Negotiating Agents

Koen Hindriks

November 22, 2006

This is a new assignment published for the first time this year, so it may still contain mistakes and/or typos. If you notice anything that needs improvement from typos to more serious flaws in this assignment document, please let us know. You can use either the Discussion Board of the Course on Blackboard to report such observations, or mail directly to Koen Hindriks.

1 Introduction

This document describes one of the two assignments that can be chosen to complete a practical assignment for the second quarter of the AI course. This assignment involves the design and implementation of a negotiating agent. The practical assignments aim at familiarizing students with parts of the course material in a more practical way. Techniques relevant for building and information about designing negotiating agents can be found among others in chapters 16 and 17 in [8]. The assignment will also require you to go beyond the course material in the book.

The objective of this assignment is to implement a *negotiation strategy* for an agent in order to get the best deal in a negotiation with an arbitrary opponent. Automated negotiation has become more and more important in recent years because of, among other things, the Internet. Market places such as e-Bay and marktplaats.nl require automated support for making deals between users. Negotiation Support Systems (NSSs) are designed to support human decision making in negotiations. Negotiation has been traditionally analyzed from a game theoretic point of view, but the recent increase in interest in this subject requires the development of new techniques using techniques from Artificial Intelligence as a starting point.

The negotiation setting in this assignment is quite simple. All that is known from the start is that there are two agents, two different sets of preferences about one or more issues, and a turn-taking protocol to exchange offers. Additionally, an agent may accept an offer or may break off a negotiation. Both of these actions terminate the negotiation. The first action, accepting an offer, results in an agreement called a *deal*. In that case, each agent is assigned a score determined by the quantitative value associated with the deal by the agent's *utility function*. The second action, breaking off a negotiation, results in a score of 0 points. Building a good negotiating agent in this simple setting is hard. Though it is very easy to stick to the rules of the game (it is not even possible to violate them), it does not mean that it is easy to negotiate well! The main reasons that explain why this is so are that agents have *incomplete information* and that even when both agents would have complete information it is not a priori clear what a *fair outcome* would be.

Negotiation between two agents can in many ways be modeled as a game, and game theory is useful to analyze the behavior of negotiating agents. As in most games, the rules of engagement during a negotiation are clear from the start. In this assignment, we will assume, for example, that the negotiating agents take turns. Negotiation is, however, also different from many board games such as chess and reversi. One of the most important differences is that negotiation as we will study it in this practical assignment never is a zero-sum game. That is, a typical negotiation does not have a winner who takes all and a looser who gets nothing. In order to start a negotiation, it is only reasonable for both parties to believe that there is a win-win situation where both agents can gain by obtaining a deal through negotiation. Another difference is that the domain of negotiation may be quite different from negotiation to negotiation. In

this assignment, however, the focus will not be on domain-specific elements, but - as is the case in most board games as well - abstract from most real-life details and concentrate on the most essential elements that return in every negotiation.

To clarify the remarks about negotiation domains, we add some remarks about the *process* of negotiation. Typically, negotiation viewed as a process is 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 techniques for modeling such domains are not considered and you may assume that the prenegotiation phase has been completed. In other words, you may assume that both the issues to be negotiated and the associated issue values are provided to you before negotiation starts. There are two activities that are usually considered part of the prenegotiation phase that you do need to solve in this assignment. The first is the setting of so-called *reservation values* for individual issues or for the negotiation outcome as such. These values specify which bids you will never accept. The second is that your agent may decide to adapt its strategy based on negotiation outcomes of previous sessions (or even during a negotiation session). But the most important part of this assignment concerns the *negotiation phase* itself, i.e. the exchange of offers between your agent and its opponent.

This assignment requires you to build a negotiating agent of your own using the techniques introduced in [8]. You will find that again the theme of search will be important in this assignment (as in complete information games such as reversi). It will become quickly clear, however, that you will also have to go beyond the ideas presented in [8] and incorporate other techniques that will further improve your agent's negotiation strength. Two of the main problems to design a negotiating agent is to find a good negotiation strategy and to find a good decision function for accepting an offer from an opponent. That is, the two most important problems for you to solve in this assignment are to think of ways to respond to your opponent and to decide on when an offer is good enough.

