius-skeleton-2014 (/Users/david/git/genius-skeleton-2014)

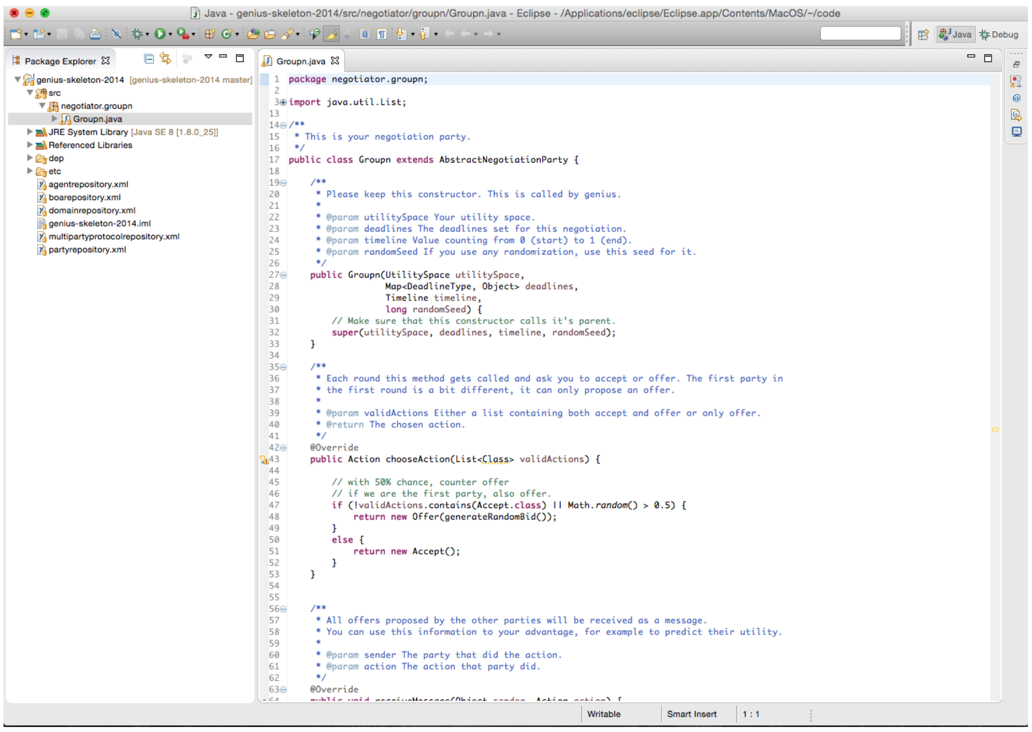
[✓] Search for nested projects

Working sets

[ ] Add project to working sets

Working sets:

Figure 9: How to run genius - Part 8


Java - genius-skeleton-2014/src/negotiator/groupn/Groupn.java - Eclipse - /Applications/eclipse/Eclipse.app/Contents/MacOS/~/code
Java
Debug
Package Explorer
genius-skeleton-2014 [genius-skeleton-2014 master]
src
negotiator.groupn
Groupn.java
JRE System Library [Java SE 8 [1.8.0_25]]
Referenced Libraries
dep
etc
agentrepository.xml
boarepository.xml
domainrepository.xml
genius-skeleton-2014.iml
multipartyprotocolrepository.xml
partyrepository.xml
Groupn.java
Writable
Smart Insert
1 : 1

```java
package negotiator.groupn;

import java.util.List;

/**
 * This is your negotiation party.
 */
public class Groupn extends AbstractNegotiationParty {

    /**
     * Please keep this constructor. This is called by genius.
     *
     * @param utilitySpace Your utility space.
     * @param deadlines The deadlines set for this negotiation.
     * @param timeline Value counting from 0 (start) to 1 (end).
     * @param randomSeed If you use any randomization, use this seed for it.
     */
    public Groupn(UtilitySpace utilitySpace,
                  Map<DeadlineType, Object> deadlines,
                  Timeline timeline,
                  long randomSeed) {
        // Make sure that this constructor calls it's parent.
        super(utilitySpace, deadlines, timeline, randomSeed);
    }

    /**
     * Each round this method gets called and ask you to accept or offer. The first party in
     * the first round is a bit different, it can only propose an offer.
     *
     * @param validActions Either a list containing both accept and offer or only offer.
     * @return The chosen action.
     */
    @Override
    public Action chooseAction(List<Class> validActions) {

        // with 50% chance, counter offer
        // if we are the first party, also offer.
        if (!validActions.contains(Accept.class) || Math.random() > 0.5) {
            return new Offer(generateRandomBid());
        }
        else {
            return new Accept();
        }
    }


    /**
     * All offers proposed by the other parties will be received as a message.
     * You can use this information to your advantage, for example to predict their utility.
     *
     * @param sender The party that did the action.
     * @param action The action that party did.
     */
    @Override
```

