GeniusWeb Tutorial

Wouter Pasman, Pradeep K. Murukannaiah, and Catholijn M. Jonker

Interactive Intelligence, Delft University of Technology

September 23, 2021

This tutorial will help you getting started on the Negotiation Assignment. It gives some exercises to test yourself, guides you through some steps like setting up parts of Eclipse, starting services, etc. This tutorial is optional, it is provided to help you, and not part of an assignment.

Sections 1 to 4 introduce the concepts of domains, issues, preference profiles, and outcome spaces as well as the GeniusWeb platform. In Sections 5 to 9, you will create a simple negotiation party in Java[1] using the GeniusWeb platform and learn to debug your party. Sections 10 to 14 introduce a more advanced strategy using opponent modelling, and shows how to evaluate your party via different metrics.

A note on terminology: We use the term negotiation *party*. In other literature you may find the term *agent*.

1 Understanding Preference Profiles

We start with a few simple exercises to make sure that you understand the basic concepts. Please refer to the Negotiating Agents handbook (available in Brightspace) to understood the theoretical concepts.

Exercise 1 *Given are the following linear-additive profiles for a buyer A (Table 1) and a seller B (Table 2)), calculate the utility of outcome o=(Price: €1200, HardDisk: 1TB, Monitor: 15) for Parties A and B.*

Table 1: Preference profile of Party A (buyer)

Issue	**Weight**	**Evaluations**
Price	0.3	eval(€1000)=1, eval(€1200)=0.5, eval(€1400)=0
Hard disk	0.4	eval(256GB)=0, eval(512GB)=0.6, eval(1TB)=1
Monitor	0.3	eval(15)=0, eval(17)=1

Table 2: Preference profile of Party B (seller)

Issue	**Weight**	**Evaluations**
Price	0.6	eval(€1000)=0, eval(€1200)=0.5, eval(€1400)=1
Hard disk	0.3	eval(256GB)=1, eval(512GB)=2/3, eval(1TB)=1/3
Monitor	0.1	eval(15)=1, eval(17)=0

Exercise 2 *Figure 1 plots the utilities of parties A and B corresponding to the outcomes from Example 1. Which of these outcomes are Pareto optimal?*

[1] Other languages can be used, but we chose the reference language used for GeniusWeb

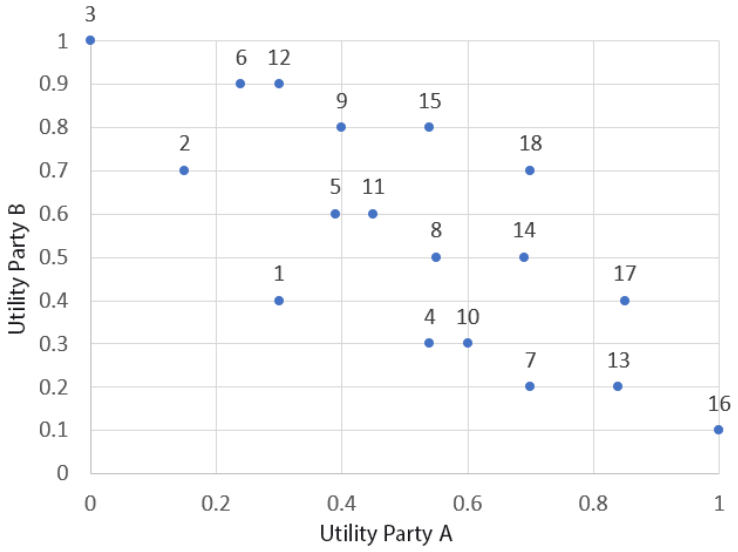

Figure 1: Negotiation outcomes for the laptop domain.

2 Installing GeniusWeb

For creating a profile, you only need a text editor, possibly with json support. You can install or use an already available text editor on your machine.

For creating a party and to run the geniusweb servers, you need to have java JDK (at least version 8) and maven to be installed on your machine.

To run a negotiation, you need the GeniusWeb servers installed in a tomcat server running on your machine. Go to `https://tracinsy.ewi.tudelft.nl/pubtrac/GeniusWeb#Installation`and follow the instructions to install the latest version of the GeniusWeb servers on your machine. You need these servers to run your own parties and profies in a negotiation. There is also a video on the installation process, `https://ii.tudelft.nl/GeniusWeb/students.html`.

Start the tomcat webserver that you have after the installation. Try the runserver GUI to check that your GeniusWeb is working.

3 Creating a Domain in GeniusWeb

We are now going to create the laptop profiles as in Table 1 and Table 2 above. If you go to the Profileserver GUI you can check the already available profiles. You should get a table like in figure 2.