There is a lot of literature available about negotiation that may help you finish this assignment successfully. As for almost any subject, you can also find more information about negotiation strategies on the Internet. For example, a brief definition of what negotiation is can be found on Wikipedia [4]. [7] provides a good introduction on negotiation. Other references that may provide a good starting point are [1, 3, 5]. See for additional references the bibliography at the end of the assignment.

Finally, to end this introduction, in the next table we provide some initial guidelines based on human experience with negotiation that may help you in building your own software negotiating agent.

Orient yourself towards a win-win approach.
Plan and have a concrete strategy.
Know your reservation value.
Create options for mutual gain.
Generate a variety of possibilities before deciding what to do.
Pay a lot of attention to the flow of negotiation.

2 Detailed Assignment Description

The assignment may be completed in teams of 2-4 students. In the sections below the objectives, deliverables, requirements, submitting of the solution, and the detailed assignment description are presented.

2.1 Objectives

- To learn to design a negotiating agent, 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 name to identify the team and the negotiating agent.
- A negotiating agent programmed in JAVA using the negotiation environment provided to you.
  - A package containing your agent code. It is obligatory to use "negotiation.group" + ♯your_agent for the negotiating agent and any other required files.
- Your (programmed) solution for computing and picturing the outcome space (including Pareto Optimal Frontier).
- A report documenting and explaining the solution, including an explanation and discussion of:
  - The main methods used in the negotiating agent and implemented in the source code,

  - The negotiation strategy and the factors it takes into account, as well as the decision function for accepting an offer,
  - Any preparatory steps the agent takes at the beginning of a negotiation session,
  - Arguments why your agent is really aiming at the best negotiation outcome possible.

The report need not be lengthy (10 a4 pages may be enough), but should include an explaination 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.

2.3 Requirements

- The agent should implement a negotiation strategy to generate offers, and a decision function to decide whether to accept an offer or not.
- The agent must 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 opponent).
- The agent meets reasonable time constraints and is able to complete a negotiation session within 2 minutes. Any agent that uses more than 2 minutes in order to initiate negotiation or to reply to an opponent's offer will be disqualified.
- The source code has to be written in Java 1.5. Moreover, it is mandatory to use the negotiation environment provided to you.
- The source code should contain explanatory comments that will allow third parties not involved in programming the code to understand it. A clear explanation of a method must be provided at the beginning of every method. Parts of the code that are not self-explaining should receive additional documentation. Please incorporate enough information in comments in the code to ensure this.
- The agent implementation should be tested to ensure that it works on multiple systems, and requires nothing other than the files contained in your group's package. Integrating your agent into the negotiation environment should not be more difficult than adding your package in the right folder (which needs to comply with the directory structure: "negotiation.group♯.YourAgentHere".
- The report should include:
  - 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.
- answers to the questions posed in the assignment, and
- 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) negotations performed by humans.

Please remove all debug printouts from your code, and make sure that your agent does not flood the 'System.out' channel.

Note that the report should help the reader to understand the organization of the source code (details should be commented on in the source code itself).

2.4 Submitting Assignment Solution

Please submit your report in PDF format and the package for your negotiating agent by using the website http://dublin.twi.tudelft.nl/webapp/. Use the naming conventions described elsewhere in this document, including your group number. *Your actual group number is determined by the number you received while registering on http://dublin.twi.tudelft.nl/webapp/ for the Practical. Make sure you use this number to submit your deliverable.* Do not send your completed assignment by email to ai@mmi.tudelft.nl. Do not submit incomplete assignment solutions; only a complete assignment solution containing all deliverables will be accepted. The deadline for submitting the assignment is January 12, 2007 at 11:59pm.