Figure 10: How to run genius - Part 9

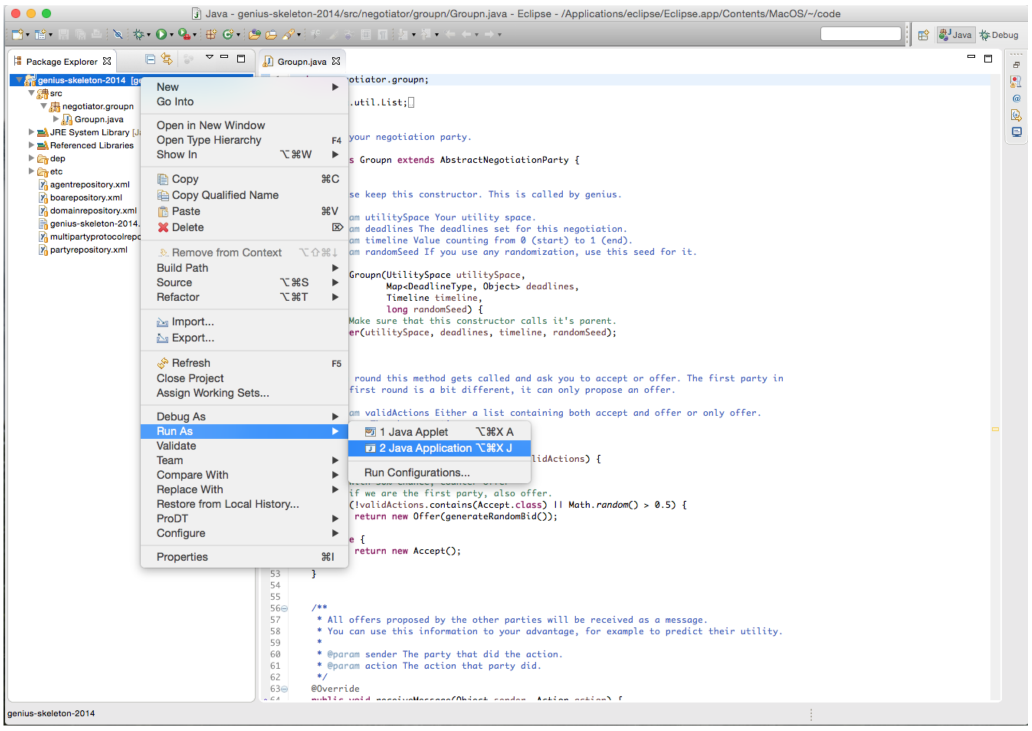


Figure 11: How to run genius - Part 10

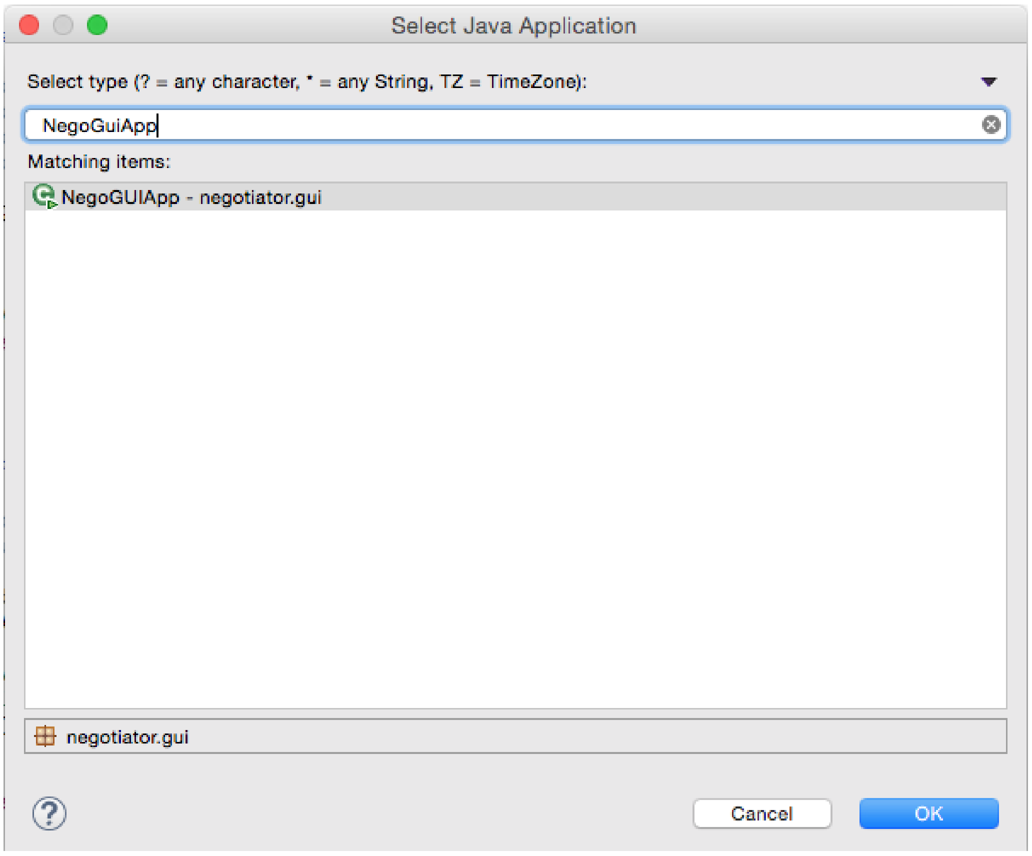

Select Java Application
Select type (? = any character, * = any String, TZ = TimeZone):
NegoGuiApp
Matching items:
NegoGUIApp - negotiator.gui
negotiator.gui
Cancel
OK