You can see that there already is a laptop domain available. We are going to create a new **laptop1** domain and leave the existing laptop domain intact.

ProfilesServer messages: At this point it is useful to keep an eye on the tomcat log messages. You can see those inside the logs/catalina.out file inside your tomcat installation directory.

3.1 Copy laptop folder

The quickest way to get started is to copy the existing laptop domain and modify it.

1. locate and check the contents of the directory webapps/profilesserver-X.Y.Z/domainsrepo inside your tomcat application (NOTICE: this is created by tomcat from the profilesserver-X.Y.Z.war file when you start up tomcat).

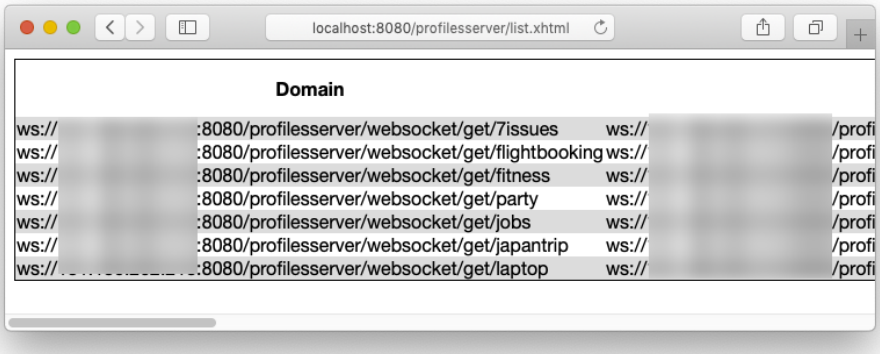

Figure 2: Default profiles on the profilesserver.

2. copy the laptop folder including its contents to laptop1.
3. at this point, you may see a message on the profilesserver: `WARNING domainsrepo/laptop1/laptop1.json is missing`.
4. to fix this change the new `laptop1/laptop` filename into `laptop1.json`. You will now get another error, which will be fixed in the next section.

3.2 Fix domain description

We are now first going to fix the domain description. For reference, domain desciptions are documented on GeniusWeb domains.

Open the file `laptop1.json` and make changes. Make sure that your text editor keeps the UTF8 text encoding and does not mess with the end-of-line markers. Do the following:

- the "name" should be changed to "laptop1".
- the "Brand" issue has to be changed into "Price".
- the "Price" values have to be changed into the correct values "600", "900" and "1200". Keep the quotes, See valueset documentation for more details.
- Fix the harddisk values.
- fix the issue "External Monitor" to "Monitor".
- fix the values of "Monitor".
- save the file.

At this point, you still get warnings in the catalina log, but now errors like "Profile has incorrect domain", indicating that domain desciption inside the laptopBuyer and laptopSeller files does not match the laptop1 description.

3.3 Fix the profiles

We now are going to fix the two profiles. The following has to be repeated for both the laptopBuyer and the laptopSeller files inside the laptop1 domain.

1. Open the `laptopBuyer.json` file with the plain text editor.

2. Replace the entire contents of the "domain" section with the full contents of the `laptop1.json` file that you just fixed.
3. If you would save the file at this point, you would get a warning that "The issues in utilityspace and domain do not match" because the issues in the domain now do not match issues in the profile itself.
4. Now we are going to fix the actual profile inside the LinearAdditiveUtilitySpace element:
   - as with the profile, fix "Brand" to "Price", and "External Monitor" to "Monitor".
   - in Price/valueUtilities change the values from Brand to Price values "600", "900", "1200". Make sure that your values match exactly the values that you set in the domain, including quotes. WARNING: if you do not copy the issue names correctly you will not get an error but your profile will get a 0 evaluation for the wrong-typed issue, leading to unexpected behaviour of your profile.
   - Set the correct evaluation values, as in the perference profile of this party.
   - repeat for the Harddisk issue. You can round 1/3 to 0.33 etc.
   - repeat for the Monitor issue.
   - fix the issue names in the issueWeights, "Brand" to "Price", etc.
   - Finally, fix the issueWeights, according to the weights of the issues in the table.
5. save the file.

Repeat this for the `laptopSeller.json` file. If you did all fixes correctly, there should be no new errors reported, and your profiles should now be visible on the profilesserver as in Figure 3.