2.5 Assignment

In this section, the main questions you need to answer are presented. But before we do this, we present additional background that is required for completing the assignment successfully. Please also review some additional constraints and assumptions summarized at the end of this document. These constraints and assumptions may also help you to clarify the content and structure of the assignment.

A negotiation in this assignment may consist of multiple *negotiation sessions*. In each session, two agents negotiating with each other need to settle a conflict and negotiate a deal. Each negotiation session is limited by a fixed amount of time; in this practical assignment it is set to 2 minutes. At the end of each 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 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. The agent that starts a negotiation is determined randomly for each negotiation session. In the tournament that will be played, each agent will negotiate 5 sessions with all other agents and the scores of each *final session* are recorded and averaged to obtain an overall score for the agent (see also below). A ranking will be compiled using these overall scores (and a bonus point will be assigned to the group with the highest ranking).

The main reason for multiple negotiation sessions is that agents may learn about their opponent from the sequence of bids exchanged and from the negotiation outcome (e.g. deal or no deal). Exchanging offers allows the parties to learn about the others' limitations, and to identify the key issues and critical areas of disagreement. During this phase, the parties realize the potential of a compromise and can assess its main features. The analysis of a negotiation may focus on the modification of strategies, the determination of concessions, and on the restriction of efficient solutions to those which may be acceptable to the parties. A second reason for having multiple sessions (where in the tournament only the last session will be scored) is that it will sometimes be useful to break off the negotiation to signal dissatisfaction about the way negotiation proceeds to the other agent.

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 choosing a type of restaurant both agents may not agree about their first choice but may agree on their second choice. They could, however, be in serious conflict over the location (e.g. city) where they would like to have diner. 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 both agents do better (i.e. they both score higher 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 of the outcome for the first agent times that for the second agent is maximal (cf. [9]).

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

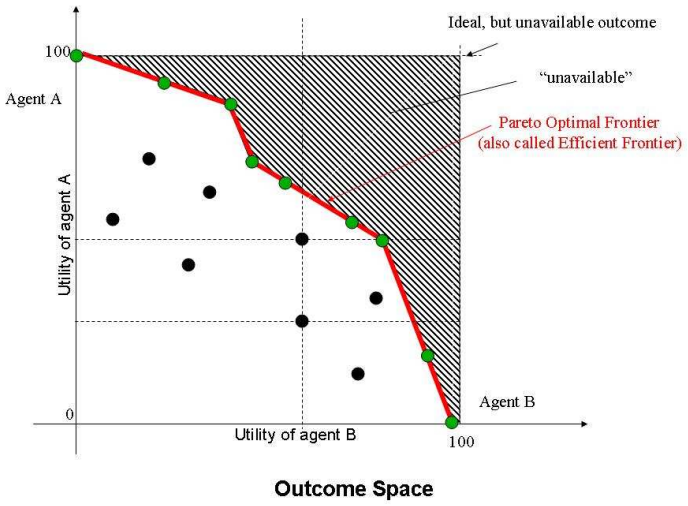

Figure 1: Pareto Frontier

an offer that is most favorite 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, but other strategies are conceivable and may be necessary given the deadlines (cf. [2]).

To compute whether a bid in a negotiation is Pareto optimal or a Nash solution we use so-called *utility functions* (cf. also [8]). In our case, utility functions assign quantative 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 4 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 $w_1, \ldots, w_4$) can be calculated by:

$$u(i_1, i_2, i_3, i_4) = w_1 * u_1(i_1) + w_2 * u_2(i_2) + w_3 * u_3(i_3) + w_4 * u_4(i_4)$$

The outcome space of a negotiation, i.e. all possible bids, can be plotted on a graph which indicates the utility of both agents on the X and Y axes respectively. An example is provided in Figure 1.

What sets the negotiations considered in this assignment apart from other negotiations is the fact that both agents have exactly the same deadline for achieving a deal and both agents "know" this. In the negotiation environment provided to you, any negotiation session between two agent is limited to take at most two minutes in total.