Figure 12: How to run genius - Part 11

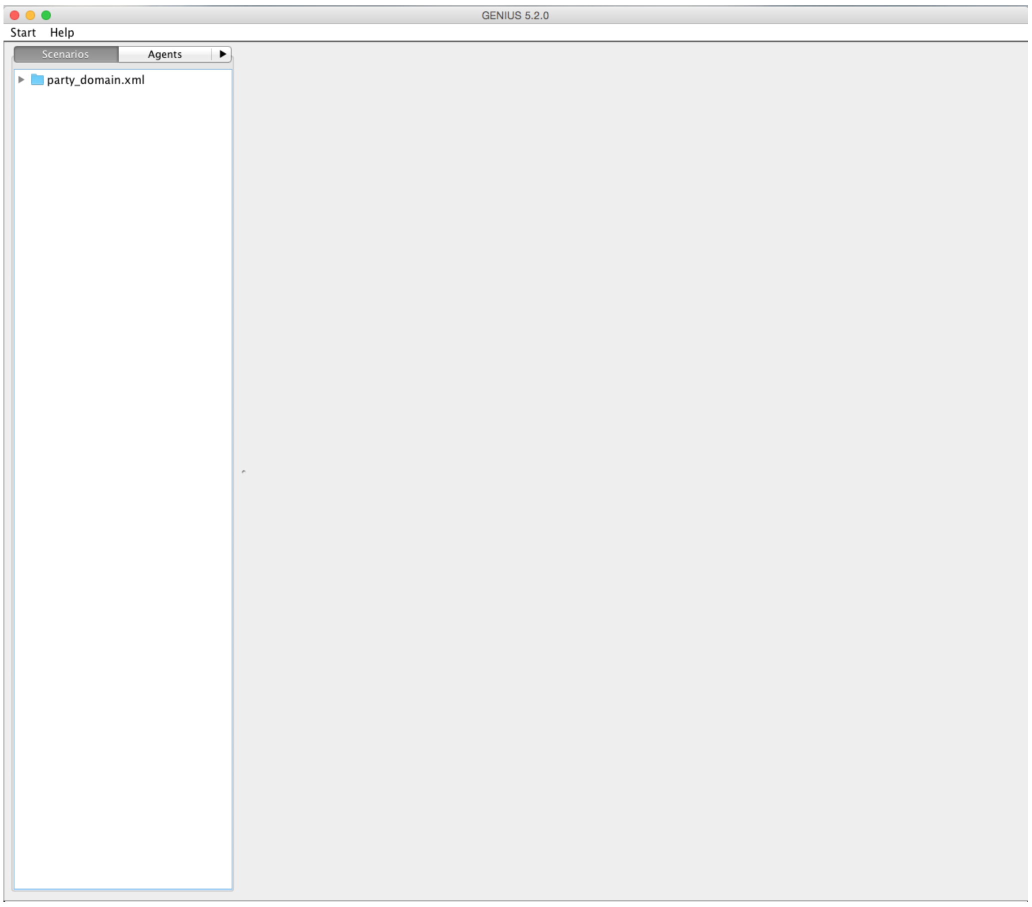
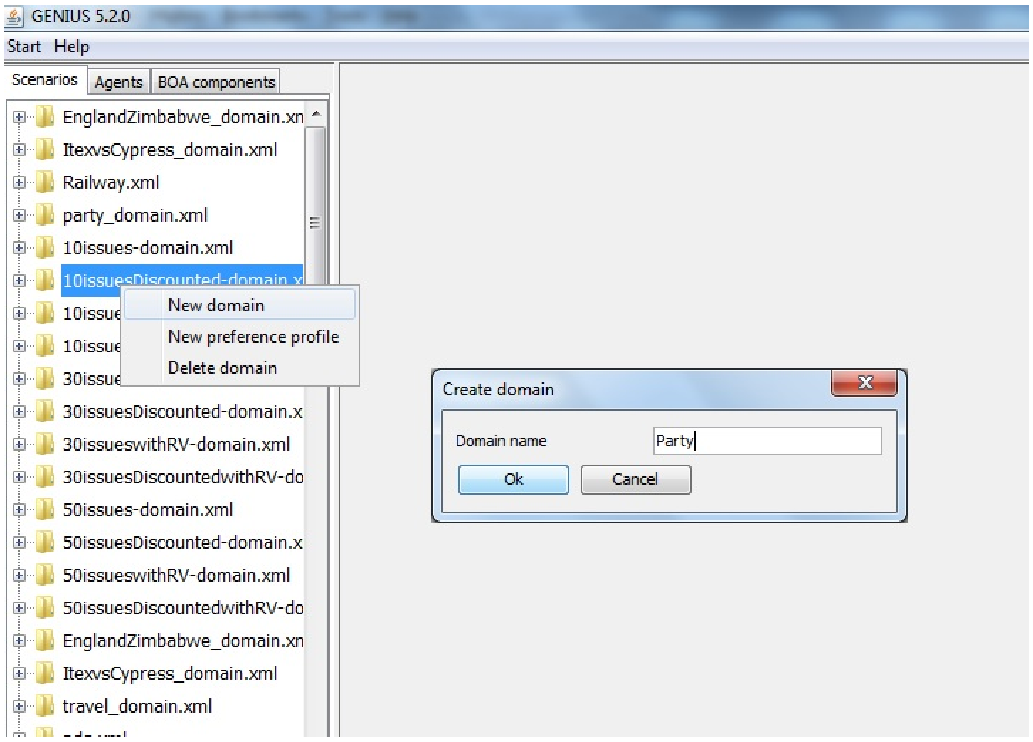