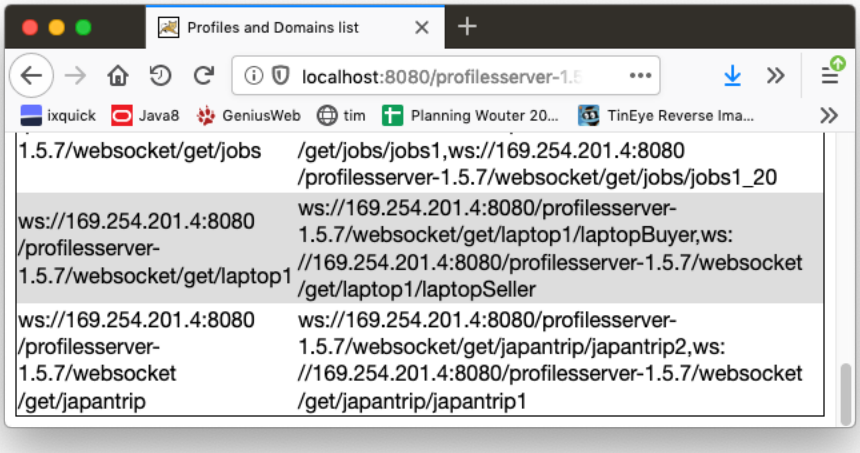

Figure 3: laptop1 domain and profiles available on profilesserver.

4 Running a Negotiation Session

We are going to use the web interface to run some automated negotiation. Browse to your local runserver (see the wiki pages) and select 'new session'.

- use the default SAOP protocol. You can read more on the other protocols on tracinsy.ewi.tudelft.nl/pubtrac/GeniusWeb#Protocol

- set 60 rounds for the negotiation.
- Select the laptop1 domain that you created.
- Select e.g. boulware and laptopBuyer for the settings and click Add.
- Select e.g. linear and laptopSeller for the settings and click Add.

Your setup should look like Figure 4 now.

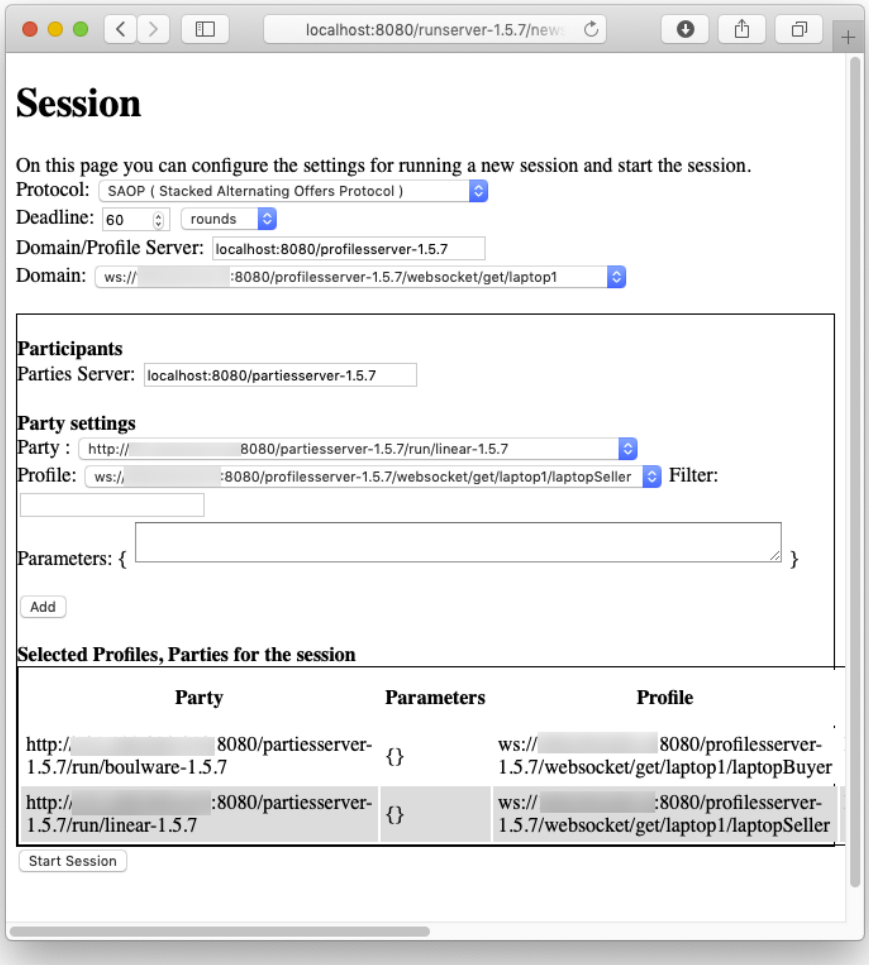

Figure 4: Session setup, ready to run

Rounds vs Time: The Deadline can either be set to ROUNDS or TIME. With TIME, the negotiation has a time deadline, at which point the negotiation will be terminated if no deal was yet reached. With ROUNDS, the negotiation is terminated if no deal is reached after the given number of rounds, or if the negotiation lasts longer than a maximum duration (typically 10 seconds).