The assignment consists of two parts. The first part concerns so-called *fair division problems* ([6]). The second concerns so-called *multi-issue negotation*. A fair division problem can be viewed as a simple instance of the more general class of multi-issue negotiations with multiple values. Fair division problems thus provide for a more simple setting to think about and study the basic negotiation concepts and techniques than general multi-issue negotiation with multiple values. In the second part the complexity of issues with multiple values will need to be dealt with. The examples used in the assignment are based on [6].

Before we introduce the two parts of the assignment, you are asked to answer the following questions.

1. Create a PEAS description (in the same format of a table as in cf. [8]) of the negotiating agent that you need to design and implement in this assignment. Be as complete as possible and pay special attention to the performance measure of the agent. Also include a brief discussion about each element included in your PEAS description. In particular, explain informally what goals the agent has in a negotiation.

2. Design the overall structure and components of your agent. Describe the structure of your agent and the components it is consists of.

2.5.1 Fair Division Problems

A fair division problem concerns the division of a number of items among two (or more) agents. Typical real-life examples of such negotiations are the division of an inheritance. Each agent that takes part in such a division problem has its own preferences about obtaining some item. Some of these may be similar to other agents. For example, two agents may both prefer most to obtain the antique chinese vase. Since we only consider bilateral agents in this assignment we will only look at the two-party case. In all negotiations that we will consider you may assume that agents need to settle multiple issues.

3. Inspect the negotiation template named `inheritance.xml` and utility templates `inheritance_Janet.xml` and `inheritance_Marty.xml`. Compute the utilities for all the possible bids in the outcome space and plot them on a graph such as Figure 1. Draw the efficient frontier. Also compute the Nash solution. Submit your solution for computing and picturing the outcome space with your report.

4. Analyze the performance of the simple agent `MyAgent` in the folder `agentExample` provided to you with the negotiation environment on the negotiation tem-

plate `inheritance.xml`. Load the negotiation template into the negotiation environment. This will automatically select and load agent `MyAgent` for both agent A and agent B into the negotiation environment, as well as load the two utility profiles. Start a negotiation session. Is a Pareto optimal outcome reached? Explain your answer. Also explain why it obtains the resulting outcome that it does.

5. Design and implement a negotiation strategy.

   (a) Decide on the way your agent will set a *reservation value*. Explain in the report informally how your agent determines its reservation value and describe how you implemented this in your agent. In case your agent also uses other considerations to determine whether it will never accept particular bids, also explain these considerations.

   (b) Design a *decision function* which your agent uses to determine whether it will accept a bid or not. Describe which factors the agent takes into account for making this decision and how you implemented the decision function.

   (c) Decide on the way your agent will compute a *counter offer* (a follow-up bid). Explain in the report informally how your agent computes a counter offer. List all considerations that your agent takes into account explicitly and explain why you have chosen to base the agent's decision on these considerations.

   (d) Does your agent ever propose offers that *increase* the utility of the offer again compared with previous offers made? If not, demonstrate that the utility associated with bids in any bid sequence your agent will ever propose in a negotiation monotonically decreases. If it does, explain when and why it will decide to do this.

6. Experiment: Negotiate against own agent.

   (a) Assume there are only two items (or issues), called *blue* and *yellow*, that need to be divided between two agents. Create a negotiation template called `two_issue.xml` with just these two issues by modifying another template. The values of these issues should be Janet (for agent A) and Marty (for agent B). Create a utility template called `Janet_two_issue.xml` which associates a utility of 70 with issue *blue* if Janet gets it and 0 if Marty gets it, and a utility of 40 with issue *yellow* if Janet gets it and 0 if Marty gets it. Create a second utility template called `Marty_two_issue.xml` which associates a utility of 50 with issue *blue* if Mart gets it and 0 if Janet gets it, and a utility of 80 with issue *yellow* if Mart gets it and 0 if Janet gets it. Change the agent name to your agent's name and the utility space labels in the negotiation template accordingly. Load the negotiation template into the negotiation environment. Run five negotiation sessions in which your software agent negotiates with itself using each

a different profile though. Report the outcome. Explain why the outcome is as it is. Is any of the outcomes a Nash solution? Is it possible to reach a non-efficient outcome, i.e. an outcome that does not lie on the Pareto Frontier?