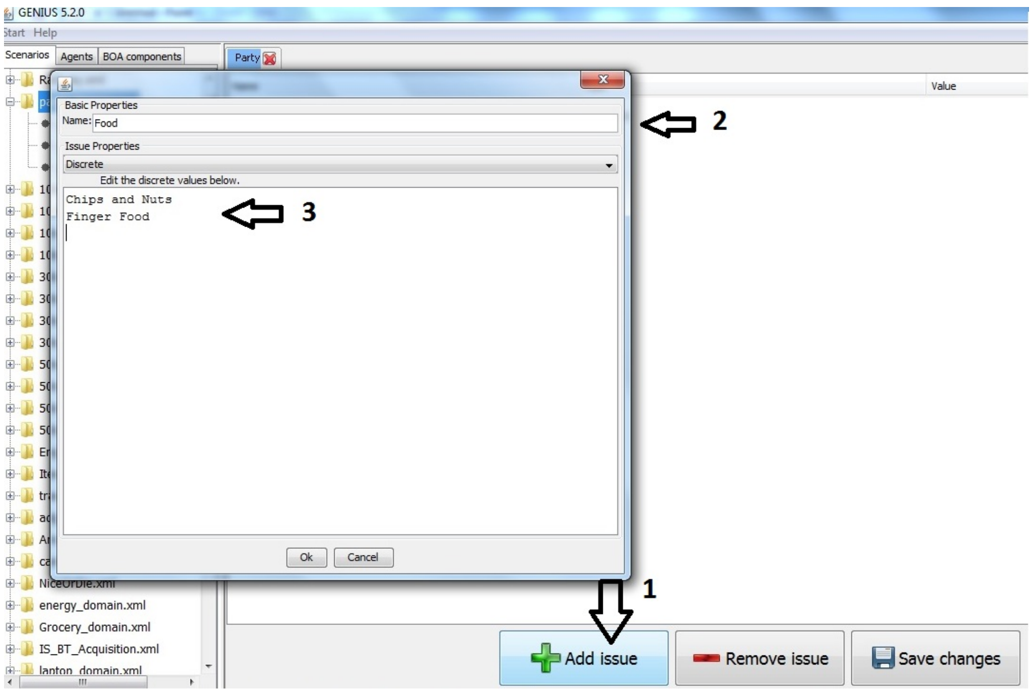


Figure 13: How to run genius - Part 12 (done!)