Click on the "Start Session" button. After a few seconds, the text appears below the "Start Session" button. You can click on "view the log file" to see the raw log file results from the negotiation. Alternatively you can render a graph of the results by clicking on "render a utilities plot". Clicking the latter, you get a window as in Figure 5 with the bidding progress and informing that a deal has been reached.

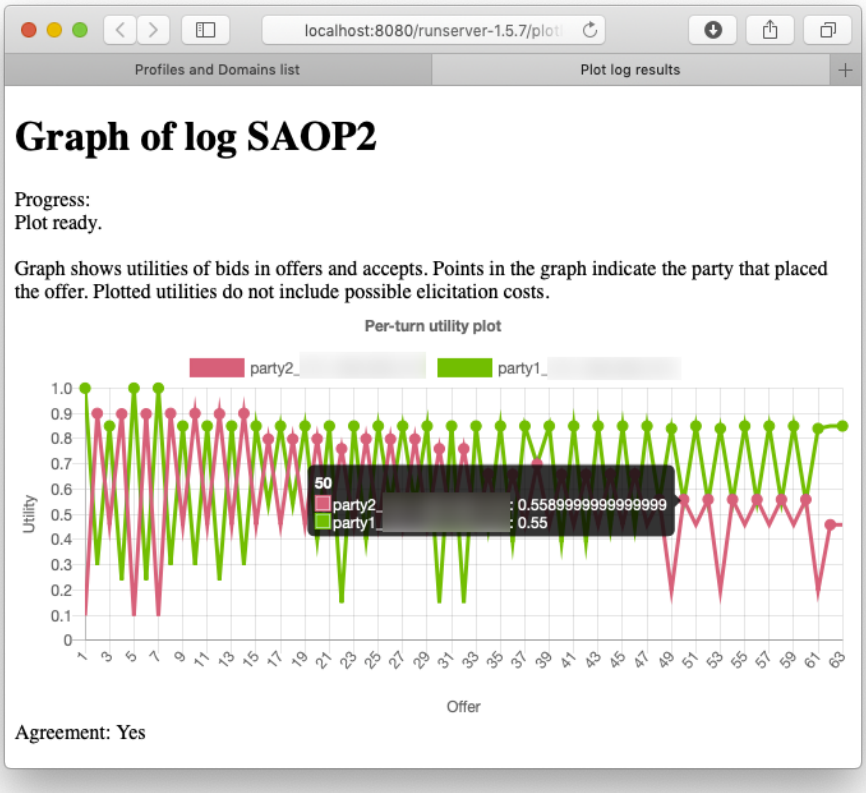

Figure 5: Session progress plot

In addition to the negotiation log visible in the GeniusWeb interface, this log can be found in the directory `webapps/runserver/log` inside your tomcat application.

5 Creating a party in Eclipse

In this section, you are going to write and compile your own party in Java.

Install Eclipse if you don't yet have it. You can download it from www.eclipse.org/downloads/packages. You need the IDE for java Developers.

Java, Maven, Tomcat, Eclipse: It is assumed you are familiar with the Java programming language, the Maven build system, Tomcat server and Eclipse. The tutorial videos show how to do a number of tricky steps and should help you to do the tasks in this exercise.

Integrated Development Environment (IDE): It is highly recommended that you use a Java IDE, such as Eclipse or IntelliJ, for developing your party. If you are already experienced with Java, then you can use any IDE you are most comfortable with. Otherwise, it is recommended that you use Eclipse since this is the one used in most of the examples and the tutorial videos.
With the Eclipse IDE, you get many tools, features and support, such as all code documentation immediately available, also from all third-party libraries, code completion suggestions, refactoring, immediate highlighting of syntax and typing erorrs, interactive debugging, automatic building, testing, quality assurance, etc. This is not the place to delve into these, but we recommend that you explore these features for yourself if you are not familiar with these.

We follow the instructions on the GeniusWeb wiki. There are Tutorial videos available for this as well. Follow the one explaining how to build your party in Eclipse. We recommend following the video tutorial if you are not used to Eclipse. :

1. Create a new workspace in Eclipse (or use a space you already made).
2. Install an SVN client, eg subclipse, if you did not yet install it, using Eclipse Marketplace menu item.
3. Clone the randomparty from the GeniusWeb example parties repository.
4. If Eclipse does not recognise it as Maven, configure the project as maven project in Eclipse.
5. Change the isGood function of RandomParty.java so that it accepts above utility 0.7 instead of 0.6.
6. Change the getDescription function accordingly.
7. run `maven clean package` and check that your party passes the tests and that a jar file is built in the target directory.