(b) Change the negotiation template `inheritance.xml` such that instead of `MyAgent` it loads your agent. Call this template `inheritance`+your agent name+`.xml`. Analyze the performance of the agent by running at least five negotiation sessions. Is a Pareto optimal outcome reached? Explain your answer. Also explain why it obtains the resulting outcome that it does.

2.5.2 Negotiating Multiple Issues with Multiple Values

7. Inspect the negotiation template named `AMPOvsCity.xml` and utility templates `AMPO.xml` and `City.xml`. Compute and draw the efficient frontier in a graph such as Figure 1. Also compute the Nash solution. Your (programmed) solution should be submitted with your report.

8. Analyze the performance of the simple agent `MyAgent` in the folder `agentExample` provided to you with the negotiation environment on the negotiation template `AMPOvsCity.xml`. Load the negotiation template into the negotiation environment. Start a negotiation session. Is a Pareto optimal outcome reached? Explain your answer. Also explain why it obtains the resulting outcome that it does.

9. Design and implement a negotiation strategy.

   (a) Adapt the functions that you implemented in your negotiating agent such that it can handle multiple values for multiple issues. That is, modify the way your agent determines a reservation value, adapt its strategy and its decision function for accepting to handle multiple values for issues.

   (b) Run your adapted agent again on the inheritence example. That is, using your new agent, answer questions 6b and 6a again. Does your new agent reach the same outcomes? If not, analyze the differences and explain why these differences result.

10. Play against own agent.

    (a) Assume there are only two items (or issues), called *typeOfFood* and *location* about which two agents need to negotiate to fix particular values. Inspect the negotiation template called `two_issue_mv.xml` that contains just these two issues. You can inspect the utility values associated with the values of these issues in the files `Janet_two_issue_mv.xml` and `Mart_two_issue_mv.xml`. Change the agent's name to your agent's name in the negotiation template. Load the negotiation template into the negotiation environment. Run five negotiation sessions

in which your software agent negotiates with itself using each a different profile though. Report the outcome. Explain why the outcome is as it is. Is any of the outcomes a Nash solution? Is it possible to reach a non-efficient outcome, i.e. an outcome that does not lie on the Pareto Frontier?

(b) Change the negotiation template `AMPOvsCity.xml` such that instead of `MyAgent` it loads your agent. Call this template `AMPOvsCity`+your agent name+`.xml`. Analyze the performance of the agent by running at least five negotiation sessions. Is a Pareto optimal outcome reached? Explain your answer. Also explain why it obtains the resulting outcome that it does.

2.5.3 Concluding: Future Perspectives

10. As you will have noticed by now, it is hard to find deals that are Pareto optimal. An alternative for one-to-one negotiation as used in this practical assignment is to use a trusted mediator to which each agent declares its private preferences. The mediator then will compute an efficient outcome using this input. Do you think that introducing a mediator is a better option for finding an optimal outcome for both agents? If so, argue why you think so and describe the circumstances in which it is particularly suitable to use a mediator. If not, argue why you think it is not useful to do this.

11. The agents in the negotiation environment have limited capabilities in order to achieve a negotiated deal. Think about capabilities (e.g. other actions or forms of communication) that an agent might be extended with. Can you think of additional capabilities that would help an agent to achieve a better deal? If so, explain why these capabilities would help an agent to achieve a better deal. If not, argue why there will not be any capabilities that could help an agent achieve a better deal.