Domain name: Party

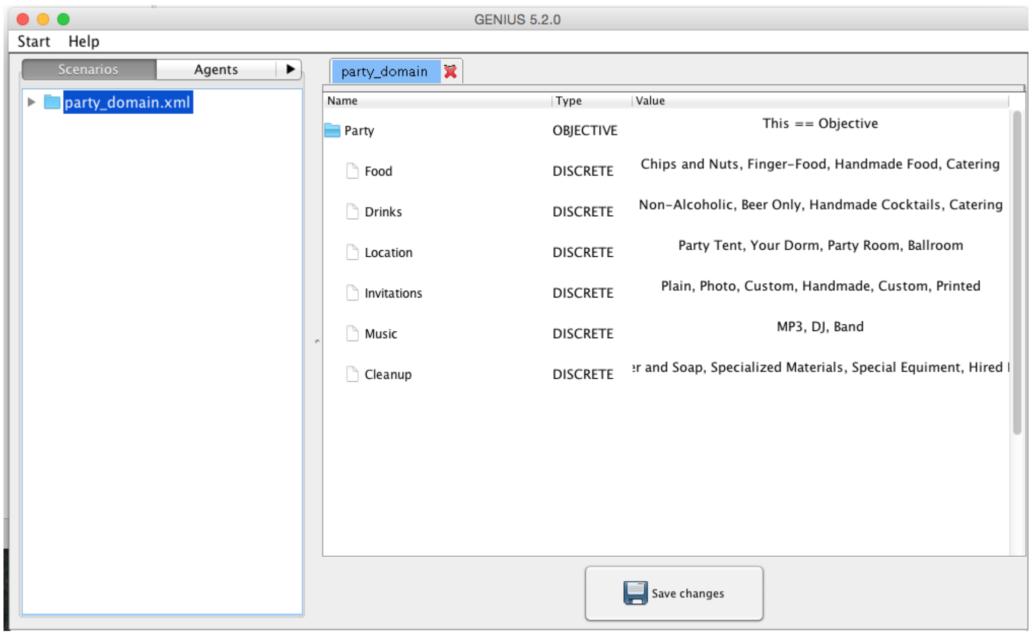
Figure 14: How to create a negotiation domain - Part 1



Basic Properties
Name: Food
Issue Properties
Discrete
Edit the discrete values below.