At this point, your compiled party will have the name `randomparty-X,Y.Z.jar`.

RandomParty supports several protocols. If you are working on an assignment, typically you need to support only one protocol and you can remove code for the other protocols.

6 Renaming your party

Assignments typically require you to use specific package names, and not use the packages already used by GeniusWeb. To rename your party:

1. In the Package Explorer, Open src/main/java of the randomparty project.
2. Right-click the geniusweb.exampleparties.randomparty folder and select refactor/rename.
3. Enter the new name for your package, e.g., `collabai.group`N where N is your group number. Check your assignment for the required name.
4. repeat for the src/test folder
5. Open the new package.
6. Right-click on RandomParty.java, select refactor/rename.

7. Enter your new partyname, e.g., `GroupNParty` where N is your group number.
8. repeat for the src/test folder
9. Open the pom.xml of the project.
10. At the top, change:
    - `<artifactId>randomparty</artifactId>` into `<artifactId>group`N`party</artifactId>`
    - `<groupId>geniusweb.exampleparties</groupId>` into `<artifactId>collabai.group`N`</arti`
11. Fix the two occurences of `<mainClass>geniusweb....RandomParty</mainClass>` into the correct full class path to your party.
12. Re-run the `mvn clean package`.

At this point, there should be a `group`N`party-X,Y,Z-jar-with-dependencies.jar` inside your target directory. You may need to right click and refresh the directory.

7 Deploy your party

Deploying your new party is done by just copying your `...jar-with-dependencies.jar` file to partiesserver.

1. Open `webapps/partiesserver-X.Y.Z/partiesrepo` on the tomcat server.
2. Copy your `group`N`party-X,Y,Z-jar-with-dependencies.jar` to this directory.
3. Assuming your server is still running, check if the party is available: open the URL of the local partyserver to see if your party is available. You should see your party in the list, as in Figure 6.
4. If it is not available, check the catalina.out log file to see reported errors.

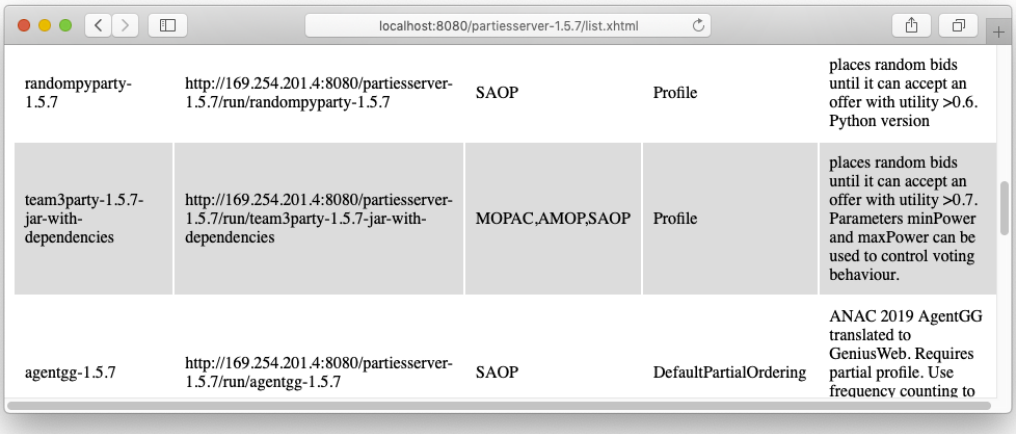

Figure 6: My agent running successfully

There is also a Tutorial video available for this. Your party is now ready for use. You can now use your party just as you used the boulware and linear parties in Section 4.

8 Debugging your party

Sometimes your code is not behaving as you expected. One way to find the issue is to use a debugger. To use the debugger directly on the party as it runs on the server is possible but a bit impractical, check the wiki if you really want to take this approach.

Instead, here we use the recommended way using the standalone runner for debugging.

Follow the steps outlined on the wiki page. A video on how to debug standalone is available on the Tutorial page.

In short, you should do the following:

1. Install the standalone runner in your workspace.
2. Edit the run settings of the runner, so that it runs your party in the correct configuration. Also increase the deadline so that you have ample time to debug before the party is killed
3. Place breakpoint in your party.
4. Run the standalone runner in eclipse debug mode.
5. Eclipse will switch to the debugger and you can inspect all variables and step through your program.

The session run will usually show "complete successfully" even if a party crashes. This is because a crashing party is considered just a protocol error.

You can also use the standalone runner to do quick standalone runs, avoiding the need to set up a session or tournament for each run.