As a final remark, we want to make clear that it is very important in order to improve the negotiation strength of your agent and to identify potential methods to improve it to extensively *test your agent*. A warning is in place though: Do not assume that your goal is to outperform a "stupid" agent such as the one provided with the negotiation environment. A good outcome is determined by other criteria, identified by efficiency of the outcome (i.e. is it close or not to the Pareto Frontier). To test your agent in this regard, it will in particular be useful to test your agent on various negotiation templates. You can create these templates both by modeling real-life examples of negotiation or create them randomly. You can run your agent against itself, or, for example, against the simple agent provided to you with the negotiation environment or against earlier versions of your own agent. You may also agree to run it against an agent of another team. In general, there are many techniques to improve the negotiation strenght of agents. Various factors can be taken into

account when deciding on acceptance, breaking off, or computing a next offer in a negotiation, e.g.: Time an opponent takes to reply with a counter offer, consistency of an opponent's offers, concession rate of your opponent, history of previous negotiation sessions, and possibly other considerations about the domain of negotiation itself. Of course, also use the suggestions and literature references provided to you elsewhere in this document.

3 Implementation

3.1 What is provided to you as a start?

The following file is provided to complete the assignment, which can be downloaded from Blackboard:

- The installer file `negotiator.zip` containing:
  - This assignment file called `negotiation.pdf`,
  - The negotiation environment implemented in JAVA, including an example agent `negotiation.group0.SimpleAgent` ,and an agent that provides a user interface for manual input of an action, called `negotiator.UIAgent`,
  - An example negotiating agent JAVA file, called `SimpleAgent.java`, located in the subfolder *negotiation/group*0,
  - A class diagram of the negotiation environment called `negotiator_class_diagram.bmp`,
  - A set of negotiation templates in directory `templates`.

In case you need more information about JAVA programming, please use the following link: http://java.sun.com/docs/books/tutorial/index.html. *You must use JAVA 1.6 to run the environment and build your agent.* You can download the JDK 6 Release Candidate from: http://java.sun.com/javase/downloads/ea.jsp.

3.2 Negotiation Environment

The negotiation environment can be started by:

```
java -cp negotiator.jar;. negotiator.Main
```

or using the batch file `run.bat` on a Windows machine.

The screen that pops up when the negotiation environment is started allows you to:

- Load a negotiation template. A negotiation template also automatically loads two agents (one playing the role of agent A and the other playing the role of agent B) and two utility templates associated with agent A and agent B.
- Load two agents, one playing the role of agent A and the other playing the role of agent B,
- Load a utility template for agent A and one for agent B,

- Start a negotiation between two agents, after a negotiation template has been loaded.

3.2.1 Adding an Agent

Loading an agent can be done by specifying the full name of the agent, prefixed by its package name. If you are in group NR, and your agent is called MyAgent, and it resides in package "negotiation.groupNR", you can add it by entering "negotiation.MyAgent" in the corresponding text field. *Your actual group number is determined by the number you received while registering on http://dublin.twi.tudelft.nl/webapp/ for the Practical. Make sure you use this number to submit your deliverable.*

3.2.2 Starting a Negotation

You can start a negotiation between any two available agents by loading a negotiation template and pushing the start button. The start button is located on the bottom left side of the interface. In the same window you can see the log of the exchanges between the agents. The outcome of a negotiation will be shown in a separate window called *Negotiation results*. The `negotiator.UIAgent` is a special agent that is not bothered by any time restrictions, and allows users to negotiate against a software agent (note the different format and exact spelling that you have to use for this agent; you can also incorporate this agent in your negotiation templates). It is not possible to select who will make the first offer in a negotiation, this is randomly allocated.

3.2.3 Negotiation Multiple Sessions

In order to put your agent to a test, it is interesting to see how it performs against a number of other agents in a number of negotiation sessions. The negotiation environment does not provide a special tournament mode, but does allow you to set the number of sessions that two agents are supposed to play against each other. A history over multiple sessions may be maintained by either agent in such a run.

3.3 Implementation instructions

In the subfolder *negotiation* all required classes for the negotiation environment can be found.