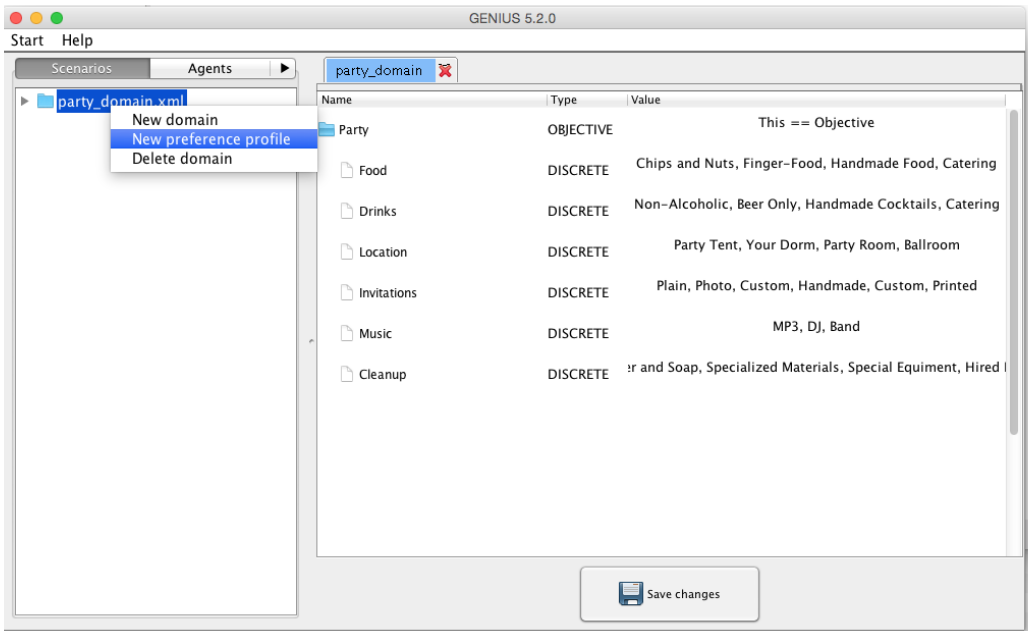
```
Chips and Nuts
Finger Food
```

Figure 15: How to create a negotiation domain - Part 2



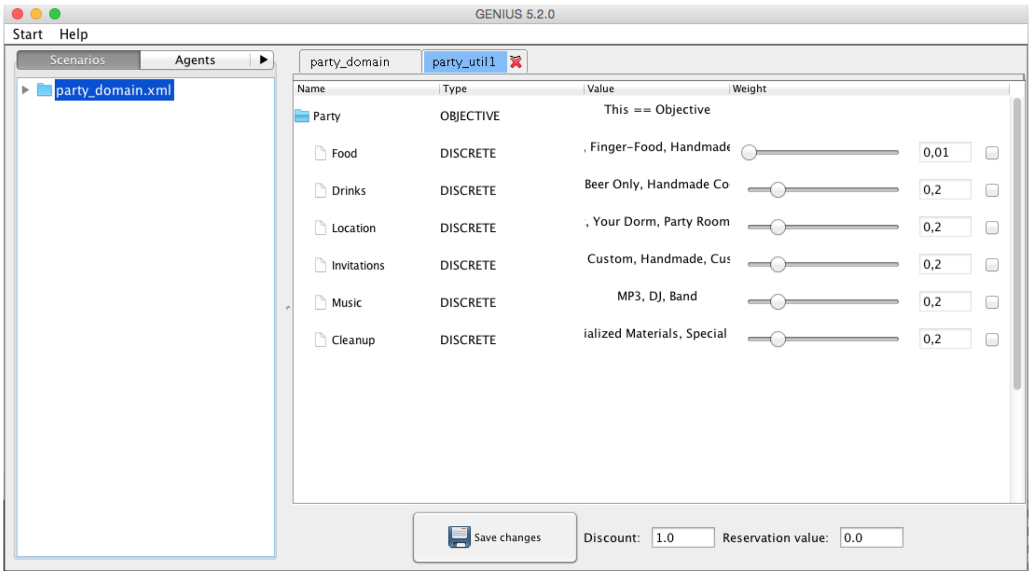
Name	Type	Value
Party	OBJECTIVE	This == Objective
Food	DISCRETE	Chips and Nuts, Finger-Food, Handmade Food, Catering
Drinks	DISCRETE	Non-Alcoholic, Beer Only, Handmade Cocktails, Catering
Location	DISCRETE	Party Tent, Your Dorm, Party Room, Ballroom
Invitations	DISCRETE	Plain, Photo, Custom, Handmade, Custom, Printed
Music	DISCRETE	MP3, DJ, Band
Cleanup	DISCRETE	ːr and Soap, Specialized Materials, Special Equiment, Hired I

Figure 16: How to create a new preference profile - Part 1



Name	Type	Value
Party	OBJECTIVE	This == Objective
Food	DISCRETE	Chips and Nuts, Finger-Food, Handmade Food, Catering
Drinks	DISCRETE	Non-Alcoholic, Beer Only, Handmade Cocktails, Catering
Location	DISCRETE	Party Tent, Your Dorm, Party Room, Ballroom
Invitations	DISCRETE	Plain, Photo, Custom, Handmade, Custom, Printed
Music	DISCRETE	MP3, DJ, Band
Cleanup	DISCRETE	ːr and Soap, Specialized Materials, Special Equiment, Hired I

Figure 17: How to create a new preference profile - Part 2



Name	Type	Value	Weight
Party	OBJECTIVE	This == Objective	
Food	DISCRETE	, Finger–Food, Handmade	0,01
Drinks	DISCRETE	Beer Only, Handmade Co	0,2
Location	DISCRETE	, Your Dorm, Party Room	0,2
Invitations	DISCRETE	Custom, Handmade, Cus	0,2
Music	DISCRETE	MP3, DJ, Band	0,2
Cleanup	DISCRETE	ialized Materials, Special	0,2

Discount: 1.0 Reservation value: 0.0

Figure 18: How to create a new preference profile - Part 3