If the run terminates, the final State (here, a `SAOPState`) is printed to stdout (usually the console). This state contains lots of details and may also contain a full stacktrace of failing parties.

9 Modifying your Party

Documentations All classes in GeniusWeb are heavily documented to help using them. An overview of GeniusWeb is given on the main wiki page.
You can also download all javadocs from all modules. For example, to get the 2.0.0 bidspace documentations, go to `http://artifactory.ewi.tudelft.nl/artifactory/webapp/#/artifacts/browse/simple/General/libs-release-local/geniusweb/bidspace/2.0.0/bidspace-2.0.0-javadoc.jar` and download and unzip the `javadoc.jar`. Open the index.html file inside. If you use Eclipse, this is all done automatically Eclipse shows the javadoc for anything you hover over.
All source code from GeniusWeb is publicly available, check the information on the main wiki page.

9.1 Using the Profile

Your party receives the profile that it has to negotiate with in the Settings object. The settings object usually is the first object that your party receives.

The Settings object contains only an URL from where the profile can be fetched. A simple method is to use the ProfileConnectionFactory for this. It gives you a ProfileInterface that does the fetching and offers you an instance of a `Profile`, ready for further use.

Here we discuss an `LinearAdditiveUtilitySpace`, as you already made one in Section 1. This space can assign a number in $[0, 1]$ to each bid, and the higher the number the higher the preference for that bid. This space works using linear, weighted functions as was explained in the handbook.

The following code accesses the domain and displays the issues, etc. Try and understand each line of code, using the Javadoc as a reference for the various methods.

Copy and paste the code snippet inside notifyChange(..), at the end of the if(info instanceof Settings) block. Run a single negotiation session with your agent and check the console for the output produced.

```
LinearAdditiveUtilitySpace space =
 (LinearAdditiveUtilitySpace) profileint.getProfile();
Map<String, ValueSetUtilities> valueutils = space.getUtilities();
for (String issue : space.getDomain().getIssues()) {
    System.out.println(">> " + issue + " weight: "
    + space.getWeight(issue));
    ValueSetUtilities utils = valueutils.get(issue);
    // ignore the non-discrete in this demo
    if (!(utils instanceof DiscreteValueSetUtilities))
        continue;
    Map<DiscreteValue, BigDecimal> values =
         ((DiscreteValueSetUtilities) utils).getUtilities();
    for (DiscreteValue value : values.keySet()) {
        System.out.println("utility( " + value + ")= "
        + values.get(value));
    }
}
```

You will get 'missing' and 'cannot be resolved' errors once you have copied the code since you need to import several classes. In Eclipse you can do this easily, by placing the cursor at the end of the class name where the error is, and then pressing CTRL+Space (you can use this for code completion). This will give you several options, and typically choose the first one. Pressing enter will import the corresponding class.

it is recommended to delay access to the profile as long as possible, to avoid waiting for server data to arrive while you don't yet need it.

This code is placed inside the Settings handler only for demonstration purposes. Avoid printing debug info in your assignment solutions.

9.2 A Simple Concession Strategy

Let's now modify the negotiation strategy. The SAOP protocol consists of 3 possible actions:

- Accepting the opponent's offer;
- Generating a (counter) offer (which automatically means rejecting an opponent offer);
- Ending the negotiation.

In the SAOP protocol, parties decide for an action when they receive a `YourTurn`. Others' actions received through the `ActionDone` object can be stored for later use.

If the party is the first to get the turn, then the first action should be to generate an `Offer`. Before proceeding any further, have a look at how these two methods are implemented in your party and make sure you understand the given code.

Next, we introduce a simple concession strategy. The concession strategy determines the target utility level at which to produce and/or accept offers.

1. Use a simple linear approach to set the target utility, which starts with the highest possible utility (i.e. the utility of the best possible offer), and then concedes up to a threshold when the time limit is reached. Set this threshold using a variable (e.g. it can be the average between the highest and lowest possible utility within the set of possible offers). Some code to help you is provided below:

   Getting a bid close to the maximum utility:

```
private Bid getMaxUtilityBid() throws IOException {
    BidsWithUtility bwu = new BidsWithUtility(
        (LinearAdditive) this.profileint.getProfile());
    BigDecimal maxutil = bwu.getRange().getMax();
    ImmutableList<Bid> bids = bwu.getBids(new Interval(
```

```
6           maxutil.subtract(new BigDecimal("0.01")), maxutil));
7       if (bids.size().compareTo(BigInteger.ZERO) == 0)
8           return null;
9       return bids.get(BigInteger.ZERO);
10  }
```

The above code uses the BidsWithUtility library to do the 'heavy lifting' of processing the utility space. Of course you could naively iterate through the bids to find the exact maximum, but this might take longer than the available time for the negotiation.