A negotiating agent will receive an object of class `negotiator.actions.Action` through method `ReceiveMessage(Action opponentAction)` . It has to reason about the best next move, and return an object of one of the `negotiator.actions.Offer`, `negotiator.actions.Accept`, `negotiator.actions.EndNegotiation` classes. The `negotiator.NegotiationTemplate` class contains all the necessary information about the current negotiation state. It contains the object of class `negotiator.Domain` that contains information regarding the negotiation issues and their values (cf. also the Class Diagram provided to you).

An important consideration for the implementation is that an agent may participate in multiple negotiation sessions. For this reason, the negotiation enviroment calls a method `init(int sessionNumber, int sessionTotalNumber, NegotiationTemplate nt)` before starting a new session. Parameter `sessionNumber` is a unique number identifying the current negotiation session. Parameter `sessionTotalNumber`— informs you about the total number of the sessions.

An object of class `Domain` represents the structure of the negotiation: issues and their values. You can get a reference on this object using method `getDomain()` of the `NegotiationTemplate` object, which you received in the `init(...)` method of your agent. The most important methods of the `Domain` class include `int getNumberOfIssues()` that returns the number of the negotiation issues and `Issue getIssue(int:index)` that returns a reference to a particular issue according to its index.

Note that a special value called `unspecified` must be distinguished from other values. The value `unspecified` always has a 0 utility value and can be used to communicate *partial offers* to an opponent. That is, by including the value `unspecified` in your bid for some issue, you do not propose any actual value for the issue but leave it open and to be agreed in next rounds of the negotiation. This also requires you to check whether or not your opponent did make an actual bid (proposed a value) for an issue or not.

Class `Issue` declares the structure of a particular issue. Method `int getNumberOfValues()` returns the total number of values of the issue. The values of an issue are represented as `String`. For a convinience, an integer index is assgined to each value. You can get an index of a particular value using method `int getValueIndes(String value)`. To get the value by its index, use method `String getValue(int index)`. Method `String getName()` returns the name of the issue.

Each agent is supplied with a utility space (class `negotiator.UtilitySpace`). This class represent preferences of the agent w.r.t. the issues and their values. Use method `double getEvaluation(int issueIndex, int valueIndex)` to get utility value of a given value of a particular issue. Method `double getWeight(int issueIndex)` returns the weight of a given issue. Method `double getUtility(Bid bid)` calculates the utility of a bid. You can extend the `negotiator.UtilitySpace` class with your own functionality. If you do so, do not forget to implement a method `void loadUtilitySpace(String fileName)` in your agent (see negotiation agent skeleton below).

To instantiate an object of any action you have to use a reference to your own object as the first parameter. Actions `Accept` and `Offer` require also an object of class `Bid` that represents your bid. You must use function `makeBid(int[] valueIndexes)` of the Domain class instead of calling the constructor of the class `Bid`. Parameter `valueIdexes` is an array of the value index for each of the issues of the domain.

Below you can find a skeleton of a negotiating agent:

```
package negotiation.groupXXX;

import negotiator.*;
```

```
import negotiator.actions.*;

public class MyAgent extends Agent{
    // declare here your custom variables and functions

    public MyAgent() {
        super();
        // here you can initialize your variables
        return;
    }
    protected void init(int sessionNumber, int sessionTotalNumber,
                NegotiationTemplate nt) {
        super.init (sessionNumber, sessionTotalNumber, nt);
        // here you can do an initialization of your parameters
        // before each negotiation session
        return;
    }

    public void ReceiveMessage(Action opponentAction) {
        // here you are informed about the action of your opponent
        return;
    }
    public Action chooseAction() {
        // here goes code that chooses the next action and
        // returns it to the enviroment.
        return action;
    }
    public void loadUtilitySpace(String fileName) {
        //in case if you want to extend the UtilitySpace class with
        // your functionality you have to implement this method in
        // the following way:
        utilitySpace = new MyUtilitySpace(getNegotiationTemplate().getDomain(),
                 fileName);
        return;
    }

}
```

An example agent has been included in the installer, and can be found in the subfolder negotiation/group0. This `negotiation.group0.SimpleAgent` can be used as a start for your implementation. In general, this agent will not perform good in a negotiation.