We use the bidspace.BidsWithUtility library to demonstrate how to cope with potentially huge spaces. Check the java documentations with BidsWithUtility for more details. Also check the other classes in bidspace.

Accessing the current progress of the negotiation, where 0 means the start and 1 the end of the negotiation:

```
Double t=progress.get(System.currentTimeMillis());
```

2. Accept any offers received by the opponent which have a utility equal or greater than the target utility.
3. If no such offer was done, select a random offer which has a minimum utility threshold and offer it to the opponent.

Check the makeOffer code. Note that the used approach is not very efficient. Think about how this could be improved, also given that you now have a target utility..

9.3 Running Tournaments

Running individual negotiation sessions for testing and evaluating your party can be a tedious task. There are two ways to easen running multiple sessions easier: running a tournament and using a script.

A tournament consists of multiple negotiation sessions, involving multiple parties and multiple profiles. The GeniusWeb APP tournament protocol will automatically generate all possible configurations and you can even repeat tournaments multiple times, all with 1 click of a button. For example, suppose you want to compare 3 different strategies and 2 different profiles, then this will generate 3*3=9 individual negotiation sessions.

To see how this works, first run a tournament using the GeniusWeb runserver interface by selecting `new tournament` (see runserver wiki if you need more detailed instructions). Select individual parties, e.g., Conceder, Boulware and your party. Furthermore, select at least 2 profiles from e.g. the Party domain. Run the tournament and see what happens.

Also you can run sessions and tournaments from a script. The starting point is a json file with the sessionsettings or tournament settings. By providing this file to the simplerunner or sending it to the runserver, you can run these settings locally or on the server without having to navigate any GUIs.

Note that, when running a tournament, there is no record of the individual bids exchanged (unlike in the case of single negotiation sessions). This is to avoid large files, bogging the system and deterioration of the parties' performance.

Now you are going to run a tournament using the simplerunner from Eclipse or from the command line, without using the GUI.

1. Edit the simplerunner pom, and add dependencies for boulware and conceder, so that we can use those in the simplerunner. boulware has this maven dependency:

```
<dependency>
    <groupId>geniusweb.exampleparties</groupId>
    <artifactId>boulware</artifactId>
    <version>${geniusweb.version}</version>
</dependency>
```

and conceder has this maven dependency:

```
<dependency>
    <groupId>geniusweb.exampleparties</groupId>
    <artifactId>conceder</artifactId>
    <version>${geniusweb.version}</version>
</dependency>
```

2. Edit the file `src/test/resources/tournament.json`:
   - Change the first partyref to `geniusweb.exampleparties.boulware.Boulware`.
   - Change the second partyref to `geniusweb.exampleparties.conceder.Conceder`.
   - Change the third partyref to the classpath of your team.
3. Edit the NegoRunner run settings and change the argument to `src/test/resources/tournament.json`.
4. Run the simplerunner to run your tournament.
5. Analyze the `AllPermutationsState` object that was finally printed to the console and check that there are 18 session results. Explain why 18. Check how many agreements were reached.

Instead of running from Eclipse, you can run from the command line. The above mostly applies, but instead of fixing Eclipse settings, do

1. run `mvn clean package`
2. run (of course fix to the correct simplerunner version) `java -jar target/simplerunner-2.0.0-jar-with-dependencies.jar src/test/resources/tournament.json`

Many editors can re-format JSON text which makes it easier to read. Even browsers like Firefox can do this: just drop an unformatted json file into it and you will get a pretty-printed, collapsable presentation.

It is recommended that you browse the details of the GeniusWeb wiki page which contains much more details that may come in handy for your following tasks.

You are now ready to start creating your own negotiation strategy!

10 Opponent Modeling

The negotiation strategy described earlier is very simple and does not result in very Pareto efficient agreements. A typical approach to improve this, and which is used in many negotiation strategies, is to estimate the profile of the other party. This process is called "Opponent Modeling". A properly estimated profile allows then for instance to generate closer to pareto-optimal bids and to determine the concession strategy used by the other(s).

One way to do this is to use the bids received from the other parties. The `FrequencyOpponentModel` assumes that others will use preferred issue values more often than less-preferred issue values. Thus, counting how often an opponent uses the issue values gives a way to assess these bids.

To do just that, do the following steps:

1. Check the FrequencyOpponentModel already available in GeniusWeb.
2. Notice that this model is immutable. Explain how you can update this object to include a new offer received from a participant.
3. Add code to your party, that keeps an up-to-date FrequencyOpponentModel for all the participants (you don't need to model yourself).
4. Check that your approach works by running your party with another party and displaying the resulting opponent model.

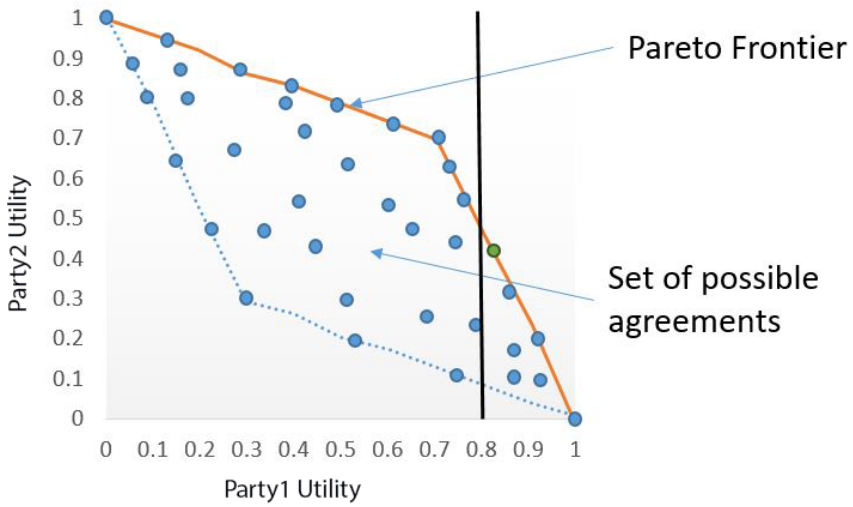

Figure 7: Finding a Pareto efficient point.

5. The FrequencyOpponentModel is also a UtiltySpace and thus implements getUtility. Explain how this function works and if that matches the mechanism described above.

6. Add code to your party that prints received bids and the estimated opponent utility yourself for some bids. Explain the results.

7. Check whether the estimated opponent profile converges to the actual opponent's model. How is this affected by different types of parties (eg Conceder versus RandomParty versus your own party)?

11 Improve the Offer Generating Strategy

We are now going to improve the simple offer generating strategy from the previous lab by using our opponent model. The aim is to obtain Pareto efficient offers.

Consider the setting in Figure 7. Suppose that Party 1 is making an offer, and the current target utility is 0.8. That is, it is willing to generate and accept offers with utility 0.8 or more. This is indicated by the black vertical line in Figure 7. Note that all the offers to the right of the line meet this threshold. Now, in order to find a Pareto efficient offer, all the party needs to do is to find an offer which meets the threshold and has the highest utility for Party 2. If the opponent model is perfect, this is necessarily a Pareto efficient offer. In the example, this would be the green point on the frontier. Obviously, in our case, the opponent model is not necessarily correct, and so the best one can hope for is that, by using this approach, the offer still is approximately Pareto efficient.

To implement this approach, there are at least 2 options:

1. Loop through all the bids, e.g. use bidspace.GenericPareto, to collect those with a utility equal or better than the target utility for the party, and then choose the one from that set which has highest utility for the opponent according to the opponent model. Realize that iterating through all the bids may already take longer than the total time available in the negotiation.

2. A quicker but heuristic approach is to test random bids until a bid is found above a threshold. But instead of selecting only 1 random bid, search multiple bids and choose the one with the highest opponent utility. Explain why this may still take very long to find a decent bid.

You may find even better solutions to use for your own party.

Answers to a few exercises

- Exercise 1: Utilities in Outcome ID 8 in Table 3.
- Exercise 2: The Pareto optimal outcomes are are Outcome IDs 3, 12, 15, 18, 17, and 16.

Table 3: Negotiation outcomes and utilities for the laptop domain.

Outcome ID	**Price**	**HardDisk**	**Monitor**	**Utility A**	**Utility B**
1	1000	256GB	15	0.3	0.4
2	1200	256GB	15	0.15	0.7
3	1400	256GB	15	0	1
4	1000	512GB	15	0.54	0.3
5	1200	512GB	15	0.39	0.6
6	1400	512GB	15	0.24	0.9
7	1000	1TB	15	0.7	0.2
8	1200	1TB	15	0.55	0.5
9	1400	1TB	15	0.4	0.8
10	1000	256GB	17	0.6	0.3
11	1200	256GB	17	0.45	0.6
12	1400	256GB	17	0.3	0.9
13	1000	512GB	17	0.84	0.2
14	1200	512GB	17	0.69	0.5
15	1400	512GB	17	0.54	0.8
16	1000	1TB	17	1	0.1
17	1200	1TB	17	0.85	0.4
18	1400	1TB	17	0.7	0.7