4 Deadline for Submission

All deliverables should be submitted at the latest on January 12. Deliverables should be submitted by using the website http://dublin.twi.tudelft.nl/webapp/ (select *Artificial Intelligence Practical* listed under the heading *Academic Year 2006/2007*).

There is no presentation or demo required since it is not realistic to schedule this between New Year and Examinations.

5 Evaluation

Assignments are evaluated based on several criteria. All 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:

- *Quality of the Deliverables*: Overall explanation and motivation of the design of your negotiating agent; Quality and completeness of answers and explanation provided to the questions posed in the assignment; Explanatory comments in source code, quality of documentation,
- *Performance:* Agents will be ranked according to negotiating strenght that is measured in a tournament (cf. also the next section below),
- *Originality:* Any original features of the agent or solutions to questions posed are evaluated positively,

5.1 Competition

Agents of teams are ranked according to negotiation strength. The ranking will be decided by playing a tournament in which all agents play 5 negotiation sessions against each other. In case there will be many teams choosing this assignment, teams may be split up in two or more pools in which agents play against each other. In that case, a winner will be determined in a second round in which pool winners play against each other. The members of the team that designed the highest ranked negotiating agents in the competition will be awarded with 1 bonus point for their exam out of a maximal score of 10. The exact number of teams that will receive a bonus will depend on the total number of teams that sign up for this assignment and will be announced when this information has been collected.

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 e.g. other students will not be evaluated and be judged as insufficient. Detection of fraude will be reported to the administration.

Negotiation has become a major and interesting research area within Artificial Intelligence. Negotation is important in many domains ranging from business applications such as Internal market places to incident and crisis management. Negotiation is one of the main tools an autonomous agent has to agree about a course of action or to resolve conflicts with other agents. As such, you might be interested in the possibility of doing a Master Thesis about negotiation and there are many angles that you can take, e.g.

- Design of a negotiation support system, either advicing humans in a particular domain, or even taking over negotiation all together,
- Design of a testbed for negotiating agents,
- Design of negotiation strategies, either related to a specific domain or not,
- Research related to negotiation, trust and the reputation of agents, either cognitively motivated or viewed from a more engineering point of view,
- Research on negotiation and culture. How does culture influence negotiation between different agents?
- Etc.

If you are interested, don't hesitate to ask us!

Please report any problems you may have with the software provided or other questions via Blackboard so everyone can learn from each others experiences. Do not sent mail to ai@mmi.tudelft.nl if you are not explicitly asked to do so.

References

[1] Dominik Deschner, Florian Lang, and Freimut Bodendorf. Strategies for software agent based multiple issue negotiations. In *Proceedings of the Second International Conference on Electronic Commerce and Web Technologies*, pages 206–215. Springer-Verlag, 2001.

[2] 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.

[3] Catholijn M. Jonker and Jan Treur. An agent architecture for multi-attribute negotiation. In *IJCAI*, pages 1195–1201, 2001. URL = citeseer.ist.psu.edu/jonker01agent.html, November, 2006.

[4] http://en.wikipedia.org/wiki/negotiation. November 2006.

[5] Iyad Rahwan, Peter McBurney, and Liz Sonenberg. Towards a theory of negotiation strategy (a preliminary report). In S. Parsons and P. Gmytrasiewicz, editors, *Proceedings of the 5th Workshop on Game Theoretic and Decision Theoretic Agents (GTDT-2003)*, pages 73–80, 2003. URL = http://citeseer.ist.psu.edu/rahwan03towards.html, November, 2006.

[6] Howard Raiffa. *Lectures on Negotiation Analysis*. PON Books, 1996.

[7] Howard Raiffa. *The Art and Science of Negotiation*. Belknap Press, reprint edition, 2005.

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

[9] Roberto Serrano. Bargaining, 2005. URL = http://www.econ.brown.edu/faculty/serrano/pdfs/2005PalBarg.pdf, November, 2006.